
Alliance Auth Documentation

R4stl1n

Jul 19, 2022

CONTENTS

1	Installation	3
1.1	Alliance Auth	3
1.2	NGINX	9
1.3	Apache	12
1.4	Gunicorn	14
1.5	Upgrading Python 3	16
2	Features	21
2.1	Overview	21
2.2	Core Features	21
2.3	Services	36
2.4	Apps	67
2.5	Community Contributions	85
3	Maintenance	87
3.1	App Maintenance	87
3.2	Folder structure	88
3.3	Troubleshooting	89
3.4	Tuning	91
4	Support	95
5	Customizing	97
5.1	Site name	97
5.2	Custom Static and Templates	97
5.3	Custom URLs and Views	98
6	Development	99
6.1	Custom apps and services	99
6.2	Developing AA Core	110
6.3	Setup dev environment for AA	111
6.4	Developing apps	120
7	Contributing	145
7.1	Publish a new community app or service	145
7.2	Help to maintain an existing community app or service	145
7.3	Help with improving Auth documentation	145
7.4	Help with support questions on Discord	146
7.5	Help to improve Alliance Auth core	146
7.6	Additional Resources	146

Python Module Index	147
Index	149

Welcome to the official documentation for **Alliance Auth**!

Alliance Auth

English (en) ▾

Night

Logout

Dashboard

Groups

Services

Corporation Stats

Fleet Activity Tracking

Applications

Fleet Operations

Ship Replacement

Structure Timers

Dashboard

Main Character

Sodi Pops Distribution Center

Same Great Taste

Add Character

Change Main

Groups

FC

Leadership

Characters

	Name	Corp	Alliance
	Emily Kasenumi	24th Imperial Crusade	None
	Yuna Kobayashi	Sodi Pops Distribution Center	Same Great Taste

Alliance Auth is a web site that helps Eve Online organizations efficiently manage access to applications and external services.

CONTENTS

1

INSTALLATION

This chapter contains the main installation guides for **Alliance Auth**.

In addition to main guide for installation Alliance Auth you also find guides for configuring web servers (Apache, NGINX) and the recommended WSGI server (Gunicorn).

1.1 Alliance Auth

This document describes how to install **Alliance Auth** from scratch.

Tip: If you are uncomfortable with Linux permissions follow the steps below as the root user.

Note: There are additional installation steps for activating services and apps that come with **Alliance Auth**. Please see the page for the respective service or apps in chapter *Features* for details.

1.1.1 Dependencies

Operating System

Alliance Auth can be installed on any Unix like operating system. Dependencies are provided below for two of the most popular Linux platforms: Ubuntu and CentOS. To install on your favorite flavour of Linux, identify and install equivalent packages to the ones listed here.

Python

Alliance Auth requires Python 3.7 or higher. Ensure it is installed on your server before proceeding.

Ubuntu 1604 1804:

Note: Ubuntu 2004 ships with Python 3.8, No updates required.

```
add-apt-repository ppa:deadsnakes/ppa
```

```
apt-get update
```

```
apt-get install python3.7 python3.7-dev python3.7-venv
```

CentOS 7/8:

```
cd ~
```

```
sudo yum install gcc openssl-devel bzip2-devel libffi-devel wget
```

```
wget https://www.python.org/ftp/python/3.7.11/Python-3.7.11.tgz
```

```
tar xvf Python-3.7.11.tgz
```

```
cd Python-3.7.11/
```

```
./configure --enable-optimizations --enable-shared
```

```
make altinstall
```

Database

It's recommended to use a database service instead of SQLite. Many options are available, but this guide will use MariaDB.

Warning: Many Ubuntu distributions come with an older version of Maria DB, which is not compatible with **Alliance Auth**. You need Maria DB 10.3 or higher!

For instructions on how To install a newer version of Maria DB on Ubuntu visit this page: [MariaDB Repositories](#).

Ubuntu:

```
apt-get install mariadb-server mariadb-client libmysqlclient-dev
```

CentOS:

```
yum install mariadb-server mariadb-devel mariadb-shared mariadb
```

Note: If you don't plan on running the database on the same server as auth you still need to install the libmysqlclient-dev package on Ubuntu or mariadb-devel package on CentOS.

Redis and Other Tools

A few extra utilities are also required for installation of packages.

Ubuntu:

```
apt-get install unzip git redis-server curl libssl-dev libbz2-dev libffi-dev
```

CentOS:


```
yum install gcc gcc-c++ unzip git redis curl bzip2-devel
```

Important: CentOS: Make sure Redis is running before continuing.

```
systemctl enable redis.service  
systemctl start redis.service
```

1.1.2 Database Setup

Alliance Auth needs a MySQL user account and database. Open an SQL shell with `mysql -u root -p` and create them as follows, replacing `PASSWORD` with an actual secure password:

```
CREATE USER 'allianceserver'@'localhost' IDENTIFIED BY 'PASSWORD';  
CREATE DATABASE alliance_auth CHARACTER SET utf8mb4;  
GRANT ALL PRIVILEGES ON alliance_auth . * TO 'allianceserver'@'localhost';
```

Once your database is set up, you can leave the SQL shell with `exit`.

Add timezone tables to your mysql installation:

```
mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root -p mysql
```

Note: You may see errors when you add the timezone tables. To make sure that they were correctly added run the following commands and check for the `time_zone` tables:

```
mysql -u root -p  
use mysql;  
show tables;
```

Close the SQL shell and secure your database server with this command:

```
mysql_secure_installation
```

1.1.3 Auth Install

User Account

For security and permissions, it's highly recommended you create a separate user to install auth under. Do not log in as this account.

Ubuntu:

```
adduser --disabled-login allianceserver
```

CentOS:

```
useradd -s /bin/nologin allianceserver
```

Virtual Environment

Create a Python virtual environment and put it somewhere convenient (e.g. `/home/allianceserver/venv/auth/`)

```
python3 -m venv /home/allianceserver/venv/auth/
```

Warning: The `python3` command may not be available on all installations. Try a specific version such as `python3.7` if this is the case.

Tip: A virtual environment provides support for creating a lightweight “copy” of Python with their own site directories. Each virtual environment has its own Python binary (allowing creation of environments with various Python versions) and can have its own independent set of installed Python packages in its site directories. You can read more about virtual environments on the [Python docs](#).

Activate the virtual environment with (Note the `/bin/activate` on the end of the path):

```
source /home/allianceserver/venv/auth/bin/activate
```

Hint: Each time you come to do maintenance on your Alliance Auth installation, you should activate your virtual environment first. When finished, deactivate it with the `deactivate` command.

Eve Online SSO

You need to have a dedicated Eve SSO app for Alliance auth. Please go to [EVE Developer](#) to create one.

For **scopes** your SSO app needs to have at least `publicData`. Additional scopes depends on which Alliance Auth apps you will be using. For convenience, we recommend adding all available ESO scopes to your SSO app. Note that Alliance Auth will always ask the users to approve specific scopes before they are used.

As **callback URL** you want to define the URL of your Alliance Auth site plus the route: `/sso/callback`. Example for a valid callback URL: `https://auth.example.com/sso/callback`

In `local.py` you will need to set `ESI_USER_CONTACT_EMAIL` to an email address to ensure that CCP has reliable contact information for you.

Alliance Auth Project

Ensure wheel is available before continuing:

```
pip install wheel
```

You can install **Alliance Auth** with the following command. This will install AA and all its Python dependencies.

```
pip install allianceauth
```

You should also install Gunicorn now unless you want to use another WSGI server (see [Gunicorn](#) for details):

```
pip install gunicorn
```

Now you need to create the application that will run the **Alliance Auth** install. Ensure you are in the `allianceserver` home directory by issuing:

```
cd /home/allianceserver
```

The following command bootstraps a Django project which will run your **Alliance Auth** instance. You can rename it from `myauth` to anything you'd like. Note that this name is shown by default as the site name but that can be changed later.

```
allianceauth start myauth
```

The settings file needs configuring. Edit the template at `myauth/myauth/settings/local.py`. Be sure to configure the EVE SSO and Email settings.

Django needs to install models to the database before it can start.

```
python /home/allianceserver/myauth/manage.py migrate
```

Now we need to round up all the static files required to render templates. Make a directory to serve them from and populate it.

```
mkdir -p /var/www/myauth/static
python /home/allianceserver/myauth/manage.py collectstatic
```

Check to ensure your settings are valid.

```
python /home/allianceserver/myauth/manage.py check
```

Finally, ensure the `allianceserver` user has read/write permissions to this directory before proceeding.

```
chown -R allianceserver:allianceserver /home/allianceserver/myauth
```

1.1.4 Services

Alliance Auth needs some additional services to run, which we will set up and configure next.

Gunicorn

To run the **Alliance Auth** website a [WSGI Server](#) is required. For this [Gunicorn](#) is highly recommended for its ease of configuring. It can be manually run from within your `myauth` base directory with `gunicorn --bind 0.0.0.0 myauth.wsgi` or automatically run using Supervisor.

The default configuration is good enough for most installations. Additional information is available in the [gunicorn](#) doc.

Supervisor

Supervisor is a process watchdog service: it makes sure other processes are started automatically and kept running. It can be used to automatically start the WSGI server and Celery workers for background tasks. Installation varies by OS:

Note: Many package managers will install Supervisor 3 by default, which requires Python 2.

Ubuntu:

```
apt-get install supervisor
```

CentOS:

```
yum install supervisor
systemctl enable supervisord.service
systemctl start supervisord.service
```

Once installed, it needs a configuration file to know which processes to watch. Your Alliance Auth project comes with a ready-to-use template which will ensure the Celery workers, Celery task scheduler and Gunicorn are all running.

Ubuntu:

```
ln -s /home/allianceserver/myauth/supervisor.conf /etc/supervisor/conf.d/myauth.conf
```

CentOS:

```
ln -s /home/allianceserver/myauth/supervisor.conf /etc/supervisord.d/myauth.ini
```

Activate it with `supervisorctl reload`.

You can check the status of the processes with `supervisorctl status`. Logs from these processes are available in `/home/allianceserver/myauth/log` named by process.

Note: Any time the code or your settings change you'll need to restart Gunicorn and Celery.

```
supervisorctl restart myauth:
```

1.1.5 Webserver

Once installed, decide on whether you're going to use **NGINX** or **Apache** and follow the respective guide.

Note that Alliance Auth is designed to run with web servers on HTTPS. While running on HTTP is technically possible, it is not recommended for production use, and some functions (e.g. Email confirmation links) will not work properly.

1.1.6 Superuser

Before using your auth site, it is essential to create a superuser account. This account will have all permissions in Alliance Auth. It's OK to use this as your personal auth account.

```
python /home/allianceserver/myauth/manage.py createsuperuser
```

The superuser account is accessed by logging in via the admin site at <https://example.com/admin>.

If you intend to use this account as your personal auth account you need to add a main character. Navigate to the normal user dashboard (at <https://example.com>) after logging in via the admin site and select **Change Main**. Once a main character has been added, it is possible to use SSO to login to this account.

1.1.7 Updating

Periodically [new releases](#) are issued with bug fixes and new features. Be sure to read the [release notes](#) which will highlight changes.

To update your install, simply activate your virtual environment and update with:

```
pip install --upgrade allianceauth
```

Some releases come with changes to the base settings. Update your project's settings with:

```
allianceauth update /home/allianceserver/myauth
```

Some releases come with new or changed models. Update your database to reflect this with:

```
python /home/allianceserver/myauth/manage.py migrate
```

Finally, some releases come with new or changed static files. Run the following command to update your static files folder:

```
python /home/allianceserver/myauth/manage.py collectstatic
```

Always restart AA, Celery and Gunicorn after updating:

```
supervisorctl restart myauth:
```

1.2 NGINX

1.2.1 Overview

Nginx (engine x) is a HTTP server known for its high performance, stability, simple configuration, and low resource consumption. Unlike traditional servers (i.e. Apache), Nginx doesn't rely on threads to serve requests, rather using an asynchronous event driven approach which permits predictable resource usage and performance under load.

If you're trying to cram Alliance Auth into a very small VPS of say, 1-2GB or less, then Nginx will be considerably friendlier to your resources compared to Apache.

You can read more about NGINX on the [NGINX wiki](#).

1.2.2 Coming from Apache

If you're converting from Apache, here are some things to consider.

Nginx is lightweight for a reason. It doesn't try to do everything internally and instead concentrates on just being a good HTTP server. This means that, unlike Apache, it won't automatically run PHP scripts via `mod_php` and doesn't have an internal WSGI server like `mod_wsgi`. That doesn't mean that it can't, just that it relies on external processes to run these instead. This might be good or bad depending on your outlook. It's good because it allows you to segment your applications, restarting Alliance Auth won't impact your PHP applications. On the other hand it means more config and more management of services. For some people it will be worth it, for others losing the centralised nature of Apache may not be worth it.

Apache	Nginx Replacement
<code>mod_php</code>	<code>php5-fpm</code> or <code>php7-fpm</code> (PHP FastCGI)
<code>mod_wsgi</code>	Gunicorn or other external WSGI server

Your `.htaccess` files won't work. Nginx has a separate way of managing access to folders via the server config. Everything you can do with `htaccess` files you can do with Nginx config. [Read more on the Nginx wiki](#)

1.2.3 Setting up Nginx

Install Nginx via your preferred package manager or other method. If you need help just search, there are plenty of guides on installing Nginx out there.

Nginx needs to be able to read the folder containing your auth project's static files. `chown -R nginx:nginx /var/www/myauth/static`.

Tip: Some specific distros may use `www-data:www-data` instead of `nginx:nginx`, causing static files (images, stylesheets etc) not to appear. You can confirm what user Nginx will run under by checking either its base config file `/etc/nginx/nginx.conf` for the "user" setting, or once Nginx has started `ps aux | grep nginx`. Adjust your `chown` commands to the correct user if needed.

You will need to have [Gunicorn](#) or some other WSGI server setup for hosting Alliance Auth.

Ubuntu

Create a config file in `/etc/nginx/sites-available` and call it `alliance-auth.conf` or whatever your preferred name is.

Create a symbolic link to enable the site `ln -s /etc/nginx/sites-available/alliance-auth.conf /etc/nginx/sites-enabled/`

CentOS

Create a config file in `/etc/nginx/conf.d` and call it `alliance-auth.conf` or whatever your preferred name is.

Basic config

Copy this basic config into your config file. Make whatever changes you feel are necessary.

```
server {
    listen 80;
    server_name example.com;

    location = /favicon.ico { access_log off; log_not_found off; }

    location /static {
        alias /var/www/myauth/static;
        autoindex off;
    }

    location /robots.txt {
        alias /var/www/myauth/static/robots.txt;
    }

    # Unicorn config goes below
    location / {
        include proxy_params;
        proxy_pass http://127.0.0.1:8000;
    }
}
```

Restart Nginx after making changes to the config files. On Ubuntu `service nginx restart` and on CentOS `systemctl restart nginx.service`.

Adding TLS/SSL

With [Let's Encrypt](#) offering free SSL certificates, there's no good reason to not run HTTPS anymore. The bot can automatically configure Nginx on some operating systems. If not proceed with the manual steps below.

Your config will need a few additions once you've got your certificate.

```
listen 443 ssl http2; # Replace listen 80; with this

ssl_certificate      /path/to/your/cert.crt;
ssl_certificate_key  /path/to/your/cert.key;

ssl on;
ssl_session_cache    builtin:1000 shared:SSL:10m;
ssl_protocols        TLSv1 TLSv1.1 TLSv1.2;
ssl_ciphers           ↪EECDH+ECDSA+AESGCM:EECDH+aRSA+AESGCM:EECDH+ECDSA+SHA384:EECDH+ECDSA+SHA256:EECDH+aRSA+SHA384:EECDH+
↪aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS;
ssl_prefer_server_ciphers on;
```

If you want to redirect all your non-SSL visitors to your secure site, below your main configs `server` block, add the following:

```
server {
    listen 80;
    server_name example.com;

    # Redirect all HTTP requests to HTTPS with a 301 Moved Permanently response.
    return 301 https://$host$request_uri;
}
```

If you have trouble with the `ssl_ciphers` listed here or some other part of the SSL config, try getting the values from [Mozilla's SSL Config Generator](#).

1.3 Apache

1.3.1 Overview

Alliance Auth gets served using a Web Server Gateway Interface (WSGI) script. This script passes web requests to Alliance Auth which generates the content to be displayed and returns it. This means very little has to be configured in Apache to host Alliance Auth.

If you're using a small VPS to host services with very limited memory, consider using [NGINX](#).

1.3.2 Installation

Ubuntu:

```
apt-get install apache2
```

CentOS:

```
yum install httpd
systemctl enable httpd
systemctl start httpd
```

1.3.3 Configuration

Apache needs to be able to read the folder containing your auth project's static files. On Ubuntu: `chown -R www-data:www-data /var/www/myauth/static`, and on CentOS: `chown -R apache:apache /var/www/myauth/static`

Apache serves sites through defined virtual hosts. These are located in `/etc/apache2/sites-available/` on Ubuntu and `/etc/httpd/conf.d/httpd.conf` on CentOS.

A virtual host for auth need only proxy requests to your WSGI server (Gunicorn if you followed the install guide) and serve static files. Examples can be found below. Create your config in its own file e.g. `myauth.conf`

Ubuntu

To proxy and modify headers a few mods need to be enabled.

```
a2enmod proxy
a2enmod proxy_http
a2enmod headers
```

Create a new config file for auth e.g. `/etc/apache2/sites-available/myauth.conf` and fill out the virtual host configuration. To enable your config use `a2ensite myauth.conf` and then reload apache with `service apache2 reload`.

CentOS

Place your virtual host configuration in the appropriate section within `/etc/httpd/conf.d/httpd.conf` and restart the httpd service with `systemctl restart httpd`.

1.3.4 Sample Config File

```
<VirtualHost *:80>
    ServerName auth.example.com

    ProxyPassMatch ^/static !
    ProxyPassMatch ^/robots.txt !

    ProxyPass / http://127.0.0.1:8000/
    ProxyPassReverse / http://127.0.0.1:8000/
    ProxyPreserveHost On

    Alias "/static" "/var/www/myauth/static"
    Alias "/robots.txt" "/var/www/myauth/static/robots.txt"

    <Directory "/var/www/myauth/static">
        Require all granted
    </Directory>

    <Location "/robots.txt">
        SetHandler None
        Require all granted
    </Location>
</VirtualHost>
```

1.3.5 SSL

It's 2018 - there's no reason to run a site without SSL. The EFF provides free, renewable SSL certificates with an automated installer. Visit their [website](#) for information.

After acquiring SSL the config file needs to be adjusted. Add the following lines inside the `<VirtualHost>` block:

```
RequestHeader set X-FORWARDED-PROTOCOL https
RequestHeader set X-FORWARDED-SSL On
```

1.4 Gunicorn

Gunicorn is a Python WSGI HTTP Server for UNIX. The Gunicorn server is light on server resources, and fairly speedy.

If you find Apache's `mod_wsgi` to be a headache or want to use NGINX (or some other webserver), then Gunicorn could be for you. There are a number of other WSGI server options out there and this documentation should be enough for you to piece together how to get them working with your environment.

Check out the full [Gunicorn docs](#).

Note: The page contains additional steps on how to setup and configure Gunicorn that are not required for users who decide to stick with the default Gunicorn configuration as described in the main installation guide for AA.

1.4.1 Setting up Gunicorn

Note: If you're using a virtual environment, activate it now. `source /path/to/venv/bin/activate`.

Install Gunicorn using `pip`, `pip install gunicorn`.

In your `myauth` base directory, try running `gunicorn --bind 0.0.0.0:8000 myauth.wsgi`. You should be able to browse to `http://yourserver:8000` and see your Alliance Auth installation running. Images and styling will be missing, but don't worry, your web server will provide them.

Once you validate its running, you can kill the process with `Ctrl+C` and continue.

1.4.2 Running Gunicorn with Supervisor

You should use [Supervisor](#) to keep all of Alliance Auth components running (instead of using `screen`). You don't *have* to but we will be using it to start and run Gunicorn so you might as well.

Sample Supervisor config

You'll want to edit `/etc/supervisor/conf.d/myauth.conf` (or whatever you want to call the config file)

```
[program:gunicorn]
user = allianceserver
directory=/home/allianceserver/myauth/
command=/home/allianceserver/venv/auth/bin/gunicorn myauth.wsgi --workers=3 --timeout_
↪120
stdout_logfile=/home/allianceserver/myauth/log/gunicorn.log
stderr_logfile=/home/allianceserver/myauth/log/gunicorn.log
autostart=true
autorestart=true
stopsignal=INT
```

- `[program:gunicorn]` - Change `gunicorn` to whatever you wish to call your process in Supervisor.
- `user = allianceserver` - Change to whatever user you wish Gunicorn to run as. You could even set this as `allianceserver` if you wished. I'll leave the question security of that up to you.
- `directory=/home/allianceserver/myauth/` - Needs to be the path to your Alliance Auth project.

- `command=/home/allianceserver/venv/auth/bin/gunicorn myauth.wsgi`
`--workers=3 --timeout 120` - Running Gunicorn and the options to launch with. This is where you have some decisions to make, we'll continue below.

Gunicorn Arguments

See the [Commonly Used Arguments](#) or [Full list of settings](#) for more information.

Where to bind Gunicorn to

What address are you going to use to reference it? By default, without a bind parameter, Gunicorn will bind to `127.0.0.1:8000`. This might be fine for your application. If it clashes with another application running on that port you will need to change it. I would suggest using UNIX sockets too, if you can.

For UNIX sockets add `--bind=unix:/run/allianceauth.sock` (or to a path you wish to use). Remember that your web server will need to be able to access this socket file.

For a TCP address add `--bind=127.0.0.1:8001` (or to the address/port you wish to use, but I would strongly advise against binding it to an external address).

Whatever you decide to use, remember it because we'll need it when configuring your webserver.

Number of workers

By default Gunicorn will spawn only one worker. The number you set this to will depend on your own server environment, how many visitors you have etc. Gunicorn suggests $(2 \times \$\text{num_cores}) + 1$ for the number of workers. So for example if you have 2 cores you want $2 \times 2 + 1 = 5$ workers. See [here](#) for the official discussion on this topic.

Change it by adding `--workers=5` to the command.

Running with a virtual environment

If you're running with a virtual environment, you'll need to add the path to the `command=` config line.

e.g. `command=/path/to/venv/bin/gunicorn myauth.wsgi`

The example config is using the `myauth` venv from the main installation guide: `command=/home/allianceserver/venv/auth/bin/gunicorn myauth.wsgi`

Starting via Supervisor

Once you have your configuration all sorted, you will need to reload your supervisor config `service supervisor reload` and then you can start the Gunicorn server via `supervisorctl start myauth:gunicorn` (or whatever you renamed it to). You should see something like the following `myauth-gunicorn: started`. If you get some other message, you'll need to consult the Supervisor log files, usually found in `/var/log/supervisor/`.

1.4.3 Configuring your webserver

Any web server capable of proxy passing should be able to sit in front of Gunicorn. Consult their documentation armed with your `--bind=` address and you should be able to find how to do it relatively easy.

1.4.4 Restarting Gunicorn

In the past when you made changes you restarted the entire Apache server. This is no longer required. When you update or make configuration changes that ask you to restart Apache, instead you can just restart Gunicorn:

```
supervisorctl restart gunicorn, or the service name you chose for it.
```

1.5 Upgrading Python 3

This guide describes how to upgrade an existing Alliance Auth (AA) installation to a newer Python 3 version.

Hint: In accordance with the installation guide we will assume you perform all actions as root. If you are not running as root you need to add `sudo` to some commands.

Note: This guide will upgrade the software components only but not change any data or configuration.

1.5.1 Install a new Python version

To run AA with a newer Python 3 version than your system's default you need to install it first. Technically it would be possible to upgrade your system's default Python 3, but since many of your system's tools have been tested to work with that specific version we would not recommend it. Instead we recommend to install an additional Python 3 version alongside your default version and use that for AA.

Note: For stability and performance we currently recommend to run AA with Python 3.7. It has proven to be the fastest and most stable version in use currently.

To install other Python versions than those included with your distribution, you need to add a new installation repository. Then you can install the specific Python 3 to your system.

Ubuntu 1604 1804:

Note: Ubuntu 2004 ships with Python 3.8, No updates required.

```
add-apt-repository ppa:deadsnakes/ppa
```

```
apt-get update
```

```
apt-get install python3.7 python3.7-dev python3.7-venv
```

CentOS 7/8:

```
cd ~
```

```
sudo yum install gcc openssl-devel bzip2-devel libffi-devel wget
```

```
wget https://www.python.org/ftp/python/3.7.11/Python-3.7.11.tgz
```

```
tar xvf Python-3.7.11.tgz
```

```
cd Python-3.7.11/
```

```
./configure --enable-optimizations --enable-shared
```

```
make altinstall
```

1.5.2 Preparing your venv

Before updating your venv it is important to make sure that your current installation is stable. Otherwise your new venv might not be consistent with your data, which might create problems.

Start by navigating to your main project folder (the one that has `manage.py` in it). If you followed the default installation the path is: `/home/allianceserver/myauth`

Activate your venv:

```
source /home/allianceserver/venv/auth/bin/activate
```

Upgrade AA

Make sure to upgrade AA to the newest version:

```
pip install -U allianceauth
```

Run migrations and collectstatic.

```
python manage.py migrate
```

```
python manage.py collectstatic
```

Restart your AA supervisor:

```
supervisorctl restart myauth:
```

Upgrade your apps

You also need to upgrade all additional apps to their newest version that you have installed. And you need to make sure that you can reinstall all your apps later, e.g. you know from which repo they came. We recommend to make a list of all your apps, so you can just go through them later when you rebuild your venv.

If you unsure which apps you have installed from repos check `INSTALLED_APPS` in your settings. Alternatively run this command to get a list all apps in your venv.

```
pip list
```

```
python manage.py migrate
```

Make sure to run migrations and collect static files for all upgraded apps.

Restart and final check

Do a final restart of your AA supervisors and make sure your installation is still running normally.

For a final check that they are no issues - e.g. any outstanding migrations - run this command:

```
python manage.py check
```

If you get the following result you are good to go. Otherwise make sure to fix any issues first before proceeding.

```
System check identified no issues (0 silenced).
```

1.5.3 Backup current venv

Make sure you are in your venv!

First we create a list of all installed packages in your venv. You can use this list later as reference to see what packages should be installed.

```
pip freeze > requirements.txt
```

At this point we recommend creating a list of the additional packages that you need to manually reinstall later on top of AA:

- Community AA apps (e.g. aa-structures)
- Additional tools you are using (e.g. flower, django-extensions)

Hint: While *requirements.txt* will contain a complete list of your packages, it will also contain many packages that are automatically installed as dependencies and don't need be manually reinstalled.

Note: Some guide on the Internet will suggest to use the *requirements.txt* file to recreate a venv. This is indeed possible, but only works if all packages can be installed from PyPI. Since most community apps are installed directly from repos this guide will not follow that approach.

Leave the venv and shutdown all AA services:

```
deactivate
```

```
supervisorctl stop myauth:
```

Rename and keep your old venv so we have a fallback in case of some unforeseeable issues:

```
mv /home/allianceserver/venv/auth /home/allianceserver/venv/auth_old
```

1.5.4 Create your new venv

Now let's create our new venv with Python 3.7 and activate it:

```
python3.7 -m venv /home/allianceserver/venv/auth
```

```
source /home/allianceserver/venv/auth/bin/activate
```

1.5.5 Reinstall packages

Now we need to reinstall all packages into your new venv.

Install basic packages

```
pip install --upgrade pip
```

```
pip install --upgrade setuptools
```

```
pip install wheel
```

Installing AA & Gunicorn

```
pip install allianceauth
```

```
pip install gunicorn
```

Install all other packages

Last, but not least you need to reinstall all other packages, e.g. for AA community apps or additional tools.

Use the list of packages you created earlier as a checklist. Alternatively you use the `requirements.txt` file we created earlier to see what you need. During the installation process you can run `pip list` to see what you already got installed.

To check whether you are missing any apps you can also run the check command:

```
python manage.py check
```

Note: In case you forget to install an app you will get this error

```
ModuleNotFoundError: No module named 'xyz'
```

Note that you should not need to run any migrations unless you forgot to upgrade one of your existing apps or you got the newer version of an app through a dependency. In that case you just migrations normally.

1.5.6 Restart

After you have completed installing all packages just start your AA supervisor again.

```
supervisorctl start myauth:
```

We recommend to keep your old venv copy for a couple of days so you have a fallback just in case. After that you should be fine to remove it.

1.5.7 Fallback

In case you run into any major issue you can always switch back to your initial venv.

Before you start double-check that you still have your old venv for auth:

```
ls /home/allianceserver/venv/auth /home/allianceserver/venv
```

If the output shows these two folders you should be safe to proceed:

- auth
- auth_old

Run these commands to remove your current venv and switch back to the old venv for auth:

```
supervisorctl stop myauth:
```

```
rm -rf /home/allianceserver/venv/auth
```

```
mv /home/allianceserver/venv/auth_old /home/allianceserver/venv/auth
```

```
supervisorctl start myauth:
```


FEATURES

Learn about the features of **Alliance Auth** and how to install and use them.

2.1 Overview

Alliance Auth (AA) is a web site that helps Eve Online organizations efficiently manage access to applications and external services.

It has the following key features:

- Automatically grants or revokes users access to external services (e.g. Discord, Mumble) and web apps (e.g. SRP requests) based on the user's current membership to *in-game organizations* and *groups*
- Provides a central web site where users can directly access web apps (e.g. SRP requests) and manage their access to external services and groups.
- Includes a set of connectors (called "*services*") for integrating access management with many popular external services like Discord, Mumble, Teamspeak 3, SMF and others
- Includes a set of web *apps* which add many useful functions, e.g.: fleet schedule, timer board, SRP request management, fleet activity tracker
- Can be easily extended with additional services and apps. Many are provided by the *community*.
- Chinese, English, German and Spanish localization

2.2 Core Features

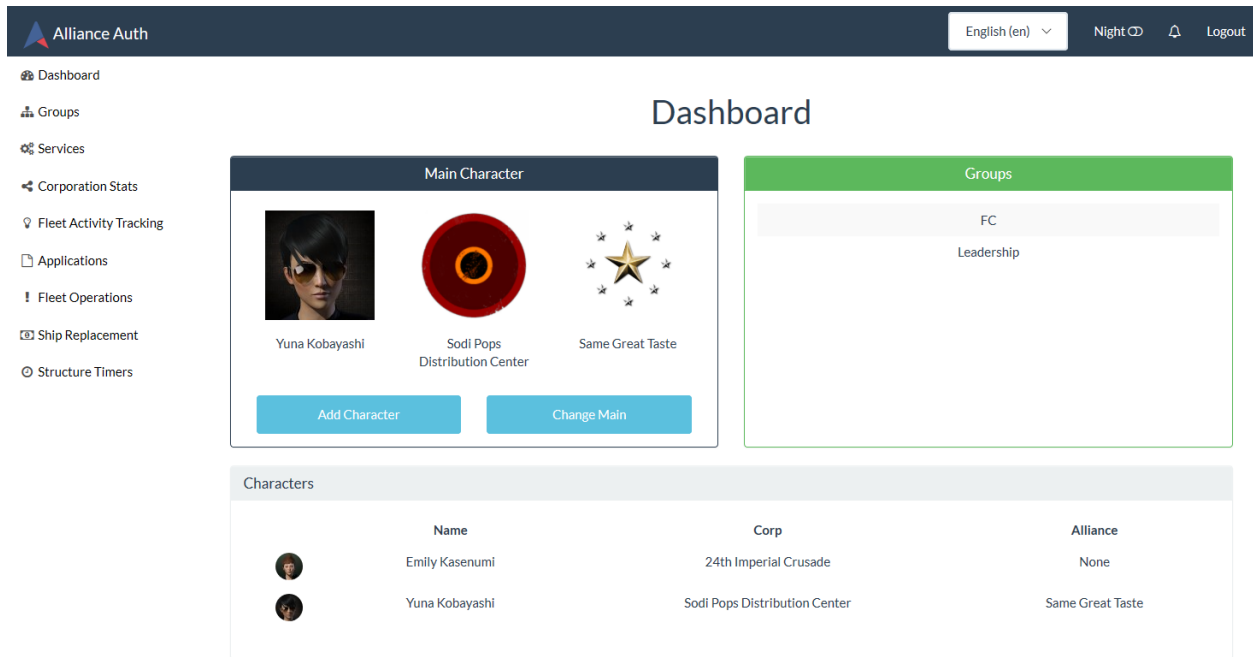
Managing access to applications and services is one of the core functions of **Alliance Auth**. The related key concepts and functionalities are describes in this section.

2.2.1 Dashboard

The dashboard is the main page of the **Alliance Auth** website and the first page every logged in user will see.

The content of the dashboard is specific to the logged in user. It has a sidebar, which will display the list of apps a user currently as access to based on his permissions. And it also shows which character the user has registered and to which group he belongs.

For admin users the dashboard shows additional technical information about the AA instance.



Settings

Here is a list of available settings for the dashboard. They can be configured by adding them to your AA settings file (`local.py`). Note that all settings are optional and the app will use the documented default settings if they are not used.

Name	Description	De- fault
<code>ALLIANCEAUTH_DASHBOARD_TASKS_MAX_HOURS</code>	Statistics will be calculated for task events not older than max hours.	24
<code>ALLIANCEAUTH_DASHBOARD_TASK_STATISTICS_DISABLE_RECORD</code>	Disables recording of task statistics. Used mainly in development.	False

2.2.2 States

States define the basic role of a user based on his affiliation with your organization. A user that has a character in your organization (e.g. alliance) will usually have the `Member` state. And a user, that has no characters in your organization will usually have the `Guest` state.

States are assigned and updated automatically. So a user which character just left your organization will automatically loose his `Member` state and get the `Guest` state instead.

The main purpose of states like `Member` is to have one place where you can assign all permissions that should apply to all users with that particular state. For example if all your members should have access to the SRP app you would add the permission that gives access to the SRP app to the `Member` state.

Creating a State

States are created through your installation's admin site. Upon install three states are created for you: `Member`, `Blue`, and `Guest`. New ones can be created like any other Django model by users with the appropriate permission (`authentication | state | Can add state`) or superusers.

A number of fields are available and are described below.

Name

This is the displayed name of a state. Should be self-explanatory.

Permissions

This lets you select permissions to grant to the entire state, much like a group. Any user with this state will be granted these permissions.

A common use case would be granting service access to a state.

Priority

This value determines the order in which states are applied to users. Higher numbers come first. So if a random user Bob could member of both the `Member` and `Blue` states, because `Member` has a higher priority Bob will be assigned to it.

Public

Checking this box means this state is available to all users. There isn't much use for this outside the `Guest` state.

Member Characters

This lets you select which characters the state is available to. Characters can be added by selecting the green plus icon.

Member Corporations

This lets you select which Corporations the state is available to. Corporations can be added by selecting the green plus icon.

Member Alliances

This lets you select which Alliances the state is available to. Alliances can be added by selecting the green plus icon.

Member Factions

This lets you select which factions the state is available to. Factions can be added by selecting the green plus icon, and are limited to those which can be enlisted in for faction warfare.

Determining a User's State

States are mutually exclusive, meaning a user can only be in one at a time.

Membership is determined based on a user's main character. States are tested in order of descending priority - the first one which allows membership to the main character is assigned to the user.

States are automatically assigned when a user registers to the site, their main character changes, they are activated or deactivated, or states are edited. Note that editing states triggers lots of state checks so it can be a very slow process.

Assigned states are visible in the `Users` section of the `Authentication` admin site.

The Guest State

If no states are available to a user's main character, or their account has been deactivated, they are assigned to a catch-all `Guest` state. This state cannot be deleted nor can its name be changed.

The `Guest` state allows permissions to be granted to users who would otherwise not get any. For example access to public services can be granted by giving the `Guest` state a service access permission.

2.2.3 Groups

Group Management is one of the core tasks of Alliance Auth. Many of Alliance Auth's services allow for synchronizing of group membership, allowing you to grant permissions or roles in services to access certain aspects of them.

Creating groups

Administrators can create custom groups for users to join. Examples might be groups like `Leadership`, `CEO` or `Scouts`.

When you create a `Group` additional settings are available beyond the normal Django group model. The admin page looks like this:

Home / Group Management / Groups / Recruiters

Change group
HISTORY

Name: Recruiters

Permissions:

Available permissions ⓘ

Filter

admin | log entry | Can add log entry
admin | log entry | Can change log entry
admin | log entry | Can delete log entry
auth | group | Can add group
auth | group | Can change group
auth | group | Can delete group
auth | permission | Can add permission
auth | permission | Can change permission
auth | user | Can add user
auth | user | Can change user

Choose all ⓘ

Chosen permissions ⓘ

auth | user | human_resources

Remove all

Hold down "Control", or "Command" on a Mac, to select more than one.

AUTH SETTINGS

: Recruiters

Description: People who screen applications.
Description of the group shown to users.

Group leaders:

AVAILABLE GROUP LEADERS ⓘ

Filter

Choose all ⓘ

CHOSEN GROUP LEADERS ⓘ

edemof

Remove all

Group leaders can process group requests for this group specifically. Use the auth.group_management permission to allow a user to manage all groups. Hold down "Control", or "Command" on a Mac, to select more than one.

☐ Internal
Internal group, users cannot see, join or request to join this group.
Used for groups such as Members, Corp_*, Alliance_*, etc.
Overrides Hidden and Open options when selected.

☒ Hidden
Group is hidden from users but can still join with the correct link.

☐ Open
Group is open and users will be automatically added upon request.
If the group is not open users will need their request manually approved.

☐ Public
Group is public. Any registered user is able to join this group, with visibility based on the other options set for this group.
Auth will not remove users from this group automatically when they are no longer authenticated.

Delete
Save and add another
Save and continue editing
SAVE

Here you have several options:

Internal

Users cannot see, join or request to join this group. This is primarily used for Auth's internally managed groups, though can be useful if you want to prevent users from managing their membership of this group themselves. This option will override the Hidden, Open and Public options when enabled.

By default, every new group created will be an internal group.

2.2. Core Features

25

Hidden

Group is hidden from the user interface, but users can still join if you give them the appropriate join link. The URL will be along the lines of `https://example.com/en/group/request_add/{group_id}`. You can get the Group ID from the admin page URL.

This option still respects the Open option.

Open

When a group is toggled open, users who request to join the group will be immediately added to the group.

If the group is not open, their request will have to be approved manually by someone with the group management role, or a group leader of that group.

Public

Group is accessible to any registered user, even when they do not have permission to join regular groups.

The key difference is that the group is completely unmanaged by Auth. **Once a member joins they will not be removed unless they leave manually, you remove them manually, or their account is deliberately set inactive or deleted.**

Most people won't have a use for public groups, though it can be useful if you wish to allow public access to some services. You can grant service permissions on a public group to allow this behavior.

Restricted

When a group is restricted only superuser admins can directly add or remove them to/from users. The purpose of this property is prevent staff admins from assigning themselves to groups that are security sensitive. The "restricted" property can be combined with all the other properties.

Reserved group names

When using Alliance Auth to manage external services like Discord, Auth will automatically duplicate groups on those services. E.g. on Discord Auth will create roles of the same name as groups. However, there may be cases where you want to manage groups on external services by yourself or by another bot. For those cases you can define a list of reserved group names. Auth will ensure that you can not create groups with a reserved name. You will find this list on the admin site under groupmanagement.

Note: While this feature can help to avoid naming conflicts with groups on external services, the respective service component in Alliance Auth also needs to be build in such a way that it knows how to prevent these conflicts. Currently only the Discord and Teamspeak3 services have this ability.

Managing groups

In order to access group management, users need to be either a superuser, granted the `auth | user | group_management` (Access to add members to groups within the alliance) permission or a group leader (discussed later).

Group Requests

When a user joins or leaves a group which is not marked as “Open”, their group request will have to be approved manually by a user with the `group_management` permission or by a group leader of the group they are requesting.

Group Membership







The group membership tab gives an overview of all of the non-internal groups.

Group Management

Group Requests

Group Membership

Groups

Name	Description	Status	Member Count	Action
Blackops	Drop it like its hot	Open	2	
Hidden Group		Hidden	0	
Open Group	An Open Group	Open	0	
Requestable Group		Requestable	1	
Test Group 2	It does things	Hidden	0	
Ts3 Test Group		Open	0	

Group Member Management



Clicking on the blue eye will take you to the group member management screen. Here you can see a list of people who are in the group, and remove members where necessary.

Group Management

Group Requests

Group Membership

Blackops Members

User	Character	Corp	Alliance	Action
basraah	Basraah	Ice Fire Warriors	Escalating Entropy	
basraah3				

Group Audit Log

Whenever a user Joins, Leaves, or is Removed from a group, this is logged. To find the audit log for a given group, click the light-blue button to the right of the Group Member Management (blue eye) button.

These logs contain the Date and Time the action was taken (in EVE/UTC), the user which submitted the request being acted upon (requestor), the user's main character, the type of request (join, leave or removed), the action taken (accept, reject or remove), and the user that took the action (actor).

Group Management Group Requests Group Membership

Example Group Audit Log

All times displayed are EVE/UTC.

Show 10 entries

Search:

Date/Time	Requestor	Main Character	Group	Type	Action	Actor
Dec. 8, 2018, 11:36 p.m.	root	Col Crunch	Example Group	Removed	Removed	root
Dec. 8, 2018, 11:36 p.m.	root	Col Crunch	Example Group	Leave	Reject	root
Dec. 8, 2018, 11:36 p.m.	root	Col Crunch	Example Group	Join	Accept	root

Showing 1 to 3 of 3 entries

Previous

1

Next

Group Leaders

Group leaders have the same abilities as users with the `group_management` permission, *however*, they will only be able to:

- Approve requests for groups they are a leader of.
- View the Group Membership and Group Members of groups they are leaders of.

This allows you to more finely control who has access to manage which groups.

Auto Leave

By default, in AA both requests and leaves for non-open groups must be approved by a group manager. If you wish to allow users to leave groups without requiring approvals, add the following lines to your `local.py`

```
## Allows users to freely leave groups without requiring approval.
GROUPMANAGEMENT_AUTO_LEAVE = True
```

Note: Before you set `GROUPMANAGEMENT_AUTO_LEAVE = True`, make sure there are no pending leave requests, as this option will hide the “Leave Requests” tab.

Settings

Here is a list of available settings for Group Management. They can be configured by adding them to your AA settings file (`local.py`). Note that all settings are optional and the app will use the documented default settings if they are not used.

Name	Description	De- fault
<code>GROUPMANAGEMENT_REQUESTS_NOTIFICATIONS</code>	Sends Auth notifications to all group leaders for join and leave requests.	False
<code>GROUPMANAGEMENT_AUTO_LEAVE</code>	Allows users to freely leave groups without requiring approval..	False

Permissions

In order to join a group other than a public group, the permission `groupmanagement.request_groups` (Can request non-public groups in the admin panel) must be active on their account, either via a group or directly applied to their User account.

When a user loses this permission, they will be removed from all groups *except* Public groups.

Note: By default, the `groupmanagement.request_groups` permission is applied to the `Member` group. In most instances this, and perhaps adding it to the `Blue` group, should be all that is ever needed. It is unsupported and NOT advisable to apply this permission to a public group. See #697 for more information.

Group Management should be mostly done using group leaders, a series of permissions are included below for thoroughness:

Permission	Admin Site	Auth Site
<code>auth.group_management</code>	None	Can Approve and Deny all Group Requests, Can view and manage all group memberships
<code>groupmanagement.request_groups</code>	None	Can Request Non-Public Groups

2.2.4 Analytics FAQ

Alliance Auth has an opt-out analytics module using [Google Analytics Measurement Protocol](#).

How to Opt-Out

Before you proceed, please read through this page and/or raise any concerns on the Alliance Auth discord. This data helps us make AA better.

To Opt-Out, modify our pre-loaded token using the Admin dashboard `*/admin/analytics/analyticstokens/1/change/`

Each of the three features Daily Stats, Celery Events and Page Views can be enabled/Disabled independently.

Alternatively, you can fully opt out of analytics with the following optional setting:

```
ANALYTICS_DISABLED = True
```

Change analytics tokens

Name:	AA Team Public Google Analytics (Universal)
Type:	Google Analytics Universal ▾
Token:	UA-186249766-2
<input checked="" type="checkbox"/> Send page views	
<input type="checkbox"/> Send celery tasks	
<input checked="" type="checkbox"/> Send stats	
Ignore paths:	<div><div></div><div>+</div></div> <div>Hold down "Control", or "Command" on a Mac, to select more than one.</div>

What

Alliance Auth has taken great care to anonymize the data sent. In order to identify *unique* installs we generate a UUIDv4, a random mathematical construct which does not contain any identifying information [UUID - UUID Objects](#)

Analytics comes pre-loaded with our Google Analytics Token, and the Three Types of task can be opted out independently. Analytics can also be loaded with your *own* GA token and the analytics module will act any/all tokens loaded.

Our Daily Stats contain the following:

- A phone-in task to identify a servers existence
- A task to send the Number of User models
- A task to send the Number of Token Models
- A task to send the Number of Installed Apps
- A task to send a List of Installed Apps
- Each Task contains the UUID and Alliance Auth Version

Our Celery Events contain the following:

- Unique Identifier (The UUID)
- Celery Namespace of the task eg allianceauth.eveonline
- Celery Task
- Task Success or Exception

- A context number for bulk tasks or sometimes a binary True/False

Our Page Views contain the following:

- Unique Identifier (The UUID)
- Page Path
- Page Title
- The locale of the users browser
- The User-Agent of the users browser
- The Alliance Auth Version

Why

This data allows Alliance Auth development to gather accurate statistics on our install base, as well as how those installs are used.

This allows us to better target our development time to commonly used modules and features and test them at the scales in use.

Where

This data is stored in a Team Google Analytics Dashboard. The Maintainers all have Management permissions here, and if you have contributed to the Alliance Auth project or third party applications feel free to ask in the Alliance Auth discord for access.

Using Analytics in my App

Analytics Event

analytics_event (*category: str, action: str, label: str, value: int = 0, event_type: str = 'Celery'*)

Send a Google Analytics Event for each token stored Includes check for if its enabled/disabled

Parameters

- **category** (*str*) – Celery Namespace
- **action** (*str*) – Task Name
- **label** (*str*) – Optional, Task Success/Exception
- **value** (*int*) – Optional, If bulk, Query size, can be a binary True/False
- **event_type** (*str*) – Optional, Celery or Stats only, Default to Celery

2.2.5 Notifications

Alliance Auth has a built-in notification system. The purpose of the notification system is to provide an easy and quick way to send messages to users of Auth. For example, some apps are using it to inform users about results after long running tasks have completed and admins will automatically get notifications about system errors.

The number of unread notifications is shown to the user in the top menu. And the user can click on the notification count to open the notifications app.

Notifications

Unread (0)

Read (4)

Mark All Read

Delete All Read

Timestamp	Title	Action
June 2, 2020, 1:22 p.m.	Alliance Structures: notifications updated for Rancid Rabid Rabis: OK	<div><div></div><div></div></div>
June 2, 2020, 1:05 p.m.	Alliance Structures: structures updated for Sodl Pops Distribution Center: OK	<div><div></div><div></div></div>
June 2, 2020, 1:05 p.m.	Alliance Structures: structures updated for Rancid Rabid Rabis: OK	<div><div></div><div></div></div>
May 31, 2020, 6:44 p.m.	ERROR [send_matching_to_webhook:678]	<div><div></div><div></div></div>

Settings

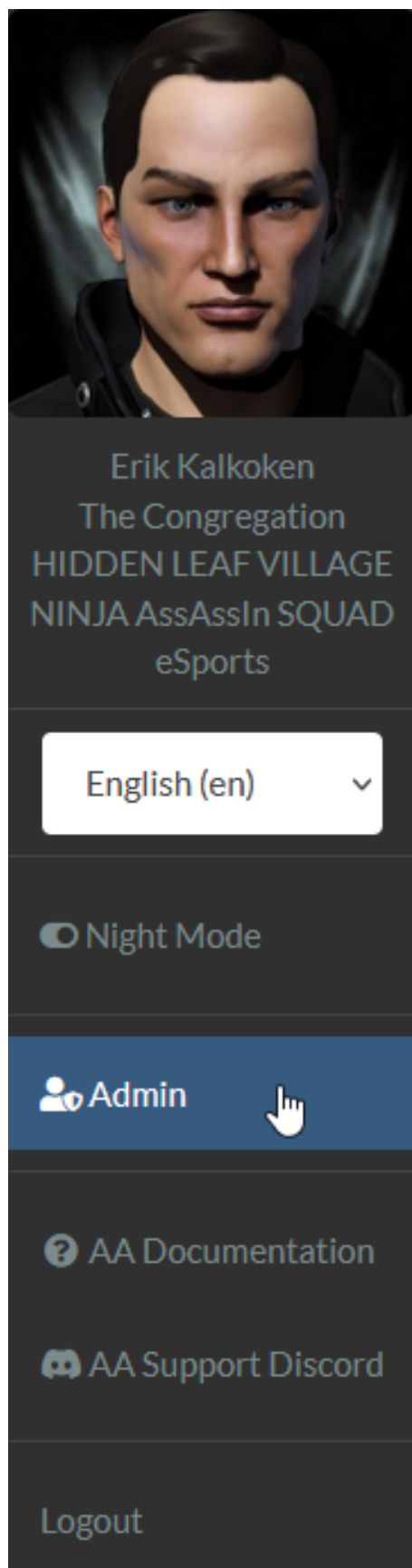
The notifications app can be configured through settings.

- `NOTIFICATIONS_REFRESH_TIME`: The unread count in the top menu is automatically refreshed to keep the user informed about new notifications. This setting allows to set the time between each refresh in seconds. You can also set it to 0 to turn off automatic refreshing. Default: 30
- `NOTIFICATIONS_MAX_PER_USER`: Maximum number of notifications that are stored per user. Older notifications are replaced by newer ones. Default: 50

2.2.6 Admin Site

The admin site allows administrators to configure, manage and trouble shoot Alliance Auth and all its applications and services. E.g. you can create new groups and assign groups to users.

You can open the admin site by clicking on “Admin” in the drop down menu for a user that has access.



Setup for small to medium size installations

For small to medium size alliances it is often sufficient to have no more than two superuser admins (admins that also are superusers). Having two admins usually makes sense, so you can have one primary and one backup.

Warning: Superusers have read & write access to everything on your AA installation. Superusers also automatically have all permissions and therefore access to all features of your apps. Therefore we recommend to be very careful to whom you give superuser privileges.

Setup for large installations

For large alliances and coalitions you may want to have a couple of administrators to be able to distribute and handle the work load. However, having a larger number of superusers may be a security concern.

As an alternative to superusers admins you can define staff admins. Staff admins can perform most of the daily admin work, but are not superusers and therefore can be restricted in what they can access.

To create a staff admin you need to do two things:

1. Enable the `is_staff` property for a user
2. Give the user permissions for admin tasks

Note: Note that staff admins have the following limitations:

- Can not promote users to staff
- Can not promote users to superuser
- Can not add/remove permissions for users, groups and states

These limitations exist to prevent staff admins to promote themselves to quasi superusers. Only superusers can perform these actions.

Staff property

Access to the admin site is restricted. Users needs to have the `is_staff` property to be able to open the site at all. The superuser that is created during the installation process will automatically have access to the admin site.

Hint: Without any permissions a “staff user” can open the admin site, but can neither view nor edit anything except for viewing the list of permissions.

Permissions for common admin tasks

Here is a list of permissions a staff admin would need to perform some common admin tasks:

Edit users

- auth | user | Can view user
- auth | user | Can change user
- authentication | user | Can view user
- authentication | user | Can change user
- authentication | user profile | Can change profile

Delete users

- auth | user | Can view user
- auth | user | Can delete user
- authentication | user | Can delete user
- authentication | user profile | Can delete user profile

Add & edit states

- authentication | state | Can add state
- authentication | state | Can change state
- authentication | state | Can view state

Delete states

- authentication | state | Can delete state
- authentication | state | Can view state

Add & edit groups

- auth | group | Can add group
- auth | group | Can change group
- auth | group | Can view group
- authentication | group | Can add group
- authentication | group | Can change group
- authentication | group | Can view group

Delete groups

- `auth | group | Can delete group`
- `authentication | group | Can delete group`

Permissions for other apps

The permissions a staff admin needs to perform tasks for other applications depends on how the applications are configured. the default is to have four permissions (change, delete, edit view) for each model of the applications. The view permission is usually required to see the model list on the admin site and the other three permissions are required to perform the respective action to an object of that model. However, app developer can chose to define permissions differently.

2.3 Services

Alliance Auth supports managing access to many 3rd party services and apps. This section describes which services are supported and how to install and configure them. Please note that any service need to be installed and configured before it can be used.

2.3.1 Supported Services

Discord

Overview

Discord is a web-based instant messaging client with voice. Kind of like TeamSpeak meets Slack meets Skype. It also has a standalone app for phones and desktop.

Discord is very popular amongst ad-hoc small groups and larger organizations seeking a modern technology. Alternative voice communications should be investigated for larger than small-medium groups for more advanced features.

Setup

Prepare Your Settings File

Make the following changes in your auth project's settings file (`local.py`):

- Add `'allianceauth.services.modules.discord'`, to `INSTALLED_APPS`
- Append the following to the bottom of the settings file:

```
# Discord Configuration
DISCORD_GUILD_ID = ''
DISCORD_CALLBACK_URL = ''
DISCORD_APP_ID = ''
DISCORD_APP_SECRET = ''
DISCORD_BOT_TOKEN = ''
DISCORD_SYNC_NAMES = False
```

(continues on next page)

(continued from previous page)

```
CELERYBEAT_SCHEDULE['discord.update_all_usernames'] = {
    'task': 'discord.update_all_usernames',
    'schedule': crontab(minute=0, hour='*/12'),
}
```

Note: You will have to add most the values for these settings, e.g. your Discord server ID (aka guild ID), later in the setup process.

Creating a Server

Navigate to the [Discord site](#) and register an account, or log in if you have one already.

On the left side of the screen you'll see a circle with a plus sign. This is the button to create a new server. Go ahead and do that, naming it something obvious.

Now retrieve the server ID [following this procedure](#).

Update your auth project's settings file, inputting the server ID as `DISCORD_GUILD_ID`

Note: If you already have a Discord server skip the creation step, but be sure to retrieve the server ID

Registering an Application

Navigate to the [Discord Developers site](#). Press the plus sign to create a new application.

Give it a name and description relating to your auth site. Add a redirect to `https://example.com/discord/callback/`, substituting your domain. Press Create Application.

Update your auth project's settings file, inputting this redirect address as `DISCORD_CALLBACK_URL`

On the application summary page, press Create a Bot User.

Update your auth project's settings file with these pieces of information from the summary page:

- From the App Details panel, `DISCORD_APP_ID` is the Client/Application ID
- From the App Details panel, `DISCORD_APP_SECRET` is the Secret
- From the App Bot Users panel, `DISCORD_BOT_TOKEN` is the Token

Preparing Auth

Before continuing it is essential to run migrations and restart Gunicorn and Celery.

Adding a Bot to the Server

Once created, navigate to the services page of your Alliance Auth install as the superuser account. At the top there is a big green button labelled Link Discord Server. Click it, then from the drop down select the server you created, and then Authorize.

This adds a new user to your Discord server with a BOT tag, and a new role with the same name as your Discord application. Don't touch either of these. If for some reason the bot loses permissions or is removed from the server, click this button again.

To manage roles, this bot role must be at the top of the hierarchy. Edit your Discord server, roles, and click and drag the role with the same name as your application to the top of the list. This role must stay at the top of the list for the bot to work. Finally, the owner of the bot account must enable 2 Factor Authentication (this is required from Discord for kicking and modifying member roles). If you are unsure what 2FA is or how to set it up, refer to [this support page](#). It is also recommended to force 2FA on your server (this forces any admins or moderators to have 2fa enabled to perform similar functions on discord).

Note that the bot will never appear online as it does not participate in chat channels.

Linking Accounts

Instead of the usual account creation procedure, for Discord to work we need to link accounts to Alliance Auth. When attempting to enable the Discord service, users are redirected to the official Discord site to authenticate. They will need to create an account if they don't have one prior to continuing. Upon authorization, users are redirected back to Alliance Auth with an OAuth code which is used to join the Discord server.

Syncing Nicknames

If you want users to have their Discord nickname changed to their in-game character name, set `DISCORD_SYNC_NAMES` to `True`.

Managing Roles

Once users link their accounts you'll notice Roles get populated on Discord. These are the equivalent to groups on every other service. The default permissions should be enough for members to use text and audio communications. Add more permissions to the roles as desired through the server management window.

By default Alliance Auth is taking over full control of role assignments on Discord. This means that users on Discord can in general only have roles that correlate to groups on Auth. However, there are two exceptions to this rule.

Internal Discord roles

First, users will keep their so called "Discord managed roles". Those are internal roles created by Discord e.g. for Nitro.

Excluding roles from being managed by Auth

Second, it is possible to exclude Discord roles from being managed by Auth at all. This can be useful if you have other bots on your Discord server that are using their own roles and which would otherwise conflict with Auth. This would also allow you to manage a role manually on Discord if you so chose.

To exclude roles from being managed by Auth you only have to add them to the list of reserved group names in Group Management.

Note: Role names on Discord are case sensitive, while reserved group names on Auth are not. Therefore reserved group names will cover all roles regardless of their case. For example if you have reserved the group name “alpha”, then the Discord roles “alpha” and “Alpha” will both be persisted.

See also:

For more information see [Reserved group names](#).

Tasks

The Discord service contains a number of tasks that can be run to manually perform updates to all users.

You can run any of these tasks from the command line. Please make sure that you are in your venv and then you can run this command from the same folder that your manage.py is located:

```
celery -A myauth call discord.update_all_groups
```

Name	Description
<i>update_all_groups</i>	Updates groups of all users
<i>update_all_nicknames</i>	Update nicknames of all users (also needs setting)
<i>update_all_usernames</i>	Update locally stored Discord usernames of all users
<i>update_all</i>	Update groups, nicknames, usernames of all users

Note: Depending on how many users you have, running these tasks can take considerable time to finish. You can calculate roughly 1 sec per user for all tasks, except update_all, which needs roughly 3 secs per user.

Settings

You can configure your Discord services with the following settings:

Name	Description	De- fault
<i>DISCORD_APP_ID</i>	Oauth client ID for the Discord Auth app	''
<i>DISCORD_APP_SECRET</i>	Oauth client secret for the Discord Auth app	''
<i>DISCORD_BOT_TOKEN</i>	Generated bot token for the Discord Auth app	''
<i>DISCORD_CALLBACK_URL</i>	Oauth callback URL	''
<i>DISCORD_GUILD_ID</i>	Discord ID of your Discord server	''
<i>DISCORD_GUILD_NAME_CACHE_MAX_AGE</i>	How long the Discord server name is cached locally in seconds	86400
<i>DISCORD_ROLES_CACHE_MAX_AGE</i>	How long roles retrieved from the Discord server are cached locally in seconds	3600
<i>DISCORD_SYNC_NAMES</i>	When set to True the nicknames of Discord users will be set to the user's main character name	False
<i>DISCORD_TASKS_RETRY_PAUSE</i>	Pause in seconds until next retry for tasks after an error occurred	60
<i>DISCORD_TASKS_MAX_RETRIES</i>	max retries of tasks after an error occurred	3

Permissions

To use this service, users will require some of the following.

Permission	Admin Site	Auth Site
discord.access_discord	None	Can Access the Discord Service

Troubleshooting

“Unknown Error” on Discord site when activating service

This indicates your callback URL doesn't match. Ensure the `DISCORD_CALLBACK_URL` setting exactly matches the URL entered on the Discord developers site. This includes `http(s)`, trailing slash, etc.

“Add/Remove” Errors in Discord Service

If you are receiving errors in your Notifications after verifying that your settings are all correct try the following:

- Ensure that the bot's role in Discord is at the top of the roles list. Each time you add it to your server you will need to do this again.
- Make sure that the bot is not trying to modify the Owner of the discord, as it will fail. A holding discord account added with invite link will mitigate this.
- Make sure that the bot role on discord has all needed permissions, Admin etc., remembering that these will need to be set every time you add the bot to the Discord server.

Discourse

Prepare Your Settings

In your auth project's settings file, do the following:

- Add `'allianceauth.services.modules.discourse'`, to your `INSTALLED_APPS` list
- Append the following to your `local.py` settings file:

```
# Discourse Configuration
DISCOURSE_URL = ''
DISCOURSE_API_USERNAME = ''
DISCOURSE_API_KEY = ''
DISCOURSE_SSO_SECRET = ''
```

Install Docker

```
wget -qO- https://get.docker.io/ | sh
```

Install Discourse

Download Discourse

```
mkdir /var/discourse
git clone https://github.com/discourse/discourse_docker.git /var/discourse
```

Configure

```
cd /var/discourse
cp samples/standalone.yml containers/app.yml
nano containers/app.yml
```

Change the following:

- `DISCOURSE_DEVELOPER_EMAILS` should be a list of admin account email addresses separated by commas.
- `DISCOURSE_HOSTNAME` should be `discourse.example.com` or something similar.
- Everything with SMTP depends on your mail settings. [There are plenty of free email services online recommended by Discourse](#) if you haven't set one up for auth already.

To install behind Apache/Nginx, look for this section:

```
...
## which TCP/IP ports should this container expose?
expose:
  - "80:80"    # fwd host port 80   to container port 80 (http)
...
```

Change it to this:

```
...  
## which TCP/IP ports should this container expose?  
expose:  
  - "7890:80"    # fwd host port 7890    to container port 80 (http)  
...
```

Or any other port will do, if taken. Remember this number.

Build and launch

```
nano /etc/default/docker
```

Uncomment this line:

```
DOCKER_OPTS="--dns 8.8.8.8 --dns 8.8.4.4"
```

Restart Docker:

```
service docker restart
```

Now build:

```
./launcher bootstrap app  
./launcher start app
```

Web Server Configuration

You will need to configure your web server to proxy requests to Discourse.

A minimal Apache config might look like:

```
<VirtualHost *:80>  
  ServerName discourse.example.com  
  ProxyPass / http://0.0.0.0:7890/  
  ProxyPassReverse / http://0.0.0.0:7890/  
</VirtualHost>
```

A minimal Nginx config might look like:

```
server {  
  listen 80;  
  server_name discourse.example.com;  
  location / {  
    include proxy_params;  
    proxy_pass http://127.0.0.1:7890;  
  }  
}
```

Configure API

Generate admin account

From the `/var/discourse` directory,

```
./launcher enter app
rake admin:create
```

Follow prompts, being sure to answer `y` when asked to allow admin privileges.

Create API key

Navigate to `discourse.example.com` and log on. Top right press the 3 lines and select Admin. Go to API tab and press Generate Master API Key.

Add the following values to your auth project's settings file:

- `DISCOURSE_URL`: `https://discourse.example.com` (do not add a trailing slash!)
- `DISCOURSE_API_USERNAME`: the username of the admin account you generated the API key with
- `DISCOURSE_API_KEY`: the key you just generated

Configure SSO

Navigate to `discourse.example.com` and log in. Back to the admin site, scroll down to find SSO settings and set the following:

- `enable_sso`: `True`
- `sso_url`: `http://example.com/discourse/sso`
- `sso_secret`: some secure key

Save, now set `DISCOURSE_SSO_SECRET` in your auth project's settings file to the secure key you just put in Discourse.

Finally run migrations and restart Gunicorn and Celery.

Permissions

To use this service, users will require some of the following.

Permission	Admin Site	Auth Site
<code>discourse.access_discourse</code>	None	Can Access the Discourse Service

Mumble

Mumble is a free voice chat server. While not as flashy as TeamSpeak, it has all the functionality and is easier to customize. And is better. I may be slightly biased.

Note: Note that this guide assumes that you have installed Auth with the official [Alliance Auth](#) guide under `/home/allianceserver` and that it is called `myauth`. Accordingly it assumes that you have a service user called `allianceserver` that is used to run all Auth services under supervisor.

Note: Same as the official installation guide this guide is assuming you are performing all steps as `root` user.

Warning: This guide is currently for Ubuntu only.

Installations

Installing Mumble Server

The mumble server package can be retrieved from a repository, which we need to add:

```
apt-add-repository ppa:mumble/release
```

```
apt-get update
```

Now three packages need to be installed:

```
apt-get install python-software-properties mumble-server libqt5sql5-mysql
```

Installing Mumble Authenticator

Next, we need to download the latest authenticator release from the [authenticator repository](#).

```
git clone https://gitlab.com/allianceauth/mumble-authenticator /home/allianceserver/  
↪mumble-authenticator
```

We will now install the authenticator into your Auth virtual environment. Please make sure to activate it first:

```
source /home/allianceserver/venv/auth/bin/activate
```

Install the python dependencies for the mumble authenticator. Note that this process can take a couple minutes to complete.

```
pip install -r requirements.txt
```


Configuring Mumble Server

The mumble server needs it's own database. Open an SQL shell with `mysql -u root -p` and execute the SQL commands to create it:

```
CREATE DATABASE alliance_mumble CHARACTER SET utf8mb4;
```

```
GRANT ALL PRIVILEGES ON alliance_mumble . * TO 'allianceserver'@'localhost';
```

Mumble ships with a configuration file that needs customization. By default it's located at `/etc/mumble-server.ini`. Open it with your favorite text editor:

```
nano /etc/mumble-server.ini
```

We need to enable the ICE authenticator. Edit the following:

- `icesecretwrite=MY_CLEVER_PASSWORD`, obviously choosing a secure password
- ensure the line containing `Ice="tcp -h 127.0.0.1 -p 6502"` is uncommented

We also want to enable Mumble to use the previously created MySQL / MariaDB database, edit the following:

- uncomment the database line, and change it to `database=alliance_mumble`
- `dbDriver=QMYSQL`
- `dbUsername=allianceserver` or whatever you called the Alliance Auth MySQL user
- `dbPassword=` that user's password
- `dbPort=3306`
- `dbPrefix=murmur_`

To name your root channel, uncomment and set `registerName=` to whatever cool name you want

Save and close the file.

To get Mumble superuser account credentials, run the following:

```
dpkg-reconfigure mumble-server
```

Set the password to something you'll remember and write it down. This is your superuser password and later needed to manage ACLs.

Now restart the server to see the changes reflected.

```
service mumble-server restart
```

That's it! Your server is ready to be connected to at `example.com:64738`

Configuring Mumble Authenticator

The ICE authenticator lives in the mumble-authenticator repository, cd to the directory where you cloned it.

Make a copy of the default config:

```
cp authenticator.ini.example authenticator.ini
```

Edit `authenticator.ini` and change these values:

- [database]
 - user = your allianceserver MySQL user
 - password = your allianceserver MySQL user's password
- [ice]
 - secret = the icewritesecret password set earlier

Test your configuration by starting it:

```
python /home/allianceserver/mumble-authenticator/authenticator.py
```

And finally ensure the allianceserver user has read/write permissions to the mumble authenticator files before proceeding:

```
chown -R allianceserver:allianceserver /home/allianceserver/mumble-authenticator
```

The authenticator needs to be running 24/7 to validate users on Mumble. This can be achieved by adding a section to your auth project's supervisor config file like the following example:

```
[program:authenticator]
command=/home/allianceserver/venv/auth/bin/python authenticator.py
directory=/home/allianceserver/mumble-authenticator
user=allianceserver
stdout_logfile=/home/allianceserver/myauth/log/authenticator.log
stderr_logfile=/home/allianceserver/myauth/log/authenticator.log
autostart=true
autorestart=true
startsecs=10
priority=996
```

In addition we'd recommend to add the authenticator to Auth's restart group in your supervisor conf. For that you need to add it to the group line as shown in the following example:

```
[group:myauth]
programs=beat,worker,gunicorn,authenticator
priority=999
```

To enable the changes in your supervisor configuration you need to restart the supervisor process itself. And before we do that we are shutting down the current Auth supervisors gracefully:

```
supervisor stop myauth:
systemctl restart supervisor
```

Configuring Auth

In your auth project's settings file (`myauth/settings/local.py`), do the following:

- Add `'allianceauth.services.modules.mumble'`, to your `INSTALLED_APPS` list
- set `MUMBLE_URL` to the public address of your mumble server. Do not include any leading `http://` or `mumble://`.

Example config:

```
# Installed apps
INSTALLED_APPS += [
    # ...
    'allianceauth.services.modules.mumble'
    # ...
]

# Mumble Configuration
MUMBLE_URL = "mumble.example.com"
```

Finally, run migrations and restart your supervisor to complete the setup:

```
python /home/allianceserver/myauth/manage.py migrate
```

```
supervisorctl restart myauth:
```

Permissions

To use this service, users will require some of the following.

Permission	Admin Site	Auth Site
<code>mumble.access_mumble</code>	None	Can Access the Mumble Service

ACL configuration

On a freshly installed mumble server only your superuser has the right to configure ACLs and create channels. The credentials for logging in with your superuser are:

- user: `SuperUser`
- password: *what you defined when configuring your mumble server*

Optimizing a Mumble Server

The needs and available resources will vary between Alliance Auth installations. Consider yours when applying these settings.

Bandwidth

<https://wiki.mumble.info/wiki/Murmur.ini#bandwidth> This is likely the most important setting for scaling a Mumble install, The default maximum Bandwidth is 72000bps Per User. Reducing this value will cause your clients to automatically scale back their bandwidth transmitted, while causing a reduction in voice quality. A value thats still high may cause robotic voices or users with bad connections to drop due entirely due to network load.

Please tune this value to your individual needs, the below scale may provide a rough starting point. 72000 - Superior voice quality - Less than 50 users. 54000 - No noticeable reduction in quality - 50+ Users or many channels with active audio. 36000 - Mild reduction in quality - 100+ Users 30000 - Noticeable reduction in quality but not function - 250+ Users

Forcing Opus

<https://wiki.mumble.info/wiki/Murmur.ini#opusthreshold> A Mumble server by default, will fall back to the older CELT codec as soon as a single user connects with an old client. This will significantly reduce your audio quality and likely place higher load on your server. We *highly* reccommend setting this to Zero, to force OPUS to be used at all times. Be aware any users with Mumble clients prior to 1.2.4 (From 2013...) Will not hear any audio.

```
opusthreshold=0
```

AutoBan and Rate Limiting

https://wiki.mumble.info/wiki/Murmur.ini#autobanAttempts.2C_autobanTimeframe_and_autobanTime The AutoBan feature has some sensible settings by default, You may wish to tune these if your users keep locking themselves out by opening two clients by mistake, or if you are receiving unwanted attention

https://wiki.mumble.info/wiki/Murmur.ini#messagelimit_and_messageburst This too, is set to a sensible configuration by default. Take note on upgrading older installs, as this may actually be set too restrictively and will rate-limit your admins accidentally, take note of the configuration in <https://github.com/mumble-voip/mumble/blob/master/scripts/murmur.ini#L156>

“Suggest” Options

There is no way to force your users to update their clients or use Push to Talk, but these options will throw an error into their Mumble Client.

<https://wiki.mumble.info/wiki/Murmur.ini#Miscellany>

We suggest using Mumble 1.3.0+ for your server and Clients, you can tune this to the latest Patch version. `suggestVersion=1.3.0`

If Push to Talk is to your tastes, configure the suggestion as follows `suggestPushToTalk=true`

General notes

Setting a server password

With the default configuration your mumble server is public. Meaning that everyone who has the address can at least connect to it and might also be able to join all channels that don't have any permissions set (Depending on your ACL configured for the root channel). If you want only registered member being able to join your mumble, you have to set a server password. To do so open your mumble server configuration which is by default located at `/etc/mumble-server.ini`.

```
nano /etc/mumble-server.ini
```

Now search for `serverpassword=` and set your password here. If there is no such line, simply add it.

```
serverpassword=YourSuperSecretServerPassword
```

Save the file and restart your mumble server afterwards.

```
service mumble-server restart
```

From now on, only registered member can join your mumble server. Now if you still want to allow guests to join you have 2 options.

- Allow the “Guest” state to activate the Mumble service in your Auth instance
- Use [Mumble temporary links](#)

Enabling Avatars in Overlay (V1.0.0+)

Ensure you have an up to date Mumble-Authenticator, this feature was added in V1.0.0

Edit `authenticator.ini` and change (or add for older installs) This code block.

```
;If enabled, textures are automatically set as player's EvE avatar for use on overlay.
avatar_enable = True
;Get EvE avatar images from this location. {charid} will be filled in.
ccp_avatar_url = https://images.evetech.net/characters/{charid}/portrait?size=32
```

Openfire

Openfire is a Jabber (XMPP) server.

Prepare Your Settings

- Add `'allianceauth.services.modules.openfire'`, to your `INSTALLED_APPS` list
- Append the following to your auth project's settings file:

```
# Jabber Configuration
JABBER_URL = ""
JABBER_PORT = 5223
JABBER_SERVER = ""
OPENFIRE_ADDRESS = ""
```

(continues on next page)

(continued from previous page)

```
OPENFIRE_SECRET_KEY = ""
BROADCAST_USER = ""
BROADCAST_USER_PASSWORD = ""
BROADCAST_SERVICE_NAME = "broadcast"
```

Dependencies

Openfire require a Java 8 runtime environment.

Ubuntu:

```
apt-get install openjdk-8-jdk
```

CentOS:

```
yum -y install java-1.8.0-openjdk java-1.8.0-openjdk-devel
```

Setup

Download Installer

Openfire is not available through repositories so we need to get a package from the developer.

On your PC, navigate to the [Ignite Realtime downloads section](#), and under Openfire select Linux, click on the Ubuntu: Debian package (second from bottom of list, ends with .deb) or CentOS: RPM Package (no JRE bundled, as we have installed it on the host)

Retrieve the file location by copying the URL from the “click here” link, depending on your browser you may have a Copy Link or similar option in your right click menu.

In the console, ensure you’re in your user’s home directory: `cd ~`

Now download the package. Replace the link below with the link you got earlier.

```
wget https://www.igniterealtime.org/downloadServlet?filename=openfire/
openfire_4.2.3_all.deb
```

Now install from the package. Replace the filename with your filename (the last part of the download URL is the file name)

Ubuntu:

```
dpkg -i openfire_4.2.3_all.deb
```

CentOS:

```
yum install -y openfire-4.2.3-1.noarch.rpm
```

Create Database

Performance is best when working from a SQL database. If you installed MySQL or MariaDB alongside your auth project, go ahead and create a database for Openfire:

```
mysql -u root -p
create database alliance_jabber;
grant all privileges on alliance_jabber . * to 'allianceserver'@'localhost';
exit;
```

Web Configuration

The remainder of the setup occurs through Openfire's web interface. Navigate to `http://example.com:9090`, or if you're behind CloudFlare, go straight to your server's IP:9090.

Select your language. I sure hope it's English if you're reading this guide.

Under Server Settings, set the Domain to `example.com` replacing it with your actual domain. Don't touch the rest.

Under Database Settings, select `Standard Database Connection`

On the next page, select `MySQL` from the dropdown list and change the following:

- `[server]` is replaced by `127.0.0.1`
- `[database]` is replaced by the name of the database to be used by Openfire
- enter the login details for your auth project's database user

If Openfire returns with a failed to connect error, re-check these settings. Note the lack of square brackets.

Under Profile Settings, leave `Default` selected.

Create an administrator account. The actual name is irrelevant, just don't lose this login information.

Finally, log in to the console with your admin account.

Edit your auth project's settings file and enter the values you just set:

- `JABBER_URL` is the public address of your jabber server
- `JABBER_PORT` is the port for clients to connect to (usually 5223)
- `JABBER_SERVER` is the name of the jabber server. If you didn't alter it during install it'll usually be your domain (eg `example.com`)
- `OPENFIRE_ADDRESS` is the web address of Openfire's web interface. Use `http://` with port 9090 or `https://` with port 9091 if you configure SSL in Openfire

REST API Setup

Navigate to the `plugins` tab, and then `Available Plugins` on the left navigation bar. You'll need to fetch the list of available plugins by clicking the link.

Once loaded, press the green plus on the right for `REST API`.

Navigate the `Server` tab, `Server Settings` subtab. At the bottom of the left navigation bar select `REST API`.

Select `Enabled`, and `Secret Key Auth`. Update your auth project's settings with this secret key as `OPENFIRE_SECRET_KEY`.

Broadcast Plugin Setup

Navigate to the `Users/Groups` tab and select `Create New User` from the left navigation bar.

Pick a username (e.g. `broadcast`) and password for your ping user. Enter these in your auth project's settings file as `BROADCAST_USER` and `BROADCAST_USER_PASSWORD`. Note that `BROADCAST_USER` needs to be in the format `user@example.com` matching your jabber server name. Press `Create User` to save this user.

Broadcasting requires a plugin. Navigate to the `plugins` tab, press the green plus for the `Broadcast` plugin.

Navigate to the `Server` tab, `Server Manager` subtab, and select `System Properties`. Enter the following:

- Name: `plugin.broadcast.disableGroupPermissions`
 - Value: `True`
 - Do not encrypt this property value
- Name: `plugin.broadcast.allowedUsers`
 - Value: `broadcast@example.com`, replacing the domain name with yours
 - Do not encrypt this property value

If you have troubles getting broadcasts to work, you can try setting the optional (you will need to add it) `BROADCAST_IGNORE_INVALID_CERT` setting to `True`. This will allow invalid certificates to be used when connecting to the Openfire server to send a broadcast.

Preparing Auth

Once all settings are entered, run migrations and restart Gunicorn and Celery.

Group Chat

Channels are available which function like a chat room. Access can be controlled either by password or ACL (not unlike mumble).

Navigate to the `Group Chat` tab and select `Create New Room` from the left navigation bar.

- Room ID is a short, easy-to-type version of the room's name users will connect to
- Room Name is the full name for the room
- Description is short text describing the room's purpose
- Set a password if you want password authentication
- Every other setting is optional. Save changes.

Now select your new room. On the left navigation bar, select `Permissions`.

ACL is achieved by assigning groups to each of the three tiers: `Owners`, `Admins` and `Members`. `Outcast` is the blacklist. You'll usually only be assigning groups to the `Member` category.

Permissions

To use this service, users will require some of the following.

Permission	Admin Site	Auth Site
openfire.access_openfire	None	Can Access the Openfire Service

phpBB3

Overview

phpBB is a free PHP-based forum.

Dependencies

phpBB3 requires PHP installed in your web server. Apache has `mod_php`, NGINX requires `php-fpm`. See [the official guide](#) for PHP package requirements.

Prepare Your Settings

In your auth project's settings file, do the following:

- Add `'allianceauth.services.modules.phpbb3'`, to your `INSTALLED_APPS` list
- Append the following to the bottom of the settings file:

```
# PHPBB3 Configuration
PHPBB3_URL = ''
DATABASES['phpbb3'] = {
    'ENGINE': 'django.db.backends.mysql',
    'NAME': 'alliance_forum',
    'USER': 'allianceserver-phpbb3',
    'PASSWORD': 'password',
    'HOST': '127.0.0.1',
    'PORT': '3306',
}
```

Setup

Prepare the Database

Create a database to install phpBB3 in.

```
mysql -u root -p
create database alliance_forum;
grant all privileges on alliance_forum . * to 'allianceserver'@'localhost';
exit;
```

Edit your auth project's settings file and fill out the `DATABASES['phpbb3']` part.

Download phpBB3

phpBB3 is available as a zip from their website. Navigate to the website's [downloads section](#) using your PC browser and copy the URL for the latest version zip.

In the console, navigate to your user's home directory: `cd ~`

Now download using `wget`, replacing the URL with the URL for the package you just retrieved

```
wget https://www.phpbb.com/files/release/phpBB-3.2.2.zip
```

This needs to be unpackaged. Unzip it, replacing the file name with that of the file you just downloaded

```
unzip phpBB-3.2.2.zip
```

Now we need to move this to our web directory. Usually `/var/www/forums`.

```
mv phpBB3 /var/www/forums
```

The web server needs read/write permission to this folder

Apache: `chown -R www-data:www-data /var/www/forums` Nginx: `chown -R nginx:nginx /var/www/forums`

Tip: Nginx: Some distributions use the `www-data:www-data` user:group instead of `nginx:nginx`. If you run into problems with permissions try it instead.

Configuring Web Server

You will need to configure you web server to serve PHPBB3 before proceeding with installation.

A minimal Apache config file might look like:

```
<VirtualHost *:80>
    ServerName forums.example.com
    DocumentRoot /var/www/forums
    <Directory /var/www/forums>
        Require all granted
        DirectoryIndex index.php
    </Directory>
</VirtualHost>
```

A minimal Nginx config file might look like:

```
server {
    listen 80;
    server_name forums.example.com;
    root /var/www/forums;
    index index.php;
    access_log /var/logs/forums.access.log;

    location ~ /(config\.php|common\.php|cache|files|images/avatars/
↪upload|includes|store) {
        deny all;
        return 403;
    }
}
```

(continues on next page)

(continued from previous page)

```
}

location ~* \.(gif|jpe?g|png|css)$ {
    expires 30d;
}

location ~ \.php$ {
    try_files $uri =404;
    fastcgi_pass unix:/tmp/php.socket;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    include fastcgi_params;
}
}
```

Enter your forum's web address as the PHPBB3_URL setting in your auth project's settings file.

Web Install

Navigate to your forums web address where you will be presented with an installer.

Click on the Install tab.

All the requirements should be met. Press Start Install.

Under Database Settings, set the following:

- Database Type is MySQL
- Database Server Hostname is 127.0.0.1
- Database Server Port is left blank
- Database Name is alliance_forum
- Database Username is your auth MySQL user, usually allianceserver
- Database Password is this user's password

If you use a table prefix other than the standard phpbb_ you need to add an additional setting to your auth project's settings file, PHPBB3_TABLE_PREFIX = ' ', and enter the prefix.

You should see Successful Connection and proceed.

Enter administrator credentials on the next page.

Everything from here should be intuitive.

phpBB will then write its own config file.

Open the Forums

Before users can see the forums, we need to remove the install directory

```
rm -rf /var/www/forums/install
```

Enabling Avatars

AllianceAuth sets user avatars to their character portrait when the account is created or password reset. We need to allow external URLs for avatars for them to behave properly. Navigate to the admin control panel for phpbb3, and under the General tab, along the left navigation bar beneath Board Configuration, select Avatar Settings. Set Enable Remote Avatars to Yes and then Submit.

You can allow members to overwrite the portrait with a custom image if desired. Navigate to Users and Groups, Group Permissions, select the appropriate group (usually Member if you want everyone to have this ability), expand Advanced Permissions, under the Profile tab, set Can Change Avatars to Yes, and press Apply Permissions.

Setting the default theme

Users generated via Alliance Auth do not have a default theme set. You will need to set this on the phpbb_users table in SQL

```
mysql -u root -p
use alliance_forum;
alter table phpbb_users change user_style user_style int not null default 1
```

If you would like to use a theme that is NOT prosilver or theme “1”. You will need to deactivate prosilver, this will then fall over to the set forum wide default.

Prepare Auth

Once settings have been configured, run migrations and restart Gunicorn and Celery.

Permissions

To use this service, users will require some of the following.

Permission	Admin Site	Auth Site
phpbb3.access_phpbb3	None	Can Access the PHPBB3 Service

SMF

Overview

SMF is a free PHP-based forum.

Dependencies

SMF requires PHP installed in your web server. Apache has `mod_php`, NGINX requires `php-fpm`. More details can be found in the [SMF requirements page](#).

Prepare Your Settings

In your auth project's settings file, do the following:

- Add `'allianceauth.services.modules.smf'`, to your `INSTALLED_APPS` list
- Append the following to the bottom of the settings file:

```
# SMF Configuration
SMF_URL = ''
DATABASES['smf'] = {
    'ENGINE': 'django.db.backends.mysql',
    'NAME': 'alliance_smf',
    'USER': 'allianceserver-smf',
    'PASSWORD': 'password',
    'HOST': '127.0.0.1',
    'PORT': '3306',
}
```

Setup

Download SMF

Using your browser, you can download the latest version of SMF to your desktop computer. All SMF downloads can be found at SMF Downloads. The latest recommended version will always be available at <http://www.simplerachines.org/download/index.php/latest/install/>. Retrieve the file location from the hyperlinked box icon for the zip full install, depending on your browser you may have a Copy Link or similar option in your right click menu.

Download using `wget`, replacing the URL with the URL for the package you just retrieved

```
wget https://download.simplerachines.org/index.php?thanks;filename=smf_2-0-15_install.
↪zip
```

This needs to be unpackaged. Unzip it, replacing the file name with that of the file you just downloaded

```
unzip smf_2-0-15_install.zip
```

Now we need to move this to our web directory. Usually `/var/www/forums`.

```
mv smf /var/www/forums
```

The web server needs read/write permission to this folder

Apache: `chown -R www-data:www-data /var/www/forums` Nginx: `chown -R nginx:nginx /var/www/forums`

Tip: Nginx: Some distributions use the `www-data:www-data` user:group instead of `nginx:nginx`. If you run into problems with permissions try it instead.

Database Preparation

SMF needs a database. Create one:

```
mysql -u root -p
```

```
create database alliance_smf;
grant all privileges on alliance_smf . * to 'allianceserver'@'localhost';
exit;
```

Enter the database information into the `DATABASES ['smf']` section of your auth project's settings file.

Web Server Configuration

Your web server needs to be configured to serve SMF.

A minimal Apache config might look like:

```
<VirtualHost *:80>
    ServerName forums.example.com
    DocumentRoot /var/www/forums
    <Directory "/var/www/forums">
        DirectoryIndex index.php
    </Directory>
</VirtualHost>
```

A minimal Nginx config might look like:

```
server {
    listen 80;
    server_name forums.example.com;
    root /var/www/forums;
    index index.php;
    access_log /var/logs/forums.access.log;

    location ~ /\.php$ {
        try_files $uri =404;
        fastcgi_pass unix:/tmp/php.socket;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

Enter the web address to your forums into the `SMF_URL` setting in your auth project's settings file.

Web Install

Navigate to your forums address where you will be presented with an installer.

Click on the `Install` tab.

All the requirements should be met. Press `Start Install`.

Under Database Settings, set the following:

- Database Type is `MySQL`
- Database Server Hostname is `127.0.0.1`
- Database Server Port is left blank
- Database Name is `alliance_smf`
- Database Username is your auth MySQL user, usually `allianceserver`
- Database Password is this user's password

If you use a table prefix other than the standard `smf_` you need to add an additional setting to your auth project's settings file, `SMF_TABLE_PREFIX = ''`, and enter the prefix.

Follow the directions in the installer.

Preparing Auth

Once settings are entered, apply migrations and restart Gunicorn and Celery.

Permissions

To use this service, users will require some of the following.

Permission	Admin Site	Auth Site
<code>smf.access_smf</code>	None	Can Access the SMF Service

TeamSpeak 3

Overview

TeamSpeak3 is the most popular VOIP program for gamers.

But have you considered using Mumble? Not only is it free, but it has features and performance far superior to Teamspeak3.

Setup

Sticking with TS3? Alright, I tried.

Prepare Your Settings

In your auth project's settings file, do the following:

- Add `'allianceauth.services.modules.teamspeak3'`, to your `INSTALLED_APPS` list
- Append the following to the bottom of the settings file:

```
# Teamspeak3 Configuration
TEAMSPEAK3_SERVER_IP = '127.0.0.1'
TEAMSPEAK3_SERVER_PORT = 10011
TEAMSPEAK3_SERVERQUERY_USER = 'serveradmin'
TEAMSPEAK3_SERVERQUERY_PASSWORD = ''
TEAMSPEAK3_VIRTUAL_SERVER = 1
TEAMSPEAK3_PUBLIC_URL = ''

CELERYBEAT_SCHEDULE['run_ts3_group_update'] = {
    'task': 'allianceauth.services.modules.teamspeak3.tasks.run_ts3_group_update',
    'schedule': crontab(minute='*/30'),
}
```

Download Installer

To install we need a copy of the server. You can find the latest version from [this dl server](#) (I'd recommend getting the latest stable version – find this version number from the [TeamSpeak site](#)). Be sure to get a link to the Linux version.

Download the server, replacing the link with the link you got earlier.

```
http://dl.4players.de/ts/releases/3.13.2/teamspeak3-server_linux_amd64-3.13.2.tar.bz2
```

Now we need to extract the file.

```
tar -xf teamspeak3-server_linux_amd64-3.1.0.tar.bz2
```

Create User

TeamSpeak needs its own user.

```
adduser --disabled-login teamspeak
```


Install Binary

Now we move the server binary somewhere more accessible and change its ownership to the new user.

```
mv teamspeak3-server_linux_amd64 /usr/local/teamspeak
chown -R teamspeak:teamspeak /usr/local/teamspeak
```

Startup

Now we generate a startup script so TeamSpeak comes up with the server.

```
ln -s /usr/local/teamspeak/ts3server_startscript.sh /etc/init.d/teamspeak
update-rc.d teamspeak defaults
```

Finally we start the server.

```
service teamspeak start
```

Update Settings

The console will spit out a block of text. If it does not appear, it can be found with `service teamspeak status`. **SAVE THIS.**

If you plan on claiming the ServerAdmin token, do so with a different TeamSpeak client profile than the one used for your auth account, or you will lose your admin status.

Edit the settings you added to your auth project's settings file earlier, entering the following:

- `TEAMSPEAK3_SERVERQUERY_USER` is `loginname` from that block of text it just spat out (usually `serveradmin`)
- `TEAMSPEAK3_SERVERQUERY_PASSWORD` is `password` from that block of text it just spat out
- `TEAMSPEAK_VIRTUAL_SERVER` is the virtual server ID of the server to be managed - it will only ever not be 1 if your server is hosted by a professional company
- `TEAMSPEAK3_PUBLIC_URL` is the public address of your TeamSpeak server. Do not include any leading `http://` or `teamspeak://`

Once settings are entered, run migrations and restart Gunicorn and Celery.

Generate User Account

And now we can generate ourselves a user account. Navigate to the services in Alliance Auth for your user account and press the checkmark for TeamSpeak 3.

Click the URL provided to automatically connect to our server. It will prompt you to redeem the `serveradmin` token, enter the `token` from startup.

Groups

Now we need to make groups. AllianceAuth handles groups in teamspeak differently: instead of creating groups it creates an association between groups in TeamSpeak and groups in AllianceAuth. Go ahead and make the groups you want to associate with auth groups, keeping in mind multiple TeamSpeak groups can be associated with a single auth group.

Navigate back to the AllianceAuth admin interface (example.com/admin) and under Teamspeak3, select Auth / TS Groups.

In the top-right corner click, first click on Update TS3 Groups to fetch the newly created server groups from TS3 (this may take a minute to complete). Then click on Add Auth / TS Group to link Auth groups with TS3 server groups.

The dropdown box provides all auth groups. Select one and assign TeamSpeak groups from the panels below. If these panels are empty, wait a minute for the database update to run, or see the [troubleshooting section](#) below.

Troubleshooting

Insufficient client permissions (failed on Invalid permission: 0x26)

Using the advanced permissions editor, ensure the Guest group has the permission Use Privilege Keys to gain permissions (under Virtual Server expand the Administration section)

To enable advanced permissions, on your client go to the Tools menu, Application, and under the Misc section, tick Advanced permission system

TS group models not populating on admin site

The method which populates these runs every 30 minutes. To populate manually you start the process from the admin site or from the Django shell.

Admin Site

Navigate to the AllianceAuth admin interface and under Teamspeak3, select Auth / TS Groups.

Then, in the top-right corner click, click on Update TS3 Groups to start the process of fetching the server groups from TS3 (this may take a minute to complete).

Django Shell

Start a django shell with:

```
python manage.py shell
```

And execute the update as follows:

```
from allianceauth.services.modules.teamspeak3.tasks import Teamspeak3Tasks
Teamspeak3Tasks.run_ts3_group_update()
```

Ensure that command does not return an error.

2564 access to default group is forbidden

This usually occurs because auth is trying to remove a user from the `Guest` group (group ID 8). The guest group is only assigned to a user when they have no other groups, unless you have changed the default teamspeak server config.

Teamspeak servers v3.0.13 and up are especially susceptible to this. Ensure the Channel Admin Group is not set to `Guest (8)`. Check by right clicking on the server name, `Edit virtual server`, and in the middle of the panel select the `Misc` tab.

TypeError: string indices must be integers, not str

This error generally means teamspeak returned an error message that went unhandled. The full traceback is required for proper debugging, which the logs do not record. Please check the superuser notifications for this record and get in touch with a developer.

3331 flood ban

This most commonly happens when your teamspeak server is externally hosted. You need to add the auth server IP to the teamspeak serverquery whitelist. This varies by provider.

If you have SSH access to the server hosting it, you need to locate the teamspeak server folder and add the auth server IP on a new line in `query_ip_allowlist.txt` (named `query_ip_whitelist.txt` on older teamspeak versions).

520 invalid loginname or password

The serverquery account login specified in `local.py` is incorrect. Please verify `TEAMSPEAK3_SERVERQUERY_USER` and `TEAMSPEAK3_SERVERQUERY_PASSWORD`. The [installation section](#) describes where to get them.

2568 insufficient client permissions

This usually occurs if you've created a separate serverquery user to use with auth. It has not been assigned sufficient permissions to complete all the tasks required of it. The full list of required permissions is not known, so assign liberally.

Permissions

To use and configure this service, users will require some of the following.

Permission	Admin Site	Auth Site
<code>teamspeak.access_teamspeak</code>	None	Can Access the TeamSpeak Service
<code>teamspeak.add_auths</code>	Can Add Model	None
<code>teamspeak.change_auths</code>	Can Change Model	None
<code>teamspeak.delete_auths</code>	Can Delete Model	None
<code>teamspeak.view_auths</code>	Can View Model	None

XenForo

Overview

[XenForo](#) is a popular paid forum. This guide will assume that you already have XenForo installed with a valid license (please keep in mind that XenForo is not free nor open-source, therefore you need to purchase a license first). If you come across any problems related with the installation of XenForo please contact their support service.

Prepare Your Settings

In your auth project's settings file, do the following:

- Add `'allianceauth.services.modules.xenforo'`, to your `INSTALLED_APPS` list
- Append the following to your `local.py` settings file:

```
# XenForo Configuration
XENFORO_ENDPOINT = 'example.com/api.php'
XENFORO_DEFAULT_GROUP = 0
XENFORO_APIKEY = 'yourapikey'
```

XenAPI

By default XenForo does not support any kind of API, however there is a third-party package called [XenAPI](#) which provides a simple REST interface by which we can access XenForo's functions in order to create and edit users.

The installation of XenAPI is pretty straight forward. The only thing you need to do is to download the `api.php` from the official repository and upload it in the root folder of your XenForo installation. The final result should look like this: `*forumswebsite.com/*api.php`

Now that XenAPI is installed the only thing left to do is to provide a key.

```
$restAPI = new RestAPI('REPLACE_THIS_WITH_AN_API_KEY');
```

Configuration

The settings you created earlier now need to be filled out.

`XENFORO_ENDPOINT` is the address to the API you added. No leading `http://`, but be sure to include the `/api.php` at the end.

`XENFORO_DEFAULT_GROUP` is the ID of the group in XenForo auth users will be added to. Unfortunately XenAPI **cannot create new groups**, therefore you have to create a group manually and then get its ID.

`XENFORO_API_KEY` is the API key value you set earlier.

Once these are entered, run migrations and restart Gunicorn and Celery.

Permissions

To use this service, users will require some of the following.

Permission	Admin Site	Auth Site
xenforo.access_xenforo	None	Can Access the XenForo Service

2.3.2 Tools

Services Name Formats

This app allows you to customize how usernames for services are created.

Each service's username or nickname, depending on which the service supports, can be customized through the use of the Name Formatter config provided the service supports custom formats. This config can be found in the admin panel under **Services -> Name format config**

Currently the following services support custom name formats:

Service	Used with	Default Formatter
Discord	Nickname	{character_name}
Discourse	Username	{character_name}
IPS4	Username	{character_name}
Mumble	Username	[{corp_ticker}]{character_name}
Openfire	Username	{character_name}
phpBB3	Username	{character_name}
SMF	Username	{character_name}
Teamspeak 3	Nickname	[{corp_ticker}]{character_name}
Xenforo	Username	{character_name}

Note: It's important to note here, before we get into what you can do with a name formatter, that before the generated name is passed off to the service to create an account it will be sanitized to remove characters (the letters and numbers etc.) that the service cannot support. This means that, despite what you configured, the service may display something different. It is up to you to test your formatter and understand how your format may be disrupted by a certain services sanitization function.

Available format data

The following fields are available from a users account and main character:

- username - Alliance Auth username
- character_id
- character_name
- corp_id
- corp_name
- corp_ticker
- alliance_id

- `alliance_name`
- `alliance_ticker`
- `alliance_or_corp_name` (defaults to Corporation name if there is no Alliance)
- `alliance_or_corp_ticker` (defaults to Corporation ticker if there is no Alliance)

Building a formatter string

The name formatter uses the advanced string formatting specified by [PEP-3101](#). Anything supported by this specification is supported in a name formatter.

A more digestible documentation of string formatting in Python is available on the [PyFormat](#) website.

Some examples of strings you could use:

Formatter	Result
<code>{alliance_ticker} - {character_name}</code>	MYALLI - My Character
<code>[{corp_ticker}] {character_name}</code>	[CORP] My Character
<code>{{{corp_name}}}{character_name}</code>	{My Corp}My Character

Important: For most services, name formats only take effect when a user creates an account. This means if you create or update a name formatter it won't retroactively alter the format of users names. There are some exceptions to this where the service updates nicknames on a periodic basis. Check the service's documentation to see which of these apply.

Important: You must only create one formatter per service per state. E.g. don't create two formatters for Mumble for the Member state. In this case one of the formatters will be used and it may not be the formatter you are expecting.

Service Permissions

In the past, access to services was dictated by a list of settings in `settings.py`, granting access to each particular service for Members and/or Blues. This meant that granting access to a service was very broad and rigidly structured around these two states.

Permissions based access

Instead of granting access to services by the previous rigid structure, access to services is now granted by the built in Django permissions system. This means that service access can be more granular, allowing only certain states, certain groups, for instance Corp CEOs, or even individual user access to each enabled service.

Important: If you grant access to an individual user, they will have access to that service regardless of whether or not they are a member.

Each service has an access permission defined, named like `Can access the <service name> service`.

To mimick the old behaviour of enabling services for all members, you would select the `Member` group from the admin panel, add the required service permission to the group and save. Likewise for Blues, select the `Blue` group and add the required permission.

A user can be granted the same permission from multiple sources. e.g. they may have it granted by several groups and directly granted on their account as well. Auth will not remove their account until all instances of the permission for that service have been revoked.

Removing access

Danger: Access removal is processed immediately after removing a permission from a user or group. If you remove access from a large group, such as `Member`, it will immediately remove all users from that service.

When you remove a service permission from a user, a signal is triggered which will activate an immediate permission check. For users this will trigger an access check for all services. For groups, due to the potential extra load, only the services whose permissions have changed will be verified, and only the users in that group.

If a user no longer has permission to access the service when this permissions check is triggered, that service will be immediately disabled for them.

Disabling user accounts

When you unset a user as active in the admin panel, all of that users service accounts will be immediately disabled or removed. This is due to the built in behaviour of the Django permissions system, which will return `False` for all permissions if a users account is disabled, regardless of their actual permissions state.

2.4 Apps

Alliance Auth comes with a set of apps (also called plugin-apps) which provide basic functions useful to many organizations in Eve Online like a fleet schedule and a timerboard. This section describes which apps are available and how to install and use them. Please note that any app need to be installed before it can be used.

2.4.1 Auto Groups

Note: New in 2.0

Auto groups allows you to automatically place users of certain states into Corp or Alliance based groups. These groups are created when the first user is added to them and removed when the configuration is deleted.

Installation

This is an optional app that needs to be installed.

To install this app add `'allianceauth.eveonline.autogroups'`, to your `INSTALLED_APPS` list and run migrations. All other settings are controlled via the admin panel under the `Eve_Autogroups` section.

Configuring a group

When you create an autogroup config you will be given the following options:

Add autogroups config

States:

Member

Blue

Guest

+

Hold down "Control", or "Command" on a Mac, to select more than one.

☐ Corp groups

Setting this to false will delete all the created groups.

Corp group prefix:

Corp name source:

Full name

☐ Alliance groups

Setting this to false will delete all the created groups.

Alliance group prefix:

Alliance name source:

Full name

☐ Replace spaces

Replace spaces with:

Any spaces in the group name will be replaced with this.

Warning: After creating a group you won't be able to change the Corp and Alliance group prefixes, name source and the replace spaces settings. Make sure you configure these the way you want before creating the config. If you need to change these you will have to create a new autogroup config.

- States selects which states will be added to automatic Corp/Alliance groups
- Corp/Alliance groups checkbox toggles Corp/Alliance autogroups on or off for this config.
- Corp/Alliance group prefix sets the prefix for the group name, e.g. if your Corp was called `MyCorp` and your prefix was `Corp`, your autogroup name would be created as `Corp MyCorp`. This field accepts leading/trailing spaces.
- Corp/Alliance name source sets the source of the Corp/Alliance name used in creating the group name. Currently the options are Full name and Ticker.
- Replace spaces allows you to replace spaces in the autogroup name with the value in the Replace spaces with field. This can be blank.

Permissions

Auto Groups are configured via models in the Admin Interface, a user will require the `Staff` Flag in addition to the following permissions.

Permission	Admin Site	Auth Site
<code>eve_autogroups.add_autogroupsconfig</code>	Can create model	None.
<code>eve_autogroups.change_autogroupsconfig</code>	Can edit model	None.
<code>eve_autogroups.delete_autogroupsconfig</code>	Can delete model	None.


There exists more models that will be automatically created and maintained by this module, they do not require end-user/admin interaction. `managedalliancegroup` `managedcorpgroups`

2.4.2 Corporation Stats

This module is used to check the registration status of Corp members and to determine character relationships, being mains or alts.


Corporation Member Data

Corporations ▾
Add








VOLTAGE REGULATORS

Mains (1)
Members (17)
Unregistered (16)

Last update: 5 days, 6 hours ago


Show entries

		Character	Corporation	Alliance	
		Emily Kasenumi	24th Imperial Crusade	None	
Naoko Kobayashi		Naoko Kobayashi	VOLTAGE REGULATORS	None	

Showing 1 to 1 of 1 entries

Previous
1
Next

Installation

Corp Stats requires access to the `esi-corporations.read_corporation_membership.v1` SSO scope. Update your application on the [EVE Developers site](#) to ensure it is available.

Add `'allianceauth.corputils'`, to your `INSTALLED_APPS` list in your auth project's settings file. Run migrations to complete installation.

Creating a Corp Stats

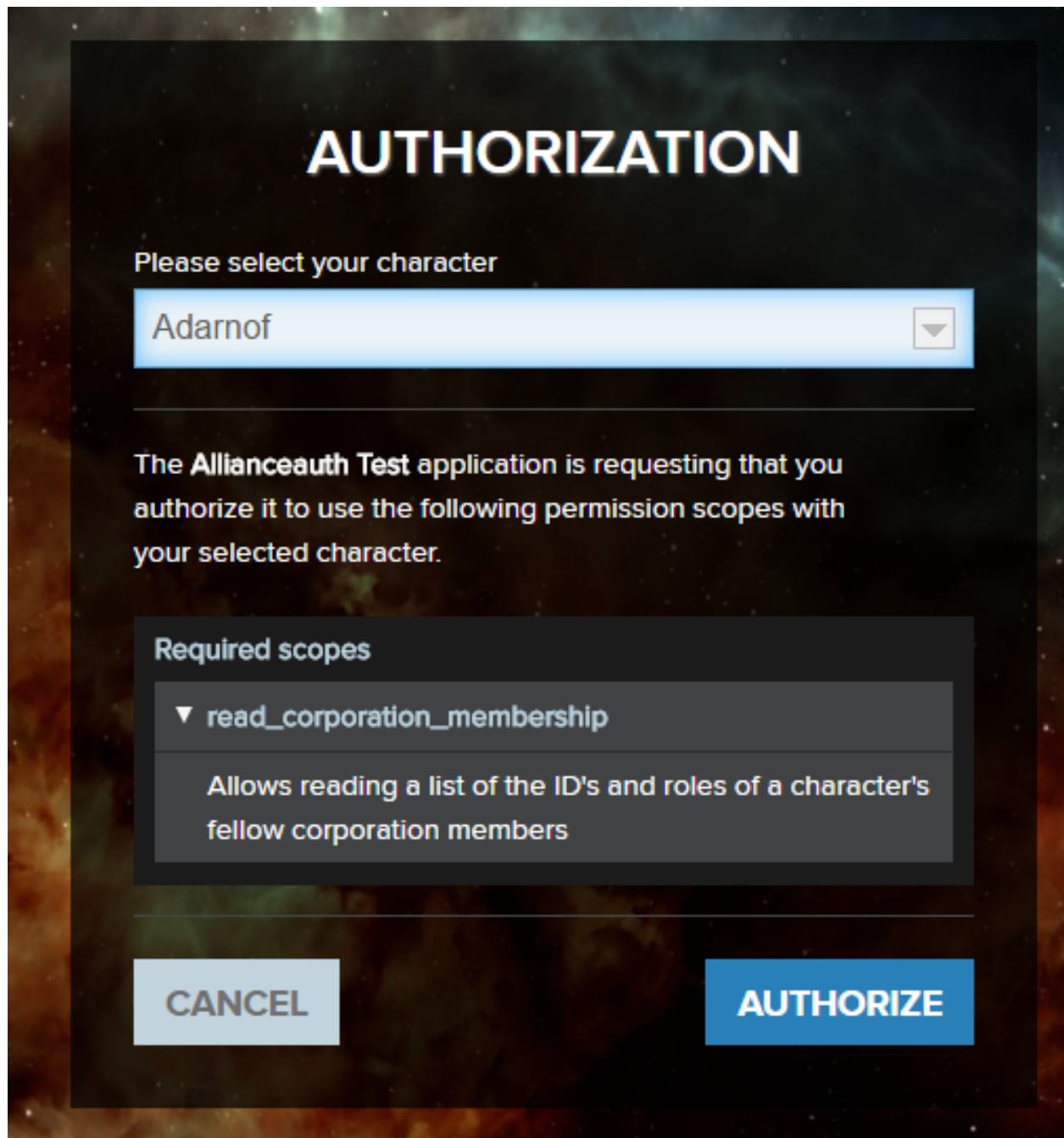
Upon initial install, nothing will be visible. For every Corp, a model will have to be created before data can be viewed.

Corporation Member Data

Corporations ▾
Add









If you are a superuser, the add button will be immediate visible to you. If not, your user account requires the `add_corpstats` permission.

Corp Stats requires an EVE SSO token to access data from the EVE Swagger Interface. Upon pressing the Add button, you will be prompted to authenticated. Please select the character who is in the Corporation you want data for.



You will return to auth where you are asked to select a token with the green arrow button. If you want to use a different character, press the LOG IN with EVE Online button.

Select a Token

	Character	Scopes	
	Adarnof	read_corporation_membership	
		read_corporation_membership	
		read_corporation_membership	



If this works (and you have permission to view the Corp Stats you just created) you'll be returned to a view of the Corp Stats. If it fails an error message will be displayed.

Corp Stats View


Navigation Bar

Corporations ▾ Add

This bar contains a dropdown menu of all available Corporations. If the user has the `add_corpstats` permission, a button to add a Corp Stats will be shown.

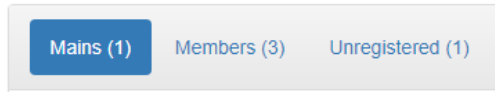
On the right of this bar is a search field. Press enter to search. It checks all characters in all Corp Stats you have view permission to and returns search results.

Last Update

Last update: 27 minutes ago 

An update can be performed immediately by pressing the update button. Anyone who can view the Corp Stats can update it.

Character Lists



Three views are available:

- main characters and their alts
- registered characters and their main character
- unregistered characters

Each view contains a sortable and searchable table. The number of listings shown can be increased with a dropdown selector. Pages can be changed using the controls on the bottom-right of the table. Each list is searchable at the top-right. Tables can be re-ordered by clicking on column headings.

	Character		Main Character	Main Corporation	Main Alliance
	Adarnof	Killboard	Adarnof	Super Secret Spaceship Syndicate	You Are Being Monitored
	Joringer Fidard	Killboard	Adarnof	Super Secret Spaceship Syndicate	You Are Being Monitored
	Erafius Arzi	Killboard			

Showing 1 to 3 of 3 entries

Main List

	Character	Corporation	Alliance	
 Adarnof	Adarnof	Super Secret Spaceship Syndicate	You Are Being Monitored	Killboard
	Joringer Fidard	Super Secret Spaceship Syndicate	You Are Being Monitored	Killboard

Showing 1 to 1 of 1 entries

This list contains all main characters in registered in the selected Corporation and their alts. Each character has a link to [zKillboard](#).

Member List

Mains (1)

Members (3)

Unregistered (1)

Last update: 1 week ago

Show

10

entries

Search:

Character

Main Character

Main Corporation

Main Alliance

Adamof

Killboard

Adamof

Super Secret Spaceship Syndicate

You Are Being Monitored

Erafius Arzi

Killboard

Joringer Fidard

Killboard

Adamof

Super Secret Spaceship Syndicate

You Are Being Monitored

Showing 1 to 3 of 3 entries






Previous

1

Next

The list contains all characters in the Corporation. Red backgrounds means they are not registered in auth. A link to [zKillboard](#) is present for all characters. If registered, the character will also have a main character, main Corporation, and main Alliance field.

Unregistered List

Mains (1) Members (3) Unregistered (1)			Last update: 1 week ago 		
Show	10	entries	Search: <input type="text"/>		
	Character				
	Erafius Arzi				
Showing 1 to 1 of 1 entries			Previous 1 Next		

This list contains all characters not registered on auth. Each character has a link to [zKillboard](#).


Search View

Corporations ▾
Add

Search Results

Show entries

Search:

↓ ↑	Character	↑ ↓	Corporation	↑ ↓	zKillboard	↑ ↓	Main Character	↑ ↓	Main Corporation	↑ ↓	Main Alliance	↑ ↓
	Adarnof		Super Secret Spaceship Syndicate		<div>Killboard</div>		Adarnof		Super Secret Spaceship Syndicate		You Are Being Monitored	

Showing 1 to 1 of 1 entries

Previous
1
Next

This view is essentially the same as the Corp Stats page, but not specific to a single Corporation. The search query is visible in the search box. Characters from all Corp Stats to which the user has view access will be displayed. APIs respect permissions.

Permissions

To use this feature, users will require some of the following:

Permission	Admin Site	Auth Site
corpstats.view_corp_corpstats	None	Can view corp stats of their corporation.
corpstats.view_alliance_corpstats	None	Can view corp stats of members of their alliance.
corpstats.view_state_corpstats	None	Can view corp stats of members of their auth state.
corpstats.add_corpstats	Can create model	Can add new corpstats using an SSO token.

Users who add a Corp Stats with their token will be granted permissions to view it regardless of the above permissions. View permissions are interpreted in the “OR” sense: a user can view their corporation’s Corp Stats without the `view_corp_corpstats` permission if they have the `view_alliance_corpstats` permission, same idea for their state. Note that these evaluate against the user’s main character.

Automatic Updating

By default Corp Stats are only updated on demand. If you want to automatically refresh on a schedule, add an entry to your project’s settings file:

```
CELERYBEAT_SCHEDULE['update_all_corpstats'] = {
    'task': 'allianceauth.corputils.tasks.update_all_corpstats',
    'schedule': crontab(minute=0, hour="*/6"),
}
```

Adjust the crontab as desired.

Troubleshooting

Failure to create Corp Stats

Unrecognized corporation. Please ensure it is a member of the alliance or a blue.

Corp Stats can only be created for Corporations who have a model in the database. These only exist for tenant corps, corps of tenant alliances, blue corps, and members of blue alliances.

Selected corp already has a statistics module.

Only one Corp Stats may exist at a time for a given Corporation.

Failed to gather corporation statistics with selected token.

During initial population, the EVE Swagger Interface did not return any member data. This aborts the creation process. Please wait for the API to start working before attempting to create again.

Failure to update Corp Stats

Any of the following errors will result in a notification to the owning user, and deletion of the Corp Stats model.

Your token has expired or is no longer valid. Please add a new one to create a new CorpStats.

This occurs when the SSO token is invalid, which can occur when deleted by the user, the character is transferred between accounts, or the API is having a bad day.

CorpStats for (corp name) cannot update with your ESI token as you have left corp.

The SSO token's character is no longer in the Corporation which the Corp Stats is for, and therefore membership data cannot be retrieved.

HTTPForbidden

The SSO token lacks the required scopes to update membership data.

2.4.3 Fleet Activity Tracking

The Fleet Activity Tracking (FAT) app allows you to track fleet participation.

Participation data

Most recent clicked fatlinks					Personal statistics
Fleet	Character	System	Ship	Eve Time	
Dummy	Veronica Blomquist	Docked in Amamake	Capsule	Dec. 16, 2019, 12:29 a.m.	
Dummy	Erik Kalkoken	Docked in Amamake	Capsule	Dec. 16, 2019, 12:29 a.m.	
Most recent fatlinks					View statistics Create fatlink
Name	Creator	Fleet	Eve Time	Duration	Edit
Dummy	Erik_Kalkoken	Dummy	Dec. 16, 2019, 12:29 a.m.	30	

Installation

Fleet Activity Tracking requires access to the `esi-location.read_location.v1`, `esi-location.read_ship_type.v1`, and `esi-universe.read_structures.v1` SSO scopes. Update your application on the [EVE Developers site](#) to ensure these are available.

Add `'allianceauth.fleetactivitytracking'`, to your `INSTALLED_APPS` list in your auth project's settings file. Run migrations to complete installation.

Permissions

To administer this feature, users will require some of the following.

Users do not require any permissions to interact with FAT Links created.

Permission	Admin Site	Auth Site
<code>auth.fleetactivitytracking</code>	None	Create and Modify FATLinks
<code>auth.fleetactivitytracking_statistics</code>	None	Can view detailed statistics for corp models and other characters.

2.4.4 HR Applications


This app allows you to manage applications for multiple corporations in your alliance. Key features include:

- Define application questionnaires for corporations
- Users can apply to corporations by filling out questionnaires
- Manage review and approval process of applications

Personal Applications

				Create Application
Username	Corporation	Status	Actions	
Erik_Kalkoken	project HAVEN	Pending	 	

Application Management

						Search Applications
Pending	Reviewed					
Date	Username	Main Character	Corporation	Status	Actions	
Dec. 16, 2019, 12:40 a.m.	Erik_Kalkoken	Erik Kalkoken	project HAVEN	Pending		

Installation

Add `'allianceauth.hrapplications'`, to your `INSTALLED_APPS` list in your auth project's settings file. Run migrations to complete installation.

Management

Creating Forms

The most common task is creating `ApplicationForm` models for corps. Only when such models exist will a Corporation be listed as a choice for applicants. This occurs in the Django admin site, so only staff have access.

The first step is to create questions. This is achieved by creating `ApplicationQuestion` models, one for each question. Titles are not unique.

Next step is to create the actual `ApplicationForm` model. It requires an existing `EveCorporationInfo` model to which it will belong. It also requires the selection of questions. `ApplicationForm` models are unique per Corporation: only one may exist for any given Corporation concurrently.

You can adjust these questions at any time. This is the preferred method of modifying the form: deleting and recreating will cascade the deletion to all received applications from this form which is usually not intended.

Once completed the Corporation will be available to receive applications.

Reviewing Applications

Superusers can see all applications, while normal members with the required permission can view only those to their corp.

Selecting an application from the management screen will provide all the answers to the questions in the form at the time the user applied.

When a reviewer assigns themselves an application, they mark it as in progress. This notifies the applicant and permanently attached the reviewer to the application.

Only the assigned reviewer can approve/reject/delete the application if they possess the appropriate permission.

Any reviewer who can see the application can view the applicant's APIs if they possess the appropriate permission.

Permissions

To administer this feature, users will require some of the following.

Users do not require any permission to apply to a corporation and fill out the form.

Permission	Admin Site	Auth Site
<code>auth.human_resources</code>	None	Can view applications and mark in progress
<code>hrapplications.approve_application</code>	None	Can approve applications
<code>hrapplications.delete_application</code>	Can delete model	Can delete applications
<code>hrapplications.reject_applications</code>	None	Can reject applications
<code>hrapplications.add_applicationcomment</code>	Can create model	Can comment on applications

A user with `auth.human_resources` can only see applications to their own corp.

Best practice is to bundle the `auth.human_resources` permission alongside the `hrapplications.approve_application` and `hrapplications.reject_application` permissions, as in isolation these don't make much sense.

Models

ApplicationQuestion

This is the model representation of a question. It contains a title, and a field for optional “helper” text. It is referenced by `ApplicationForm` models but acts independently. Modifying the question after it has been created will not void responses, so it's not advisable to edit the title or the answers may not make sense to reviewers.

ApplicationForm

This is the template for an application. It points at a Corporation, with only one form allowed per Corporation. It also points at `ApplicationQuestion` models. When a user creates an application, they will be prompted with each question the form includes at the given time. Modifying questions in a form after it has been created will not be reflected in existing applications, so it's perfectly fine to adjust them as you see fit. Changing Corporations however is not advisable, as existing applications will point at the wrong Corporation after they've been submitted, confusing reviewers.

Application

This is the model representation of a completed application. It references an `ApplicationForm` from which it was spawned which is where the Corporation specificity comes from. It points at a user, contains info regarding its reviewer, and has a status. Shortcut properties also provide the applicant's main character, the applicant's APIs, and a string representation of the reviewer (for cases when the reviewer doesn't have a main character or the model gets deleted).

ApplicationResponse

This is an answer to a question. It points at the `Application` to which it belongs, to the `ApplicationQuestion` which it is answering, and contains the answer text. Modifying any of these fields is dangerous.

ApplicationComment

This is a reviewer's comment on an application. Points at the application, points to the user, and contains the comment text. Modifying any of these fields is dangerous.

Troubleshooting

No corps accepting applications

Ensure there are `ApplicationForm` models in the admin site. Ensure the user does not already have an application to these Corporations. If the users wishes to re-apply they must first delete their completed application

Reviewer unable to complete application

Reviewers require a permission for each of the three possible outcomes of an application, Approve Reject or Delete. Any user with the human resources permission can mark an application as in-progress, but if they lack these permissions then the application will get stuck. Either grant the user the required permissions or change the assigned reviewer in the admin site. Best practice is to bundle the `auth.human_resources` permission alongside the `hrapplications.approve_application` and `hrapplications.reject_application` permissions, as in isolation these don't serve much purpose.

2.4.5 Fleet Operations

Fleet Operations is an app for organizing and communicating fleet schedules.

Fleet Operation Timers									
Current Ops Times: Friday February 21, 2020 20:22:50									
Create Operation									
Next Timers									
Operation Name	Doctrine	Form Up System	Start Time	Local Time	Duration	FC	Creator	Action	
Test the West	Leshaks	Abune	2020-02-29 22:56	Sat @ 11:56 PM 8d 2h 33m 9s	1h	Chuck Norris	Erik Kalkoken		
Past Timers									
Operation Name	Doctrine	Form Up System	Start Time	Local Time	Duration	FC	Creator	Action	
PVP	Super Secret	Abune	2020-02-06 19:19	Thu @ 8:19 PM	1h	Chuck Norris	Erik Kalkoken		
Dank Brawl	Carriers	Amamake	2020-02-02 23:13	Mon @ 12:13 AM	1h	Chuck Norris	Erik Kalkoken		

Installation

Add `'allianceauth.optimer'`, to your `INSTALLED_APPS` list in your auth project's settings file. Run migrations to complete installation.

Permissions

To use and administer this feature, users will require some of the following.

Permission	Admin Site	Auth Site
<code>auth.optimer_view</code>	None	Can view Fleet Operation Timers
<code>auth.optimer_manage</code>	None	Can Manage Fleet Operation timers

2.4.6 Permissions Auditing

Access to most of Alliance Auth's features are controlled by Django's permissions system. In order to help you secure your services, Alliance Auth provides a permissions auditing tool.

This is an optional app that needs to be installed.

To install it add `'allianceauth.permissions_tool'`, to your `INSTALLED_APPS` list in your auth project's settings file.

Usage

Access

In order to grant users access to the permissions auditing tool they will need to be granted the `permissions_tool.audit_permissions` permission or be a superuser.

When a user has access to the tool they will see the “Permissions Audit” menu item under the “Util” sub menu.

Permissions Overview

The first page gives you a general overview of permissions and how many users have access to each permission.

Permissions Overview

Showing only applied permissions [Show All](#)

Filter (App) (Model)

Show 10 entries

Search:

Model	Code Name	Name	Users	Groups	States
auth					
user	optimizer_management	optimizer_management	0	1 (1)	0 (0)
user	optimizer_view	optimizer_view	1	1 (1)	1 (871)
user	timer_view	timer_view	0	0 (0)	1 (871)
corputils					
corpstats	add_corpstats	Can add corp stats	0	0 (0)	1 (871)
corpstats	view_alliance_corpstats	Can view corp stats of members of their alliance.	0	0 (0)	1 (871)
corpstats	view_corp_corpstats	Can view corp stats of their corporation.	0	0 (0)	1 (871)
corpstats	view_state_corpstats	Can view corp stats of members of their auth state.	0	0 (0)	1 (871)
fleetactivitytracking					
fat	add_fat	Can add fat	0	0 (0)	1 (871)
groupmanagement					
authgroup	request_groups	Can request non-public groups	0	0 (0)	1 (871)
hrapplications					
application	add_application	Can add application	0	0 (0)	1 (871)

Showing 1 to 10 of 12 entries

Previous 1 2 Next

App, **Model** and **Code Name** contain the internal details of the permission while **Name** contains the name/description you’ll see in the admin panel.

Users is the number of users explicitly granted this permission on their account.

Groups is the number of groups with this permission assigned.

Groups Users is the total number of users in all of the groups with this permission assigned.

Clicking on the **Code Name** link will take you to the [Permissions Audit Page](#)

Permissions Audit Page

The permissions audit page will give you an overview of all the users who have access to this permission either directly or granted via group membership.

Permissions Audit: optimer_view

[< Back](#)

Filter (Source)
Show 10 entries
Search:

User / Character
Organization

Group: Lumberjacks

Naoko_Kobayashi
Naoko Kobayashi

VOLTAGE REGULATORS

State: Member

1234_Anemone
1234 Anemone

CLOUD TEMPLE
Test Alliance Please Ignore

Abigail_Flux
Abigail Flux

Minimatar Death Squad
Test Alliance Please Ignore

Acabar_Ghasha
Acabar Ghasha

Wombo United
Test Alliance Please Ignore

Acura_Yong
Acura Yong

Soviet Union
Test Alliance Please Ignore

Aden_Parmala
Aden Parmala

hrrr
Test Alliance Please Ignore

Aderyn_Brody
Aderyn Brody

Worthless Carebears
Test Alliance Please Ignore

Adfogul_Kemersesti
Adfogul Kemersesti

Soviet Union
Test Alliance Please Ignore

Admiral_Trump
Admiral Trump

Upvote
Test Alliance Please Ignore

Ahill_Jr
Ahill Jr

XAOS RELOADED
Test Alliance Please Ignore

Showing 1 to 10 of 873 entries

Previous
1
2
3
4
5
...
88
Next

Please note that users may appear multiple times if this permission is granted via multiple sources.

Permissions

To use this feature, users will require some of the following.

Permission	Admin Site	Auth Site
permissions_tool.audit_permissions	None	Can view the Permissions Audit tool

2.4.7 Ship Replacement

Ship Replacement helps you to organize ship replacement programs (SRP) for your alliance.

SRP Management

View All

Add SRP Fleet

Total ISK Cost: 0

Fleet Name	Fleet Time	Fleet Doctrine	Fleet Commander	Fleet AAR	Fleet SRP Code	Fleet ISK Cost	SRP Status	Pending Requests	Actions
<div>Defend our home</div>	2020-02-20 21:00	Caracals	<div>Erik Kaikkonen</div>	<div>Link</div>	<div>66C003A5</div>	ISK: 0	<div>Pending</div>	<div>1</div>	<div><div></div><div></div><div></div><div></div></div>

Installation

Add `'allianceauth.srp'`, to your `INSTALLED_APPS` list in your auth project's settings file. Run migrations to complete installation.

Permissions

To use and administer this feature, users will require some of the following.

Permission	Admin Site	Auth Site
<code>auth.access_srp</code>	None	Can create an SRP request from a fleet
<code>auth.srp_management</code>	None	Can Approve and Deny SRP requests, Can create an SRP Fleet
<code>srp.add_srpfleetmain</code>	Can Add Model	Can Create an SRP Fleet

2.4.8 Structure Timers



Structure Timers helps you keep track of both offensive and defensive structure timers in your space.

Structure Timers







[Create Structure Timer](#)

Next Timers

Current Eve Time: Friday February 21, 2020 20:32:37

Details	Objective	System	Structure	Eve Time	Local Time	Creator	Action
Extraction ready	Friendly	Amamake II / M4	Moon Mining Cycle	2020-02-25 20:32	Tue @ 9:32 PM 3d 23h 59m 31s	Erik Kalkoken	 

Past Timers

Details	Objective	System	Structure	Eve Time	Local Time	Creator	Action
Test 4	Friendly	Amamake II/3	Azbel	2020-02-01 22:05	Sat @ 11:05 PM	Erik Kalkoken	 
Test Timer	Friendly	Abune	POCO	2020-02-03 10:05	Mon @ 11:05 AM	Erik Kalkoken	 
Tes 2	Friendly	Jita II/4	Azbel	2020-02-03 18:08	Mon @ 7:08 PM	Erik Kalkoken	 
Faction Fort	Hostile	Bosbooger	Fortizar	2020-02-19 21:59	Wed @ 10:59 PM	Erik Kalkoken	 
XYZ	Hostile	Abune	Sotyo	2020-02-20 23:59	Fri @ 12:59 AM	Erik Kalkoken	 

Installation

Add `'allianceauth.timerboard'`, to your `INSTALLED_APPS` list in your auth project's settings file. Run migrations to complete installation.

Permissions

To use and administer this feature, users will require some of the following.

Permission	Admin Site	Auth Site
<code>auth.timer_view</code>	None	Can view Timerboard Timers
<code>auth.timer_manage</code>	None	Can Manage Timerboard timers

2.5 Community Contributions

Another key feature of **Alliance Auth** is that it can be easily extended. Our great community is providing a variety of plug-in apps and services, which you can choose from to add more functions to your AA installation.

Check out the [Community Creations](#) repo for more details.

Or if you have specific needs you can of course always develop your own plugin- apps and services. Please see the [Development](#) chapter for details.

MAINTENANCE

In the maintenance chapter you find details about where important log files are found, how you can customize your AA installation and how to solve common issues.

3.1 App Maintenance

3.1.1 Adding and Removing Apps

Your auth project is just a regular Django project - you can add in [other Django apps](#) as desired. Most come with dedicated setup guides, but here is the general procedure:

1. add 'appname', to your `INSTALLED_APPS` setting in `local.py`
2. `run python manage.py migrate`
3. `run python manage.py collectstatic`
4. restart AA with `supervisorctl restart myauth:`

If you ever want to remove an app, you should first clear it from the database to avoid dangling foreign keys: `python manage.py migrate appname zero`. Then you can remove it from your auth project's `INSTALLED_APPS` list.

3.1.2 Permission Cleanup

Mature Alliance Auth installations, or those with actively developed extensions may find themselves with stale or duplicated Permission models.

This can make it confusing for admins to apply the right permissions, contribute to larger queries in backend management or simply look unsightly.

```
python manage.py remove_stale_contenttypes --include-stale-apps
```

This inbuilt Django command will step through each contenttype and offer to delete it, displaying what exactly this will cascade to delete. Pay attention and ensure you understand exactly what is being removed before answering `yes`.

This should only cleanup uninstalled apps, deprecated permissions within apps should be cleaned up using Data Migrations by each responsible application.

3.2 Folder structure

When installing Alliance Auth you are instructed to run the `allianceauth start` command which generates a folder containing your auth project. This auth project is based off Alliance Auth but can be customized how you wish.

3.2.1 The myauth folder

The first folder created is the root directory of your auth project. This folder contains:

- the `manage.py` management script used to interact with Django
- a preconfigured `supervisor.conf` Supervisor config for running Celery (and optionally Gunicorn) automatically
- a `log` folder which contains log files generated by Alliance Auth

3.2.2 The myauth subfolder

Within your auth project root folder is another folder of the same name (a quirk of Django project structures). This folder contains:

- a Celery app definition in `celery.py` for registering tasks with the background workers
- a web server gateway interface script `wsgi.py` for processing web requests
- the root URL config `urls.py` which Django uses to direct requests to the appropriate view

There are also two subfolders for `static` and `templates` which allow adding new content and overriding default content shipped with Alliance Auth or Django.

And finally the settings folder.

3.2.3 Settings Files

With the settings folder lives two settings files: `base.py` and `local.py`

The base settings file contains everything needed to run Alliance Auth. It handles configuration of Django and Celery, defines logging, and many other Django-required settings. This file should not be edited. While updating Alliance Auth you may be instructed to update the base settings file - this is achieved through the `allianceauth update` command which overwrites the existing base settings file.

The local settings file is referred to as “your auth project’s settings file” and you are instructed to edit it during the install process. You can add any additional settings required by other apps to this file. Upon creation the first line is `from .base import *` meaning all settings defined in the base settings file are loaded. You can override any base setting by simply redefining it in your local settings file.

3.2.4 Log Files

Your auth project comes with four log file definitions by default. These are created in the `myauth/log/` folder at runtime.

- `allianceauth.log` contains all INFO level and above logging messages from Alliance Auth. This is useful for tracking who is making changes to the site, what is happening to users, and debugging any errors that may occur.
- `worker.log` contains logging messages from the Celery background task workers. This is useful for monitoring background processes such as group syncing to services.
- `beat.log` contains logging messages from the background task scheduler. This is of limited use unless the scheduler isn't starting.
- `gunicorn.log` contains logging messages from Gunicorn workers. This contains all web-sourced messages found in `allianceauth.log` as well as runtime errors from the workers themselves.

When asking for assistance with your auth project be sure to first read the logs, and share any relevant entries.

3.3 Troubleshooting

3.3.1 Logging

In its default configuration your auth project logs INFO and above messages to `myauth/log/allianceauth.log`. If you're encountering issues it's a good idea to view DEBUG messages as these greatly assist the troubleshooting process. These are printed to the console with manually starting the webserver via `python manage.py runserver`.

To record DEBUG messages in the log file, alter a setting in your auth project's settings file: `LOGGING['handlers']['log_file']['level'] = 'DEBUG'`. After restarting gunicorn and celery your log file will record all logging messages.

3.3.2 Common Problems

I'm getting an error 500 trying to connect to the website on a new install

Great. Error 500 is the generic message given by your web server when *anything* breaks. The actual error message is hidden in one of your auth project's log files. Read them to identify it.

Failed to configure log handler

Make sure the log directory is writeable by the `allianceserver` user: `chmown -R allianceserver:allianceserver /path/to/myauth/log/`, then restart the auth supervisor processes.

Groups aren't syncing to services

Make sure the background processes are running: `supervisorctl status myauth:.` If `myauth:worker` or `myauth:beat` do not show `RUNNING` read their log files to identify why.

Task queue is way too large

Stop celery workers with `supervisorctl stop myauth:worker` then clear the queue:

```
redis-cli FLUSHALL
celery -A myauth worker --purge
```

Press Control+C once.

Now start the worker again with `supervisorctl start myauth:worker`

Proxy timeout when entering email address

This usually indicates an issue with your email settings. Ensure these are correct and your email server/service is properly configured.

No images are available to users accessing the website

This is likely due to a permissions mismatch. Check the setup guide for your web server. Additionally ensure the user who owns `/var/www/myauth/static` is the same user as running your webserver, as this can be non-standard.

Unable to execute 'gunicorn myauth.wsgi' or ImportError: No module named 'myauth.wsgi'

Gunicorn needs to have context for its running location, `/home/allianceserver/myauth/gunicorn myauth.wsgi` will not work, instead `cd /home/allianceserver/myauth then gunicorn myauth.wsgi` is needed to boot Gunicorn. This is handled in the Supervisor config, but this may be encountered running Gunicorn manually for testing.

Specified key was too long error

Migrations may about with the following error message:

```
Specified key was too long; max key length is 767 bytes
```

This error will occur if one is trying to use Maria DB prior to 10.2.x, which is not compatible with Alliance Auth. Install a newer Maria DB version to fix this issue another DBMS supported by Django 2.2.

3.4 Tuning

The official installation guide will install a stable version of Alliance Auth that will work fine for most cases. However, there are a lot of levels that can be used to optimize a system. For example some installations may be short on RAM and want to reduce the total memory footprint, even though that may reduce system performance. Others are fine with further increasing the memory footprint to get better system performance.

Warning: Tuning usually has benefits and costs and should only be performed by experienced Linux administrators who understand the impact of tuning decisions on their system.

3.4.1 Gunicorn

Number of workers

The default installation will have 3 workers configured for Gunicorn. This will be fine on most system, but if your system has more than one core than you might want to increase the number of workers to get better response times. Note that more workers will also need more RAM though.

The number you set this to will depend on your own server environment, how many visitors you have etc. Gunicorn suggests $(2 \times \text{\$num_cores}) + 1$ for the number of workers. So for example if you have 2 cores you want $2 \times 2 + 1 = 5$ workers. See [here](#) for the official discussion on this topic.

For example to get 5 workers change the setting `--workers=5` in your `supervisor.conf` file and then reload the supervisor with the following command to activate the change (Ubuntu):

```
systemctl restart supervisor
```

3.4.2 Celery

Hint: Most tunings will require a change to your supervisor configuration in your `supervisor.conf` file. Note that you need to restart the supervisor daemon in order for any changes to take effect. And before restarting the daemon you may want to make sure your supervisors stop gracefully:(Ubuntu):

```
supervisor stop myauth:
systemctl supervisor restart
```

Task Logging

By default task logging is deactivated. Enabling task logging allows you to monitor what tasks are doing in addition to getting all warnings and error messages. To enable info logging for tasks add the following to the command configuration of your worker in the `supervisor.conf` file:

```
-l info
```

Full example:

```
command=/home/allianceserver/venv/auth/bin/celery -A myauth worker -l info
```

Protection against memory leaks

Celery workers often have memory leaks and will therefore grow in size over time. While the Alliance Auth team is working hard to ensure Auth is free of memory leaks some may still be caused by bugs in different versions of libraries or community apps. It is therefore good practice to enable features that protect against potential memory leaks.

There are two ways to protect against memory leaks:

- Worker
- Supervisor

Worker

Celery workers can be configured to automatically restart if they grow above a defined memory threshold. Restarts will be graceful, so current tasks will be allowed to complete before the restart happens.

To add protection against memory leaks add the following to the command configuration of your worker in the `supervisor.conf` file. This sets the upper limit to 256MB.

```
--max-memory-per-child 262144
```

Full example:

```
command=/home/allianceserver/venv/auth/bin/celery -A myauth worker --max-memory-per-  
↪child 262144
```

Hint: The 256 MB limit is just an example and should be adjusted to your system configuration. We would suggest to not go below 128MB though, since new workers start with around 80 MB already. Also take into consideration that this value is per worker and that you properly have more than one worker running in your system (if your workers run as processes, which is the default).

Warning: The `max-memory-per-child` parameter only works when workers run as processes (which is the default). It does not work for threads.

Note: Alternatively, you can also limit the number of runs per worker until a restart is performed with the worker parameter `max-tasks-per-child`. This can also protect against memory leaks if you set the threshold is low enough. However, it is less precise since than using `max-memory-per-child`.

See also the [official Celery documentation](#) for more information about these two worker parameters.

Supervisor

It is also possible to configure your supervisor to monitor and automatically restart programs that exceed a memory threshold.

This is not a built in feature and requires the 3rd party extension [superlance](#), which includes a set of plugin utilities for supervisor. The one that watches memory consumption is [memmon](#).

To setup install superlance into your venv with:

```
pip install superlance
```

You can then add memmon to your `supervisor.conf`. Here is an example setup with a worker that runs with gevent:

```
[eventlistener:memmon]
command=/home/allianceserver/venv/auth/bin/memmon -p worker=512MB
directory=/home/allianceserver/myauth
events=TICK_60
```

This setup will check the memory consumption of the program “worker” every 60 secs and automatically restart it if it goes above 512 MB. Note that it will use the stop signal configured in supervisor, which is `TERM` by default. `TERM` will cause a “warm shutdown” of your worker, so all currently running tasks are completed before the restart.

Again, the 512 MB is just an example and should be adjusted to fit your system configuration.

Increasing task throughput

Celery tasks are designed to run concurrently, so one obvious way to increase task throughput is run more tasks in parallel.

Concurrency

This can be achieved by the setting the concurrency parameter of the celery worker to a higher number. For example:

```
--concurrency=4
```

However, there is a catch: In the default configuration each worker will spawn as it’s own process. So increasing the number of workers will increase both CPU load and memory consumption in your system.

The recommended number of workers is one per core, which is what you get automatically with the default configuration. Going beyond that can quickly reduce your overall system performance. i.e. the response time for Alliance Auth or other apps running on the same system may take a hit while many tasks are running.

Hint: The optimal number will hugely depend on your individual system configuration and you may want to experiment with different settings to find the optimal. One way to generate task load and verify your configuration is to run a model update with the following command:

```
celery -A myauth call allianceauth.eveonline.tasks.run_model_update
```

Processes vs. Threads

A better way to increase concurrency without impacting is to switch from processes to threads for celery workers. In general celery workers perform better with processes when tasks are primarily CPU bound. And they perform better with threads when tasks that are primarily I/O bound.

Alliance Auth tasks are primarily I/O bound (most tasks are fetching data from ESI and/or updating the local database), so threads are clearly the better choice for Alliance Auth. However, there is a catch. Celery's out-of-the-box support for threads is limited and additional packages and configurations is required to make it work. Nonetheless, the performance gain - especially in smaller systems - is significant, so it may well be worth the additional configuration complexity.

Warning: One important feature that no longer works with threads is the worker parameter `--max-memory-per-child` that protects against memory leaks. But you can alternatively use *supervisor* to monitor and restart your workers.

See also the also [this guide](#) on more information about how to configure the execution pool for workers.

Setting up for threads

First, you need to install a threads packages. Celery supports both gevent and eventlet. We will go with gevent, since it's newer and better supported. Should you encounter any issues with gevent, you may want to try eventlet.

To install gevent make sure you are in your venv and install the following:

```
pip install gevent
```

Next we need to reconfigure the workers to use gevent threads. For that add the following parameters to your worker config:

```
--pool=gevent --concurrency=10
```

Full example:

```
command=/home/allianceserver/venv/auth/bin/celery -A myauth worker --pool=gevent --  
↪ concurrency=10
```

Make sure to restart supervisor to activate the changes.

Hint: The optimal number of concurrent workers will be different for every system and we recommend experimenting with different figures to find the optimal for your system. Note, that the example of 10 threads is conservative and should work even with smaller systems.

SUPPORT

If you encounter any AA related issues during installation or otherwise please first check the following resources:

- See the section on [troubleshooting](#) your AA instance, e.g. the list of common problems
- Search the AA [issue list](#) (especially the closed ones)

No solution?

- Open an [issue](#)
- Ask for help on our [Discord](#)

CUSTOMIZING

It is possible to customize your **Alliance Auth** instance.

Warning: Keep in mind that you may need to update some of your customizations manually after new Auth releases (e.g. when replacing templates).

5.1 Site name

You can replace the default name shown on the web site with your own, e.g. the name of your Alliance.

Just update `SITE_NAME` in your `local.py` settings file accordingly, e.g.:

```
SITE_NAME = 'Awesome Alliance'
```

5.2 Custom Static and Templates

Within your auth project exists two folders named `static` and `templates`. These are used by Django for rendering web pages. Static refers to content Django does not need to parse before displaying, such as CSS styling or images. When running via a WSGI worker such as Gunicorn static files are copied to a location for the web server to read from. Templates are always read from the template folders, rendered with additional context from a view function, and then displayed to the user.

You can add extra static or templates by putting files in these folders. Note that changes to static requires running the `python manage.py collectstatic` command to copy to the web server directory.

It is possible to overload static and templates shipped with Django or Alliance Auth by including a file with the exact path of the one you wish to overload. For instance if you wish to add extra links to the menu bar by editing the template, you would make a copy of the `allianceauth/templates/allianceauth/base.html` file to `myauth/templates/allianceauth/base.html` and edit it there. Notice the paths are identical after the `templates/` directory - this is critical for it to be recognized. Your custom template would be used instead of the one included with Alliance Auth when Django renders the web page. Similar idea for static: put CSS or images at an identical path after the `static/` directory and they will be copied to the web server directory instead of the ones included.

5.3 Custom URLs and Views

It is possible to add or override URLs with your auth project's URL config file. Upon install it is of the form:

```
import allianceauth.urls

urlpatterns = [
    url(r'', include(allianceauth.urls)),
]
```

This means every request gets passed to the Alliance Auth URL config to be interpreted.

If you wanted to add a URL pointing to a custom view, it can be added anywhere in the list if not already used by Alliance Auth:

```
import allianceauth.urls
import myauth.views

urlpatterns = [
    url(r'', include(allianceauth.urls)),
    url(r'myview/$', myauth.views.myview, name='myview'),
]
```

Additionally you can override URLs used by Alliance Auth here:

```
import allianceauth.urls
import myauth.views

urlpatterns = [
    url(r'account/login/$', myauth.views.login, name='auth_login_user'),
    url(r'', include(allianceauth.urls)),
]
```

DEVELOPMENT

Alliance Auth is designed to be extended easily. Learn how to develop your own apps and services for AA or to develop for AA core in the development chapter.

6.1 Custom apps and services

This section describes how to extend **Alliance Auth** with custom apps and services.

6.1.1 Integrating Services

One of the primary roles of Alliance Auth is integrating with external services in order to authenticate and manage users. This is achieved through the use of service modules.

The Service Module

Each service module is its own self contained Django app. It will likely contain views, models, migrations and templates. Anything that is valid in a Django app is valid in a service module.

Normally service modules live in `services.modules` though they may also be installed as external packages and installed via `pip` if you wish. A module is installed by including it in the `INSTALLED_APPS` setting.

Service Module Structure

Typically a service will contain 5 key components:

- *The Hook*
- *The Service Manager*
- *The Views*
- *The Tasks*
- *The Models*

The architecture looks something like this:

```
urls ----- Views
      |
      |           |
      |           |
```

(continues on next page)

(continued from previous page)

```
ServiceHook ---- Tasks ---- Manager
```

```
    |
    |
AllianceAuth
```

Where:

```
Module -- Dependency/Import
```

While this is the typical structure of the existing services modules, there is no enforcement of this structure and you are, effectively, free to create whatever architecture may be necessary. A service module need not even communicate with an external service, for example, if similar triggers such as `validate_user`, `delete_user` are required for a module it may be convenient to masquerade as a service. Ideally though, using the common structure improves the maintainability for other developers.

The Hook

In order to integrate with Alliance Auth service modules must provide a `services_hook`. This hook will be a function that returns an instance of the `services.hooks.ServiceHook` class and decorated with the `@hooks.registerhook` decorator. For example:

```
@hooks.register('services_hook')
def register_service():
    return ExampleService()
```

This would register the `ExampleService` class which would need to be a subclass of `services.hooks.ServiceHook`.

Important: The hook **MUST** be registered in `yourservice.auth_hooks` along with any other hooks you are registering for Alliance Auth.

A subclassed `ServiceHook` might look like this:

```
class ExampleService(ServicesHook):
    def __init__(self):
        ServicesHook.__init__(self)
        self.urlpatterns = urlpatterns
        self.service_url = 'http://exampleservice.example.com'

    """
    Overload base methods here to implement functionality
    """
```


The ServiceHook class

The base `ServiceHook` class defines function signatures that Alliance Auth will call under certain conditions in order to trigger some action in the service.

You will need to subclass `services.hooks.ServiceHook` in order to provide implementation of the functions so that Alliance Auth can interact with the service correctly. All of the functions are optional, so its up to you to define what you need.

Instance Variables:

- *self.name*
- *self.urlpatterns*
- *self.service_ctrl_template*

Properties:

- *title*

Functions:

- *delete_user*
- *validate_user*
- *sync_nickname*
- *sync_nicknames_bulk*
- *update_groups*
- *update_groups_bulk*
- *update_all_groups*
- *service_enabled_members*
- *service_enabled_blues*
- *service_active_for_user*
- *show_service_ctrl*
- *render_service_ctrl*

self.name

Internal name of the module, should be unique amongst modules.

self.urlpatterns

You should define all of your service URLs internally, usually in `urls.py`. Then you can import them and set `self.urlpatterns` to your defined urlpatterns.

```
from . import urls
...
class MyService(ServiceHook):
    def __init__(self):
        ...
        self.urlpatterns = urls.urlpatterns
```

All of your apps defined `urlpatterns` will then be included in the `URLconf` when the core application starts.

`self.service_ctrl_template`

This is provided as a courtesy and defines the default template to be used with `render_service_ctrl`. You are free to redefine or not use this variable at all.

`title`

This is a property which provides a user friendly display of your service's name. It will usually do a reasonably good job unless your service name has punctuation or odd capitalization. If this is the case you should override this method and return a string.

`delete_user`

```
def delete_user(self, user, notify_user=False):
```

Delete the users service account, optionally notify them that the service has been disabled. The `user` parameter should be a Django User object. If `notify_user` is set to `True` a message should be set to the user via the `notifications` module to alert them that their service account has been disabled.

The function should return a boolean, `True` if successfully disabled, `False` otherwise.

`validate_user`

```
def validate_user(self, user):
```

Validate the users service account, deleting it if they should no longer have access. The `user` parameter should be a Django User object.

An implementation will probably look like the following:

```
def validate_user(self, user):
    logger.debug('Validating user %s %s account' % (user, self.name))
    if ExampleTasks.has_account(user) and not self.service_active_for_user(user):
        self.delete_user(user, notify_user=True)
```

No return value is expected.

This function will be called periodically on all users to validate that the given user should have their current service accounts.

`sync_nickname`

```
def sync_nickname(self, user):
```

Very optional. As of writing only one service defines this. The `user` parameter should be a Django User object. When called, the given users nickname for the service should be updated and synchronized with the service.

If this function is defined, an admin action will be registered on the Django Users view, allowing admins to manually trigger this action for one or many users. The hook will trigger this action user by user, so you won't have to manage a list of users.

sync_nicknames_bulk

```
def sync_nicknames_bulk(self, users):
```

Updates the nickname for a list of users. The `users` parameter must be a list of Django User objects.

If this method is defined, the admin action for updating service related nicknames for users will call this bulk method instead of `sync_nickname`. This gives you more control over how mass updates are executed, e.g. ensuring updates do not run in parallel to avoid causing rate limit violations from an external API.

This is an optional method.

update_groups

```
def update_groups(self, user):
```

Update the users group membership. The `user` parameter should be a Django User object. When this is called the service should determine the groups the user is a member of and synchronize the group membership with the external service. If your service does not support groups then you are not required to define this.

If this function is defined, an admin action will be registered on the Django Users view, allowing admins to manually trigger this action for one or many users. The hook will trigger this action user by user, so you won't have to manage a list of users.

This action is usually called via a signal when a users group membership changes (joins or leaves a group).

update_groups_bulk

```
def update_groups_bulk(self, users):
```

Updates the group memberships for a list of users. The `users` parameter must be a list of Django User objects.

If this method is defined, the admin action for updating service related groups for users will call this bulk method instead of `update_groups`. This gives you more control over how mass updates are executed, e.g. ensuring updates do not run in parallel to avoid causing rate limit violations from an external API.

This is an optional method.

update_all_groups

```
def update_all_groups(self):
```

The service should iterate through all of its recorded users and update their groups.

I'm really not sure when this is called, it may have been a hold over from before signals started to be used. Regardless, it can be useful to server admins who may call this from a Django shell to force a synchronization of all user groups for a specific service.

service_active_for_user

```
def service_active_for_user(self, user):
```

Is this service active for the given user? The `user` parameter should be a Django User object.

Usually you wont need to override this as it calls `service_enabled_members` or `service_enabled_blues` depending on the users state.

show_service_ctrl

```
def show_service_ctrl(self, user, state):
```

Should the service be shown for the given user with the given state? The `user` parameter should be a Django User object, and the `state` parameter should be a valid state from `authentication.states`.

Usually you wont need to override this function.

For more information see the [render_service_ctrl](#) section.

render_service_ctrl

```
def render_services_ctrl(self, request):
```

Render the services control row. This will be called for all active services when a user visits the `/services/` page and [show_service_ctrl](#) returns `True` for the given user.

It should return a string (usually from `render_to_string`) of a table row (`<tr>`) with 4 columns (`<td>`). Column #1 is the service name, column #2 is the users username for this service, column #3 is the services URL, and column #4 is the action buttons.

You may either define your own service template or use the default one provided. The default can be used like this example:

```
def render_services_ctrl(self, request):
    """
    Example for rendering the service control panel row
    You can override the default template and create a
    custom one if you wish.
    :param request:
    :return:
    """
    urls = self.Urns()
    urls.auth_activate = 'auth_example_activate'
    urls.auth_deactivate = 'auth_example_deactivate'
    urls.auth_reset_password = 'auth_example_reset_password'
    urls.auth_set_password = 'auth_example_set_password'
    return render_to_string(self.service_ctrl_template, {
        'service_name': self.title,
        'urls': urls,
        'service_url': self.service_url,
        'username': 'example username'
    }, request=request)
```

the `Urns` class defines the available URL names for the 4 actions available in the default template:

- Activate (create service account)

- Deactivate (delete service account)
- Reset Password (random password)
- Set Password (custom password)

If you don't define one or all of these variable the button for the undefined URLs will not be displayed.

Most services will survive with the default template. If, however, you require extra buttons for whatever reason, you are free to provide your own template as long as you stick within the 4 columns. Multiple rows should be OK, though may be confusing to users.

Menu Item Hook

If your services needs cannot be satisfied by the Service Control row, you are free to specify extra hooks by subclassing or instantiating the `services.hooks.MenuItemHook` class.

For more information see the [Menu Hooks](#) page.

The Service Manager

The service manager is what interacts with the external service. Ideally it should be completely agnostic about its environment, meaning that it should avoid calls to Alliance Auth and Django in general (except in special circumstances where the service is managed locally, e.g. Mumble). Data should come in already arranged by the Tasks and data passed back for the tasks to manage or distribute.

The reason for maintaining this separation is that managers may be reused from other sources and there may not even be a need to write a custom manager. Likewise, by maintaining this neutral environment others may reuse the managers that we write. It can also significantly ease the unit testing of services.

The Views

As mentioned at the start of this page, service modules are fully fledged Django apps. This means you're free to do whatever you wish with your views.

Typically most traditional username/password services define four views.

- Create Account
- Delete Account
- Reset Password
- Set Password

These views should interact with the service via the Tasks, though in some instances may bypass the Tasks and access the manager directly where necessary, for example OAuth functionality.

The Tasks

The tasks component is the glue that holds all of the other components of the service module together. It provides the function implementation to handle things like adding and deleting users, updating groups, validating the existence of a users account. Whatever tasks `auth_hooks` and `views` have with interacting with the service will probably live here.

The Models

Its very likely that you'll need to store data about a users remote service account locally. As service modules are fully fledged Django apps you are free to create as many models as necessary for persistent storage. You can create foreign keys to other models in Alliance Auth if necessary, though I *strongly* recommend you limit this to the User and Groups models from `django.contrib.auth.models` and query any other data manually.

If you create models you should create the migrations that go along with these inside of your module/app.

Examples

There is a bare bones example service included in `services.modules.example`, you may like to use this as the base for your new service.

You should have a look through some of the other service modules before you get started to get an idea of the general structure of one. A lot of them aren't perfect so don't feel like you have to rigidly follow the structure of the existing services if you think its sub-optimal or doesn't suit the external service you're integrating.

Testing

You will need to add unit tests for all aspects of your service module before it is accepted. Be mindful that you don't actually want to make external calls to the service so you should mock the appropriate components to prevent this behavior.

6.1.2 Menu Hooks

The menu hooks allow you to dynamically specify menu items from your plugin app or service. To achieve this you should subclass or instantiate the `services.hooks.MenuItemHook` class and then register the menu item with one of the hooks.

To register a `MenuItemHook` class you would do the following:

```
@hooks.register('menu_item_hook')
def register_menu():
    return MenuItemHook('Example Item', 'glyphicon glyphicon-heart', 'example_url_name
↪', 150)
```

The `MenuItemHook` class specifies some parameters/instance variables required for menu item display.

MenuItemHook(text, classes, url_name, order=None)**text**

The text shown as menu item, e.g. usually the name of the app.

classes

The classes that should be applied to the bootstrap menu item icon

url_name

The name of the Django URL to use

order

An integer which specifies the order of the menu item, lowest to highest. Community apps are free to use an order above 1000. Numbers below are served for Auth.

navactive

A list of views or namespaces the link should be highlighted on. See [django-navhelper](#) for usage. Defaults to the supplied `url_name`.

count

`count` is an integer shown next to the menu item as badge when `count` is not `None`.

This is a great feature to signal the user, that he has some open issues to take care of within an app. For example Auth uses this feature to show the specific number of open group request to the current user.

Hint: Here is how to stay consistent with the Auth design philosophy for using this feature: 1. Use it to display open items that the current user can close by himself only. Do not use it for items, that the user has no control over. 2. If there are currently no open items, do not show a badge at all.

To use it set `count` the `render()` function of your subclass in accordance to the current user. Here is an example:

```
def render(self, request):  
    # ...  
    self.count = calculate_count_for_user(request.user)  
    # ...
```

Customization

If you cannot get the menu item to look the way you wish, you are free to subclass and override the default render function and the template used.

6.1.3 URL Hooks

Note: URLs added through URL Hooks are protected by a decorator which ensures the requesting user is logged in and has a main character set.

The URL hooks allow you to dynamically specify URL patterns from your plugin app or service. To achieve this you should subclass or instantiate the `services.hooks.UrlHook` class and then register the URL patterns with the hook.

To register a `UrlHook` class you would do the following:

```
@hooks.register('url_hook')
def register_urls():
    return UrlHook(app_name.urls, 'app_name', r'^app_name/')
```

The `UrlHook` class specifies some parameters/instance variables required for URL pattern inclusion.

`UrlHook(urls, app_name, base_url)`

urls

The `urls` module to include. See [the Django docs](#) for designing urlpatterns.

namespace

The URL namespace to apply. This is usually just the app name.

base_url

The URL prefix to match against in regex form. Example `r'^app_name/'`. This prefix will be applied in front of all URL patterns included. It is possible to use the same prefix as existing apps (or no prefix at all) but [standard URL resolution](#) ordering applies (hook URLs are the last ones registered).

Example

An app called `plugin` provides a single view:

```
def index(request):
    return render(request, 'plugin/index.html')
```

The app's `urls.py` would look like so:


```
from django.conf.urls import url
import plugin.views

urlpatterns = [
    url(r'^index$', plugins.views.index, name='index'),
]
```

Subsequently it would implement the `UrlHook` in a dedicated `auth_hooks.py` file like so:

```
from alliance_auth import hooks
from services.hooks import UrlHook
import plugin.urls

@hooks.register('url_hook')
def register_urls():
    return UrlHook(plugin.urls, 'plugin', r'^plugin/')
```

When this app is included in the project's settings, `INSTALLED_APPS` users would access the index view by navigating to `https://example.com/plugin/index`.

6.1.4 Logging from Custom Apps

Alliance Auth provides a logger for use with custom apps to make everyone's life a little easier.

Using the Extensions Logger

AllianceAuth provides a helper function to get the logger for the current module to reduce the amount of code you need to write.

```
from allianceauth.services.hooks import get_extension_logger

logger = get_extension_logger(__name__)
```

This works by creating a child logger of the extension logger which propagates all log entries to the parent (extensions) logger.

Changing the Logging Level

By default, the extension logger's level is set to `DEBUG`. To change this, uncomment (or add) the following line in `local.py`.

```
LOGGING['handlers']['extension_file']['level'] = 'INFO'
```

(Remember to restart your supervisor workers after changes to `local.py`)

This will change the logger's level to the level you define.

Options are: *(all options accept entries of levels listed below them)*

- `DEBUG`
- `INFO`
- `WARNING`
- `ERROR`

- CRITICAL

6.2 Developing AA Core

This section contains important information on how to develop Alliance Auth itself.

6.2.1 Alliance Auth documentation

The documentation for Alliance Auth uses [Sphinx](#) to build documentation. When a new commit to specific branches is made (master, primarily), the repository is automatically pulled, docs built and deployed on [readthedocs.org](#).

Documentation was migrated from the GitHub wiki pages and into the repository to allow documentation changes to be included with pull requests. This means that documentation can be guaranteed to be updated when a pull request is accepted rather than hoping documentation is updated afterwards or relying on maintainers to do the work. It also allows for documentation to be maintained at different versions more easily.

Building Documentation

If you're developing new documentation, its likely you'll want or need to test build it before committing to your branch. To achieve this you can use Sphinx to build the documentation locally as it appears on Read the Docs.

Activate your virtual environment (if you're using one) and install the documentation requirements found in `docs/requirements.txt` using `pip`, e.g. `pip install -r docs/requirements.txt`.

You can then build the docs by changing to the `docs/` directory and running `make html` or `make dirhtml`, depending on how the Read the Docs project is configured. Either should work fine for testing. You can now find the output of the build in the `/docs/_build/` directory.

Occasionally you may need to fully rebuild the documents by running `make clean` first, usually when you add or rearrange toctrees.

Documentation Format

CommonMark Markdown is the current preferred format, via [recommonmark](#). reStructuredText is supported if required, or you can execute snippets of reST inside Markdown by using a code block:

```
```eval_rst
reStructuredText here
```
```

Markdown is used elsewhere on Github so it provides the most portability of documentation from Issues and Pull Requests as well as providing an easier initial migration path from the Github wiki.

6.3 Setup dev environment for AA

Here you find guides on how to setup your development environment for AA.

6.3.1 Development on Windows 10 with WSL and Visual Studio Code

This document describes step-by-step how to setup a complete development environment for Alliance Auth apps on Windows 10 with Windows Subsystem for Linux (WSL) and Visual Studio Code.

The main benefit of this setup is that it runs all services and code in the native Linux environment (WSL) and at the same time can be full controlled from within a comfortable Windows IDE (Visual Studio Code) including code debugging.

In addition all tools described in this guide are open source or free software.

Hint: This guide is meant for development purposes only and not for installing AA in a production environment. For production installation please see chapter **Installation**.

Overview

The development environment consists of the following components:

- Visual Studio Code with Remote WSL and Python extension
- WSL with Ubuntu 18.04. LTS
- Python 3.7 environment on WSL
- MySQL server on WSL
- Redis on WSL
- Alliance Auth on WSL
- Celery on WSL

We will use the build-in Django development webserver, so we don't need to setup a WSGI server or a web server.

Note: This setup works with both WSL 1 and WSL 2. However, due to the significantly better performance we recommend WSL 2.

Requirement

The only requirement is a PC with Windows 10 and Internet connection in order to download the additional software components.

Windows installation

Windows Subsystem for Linux

- Install from here: [Microsoft docs](#)
- Choose Ubuntu 18.04. LTS

Visual Studio Code

- Install from here: [VSC Download](#)
- Open the app and install the following VSC extensions:
- Remote WSL
- Connect to WSL. This will automatically install the VSC server on the VSC server for WSL
- Once connected to WSL install the Python extension on the WSL side

WSL Installation

Open a WSL bash and update all software packets:

```
sudo apt update && sudo apt upgrade -y
```

Install Tools

```
sudo apt-get install build-essential  
sudo apt-get install gettext
```

Install Python

For AA we want to develop with Python 3.7, because that provides the maximum compatibility with today's AA installations.

Hint: To check your system's Python 3 version you can enter: `python3 --version`

Note: Should your Ubuntu come with a newer version of Python we recommend to still setup your dev environment with the oldest Python 3 version supported by AA, e.g Python 3.7 You can check out this [page](#) on how to install additional Python versions on Ubuntu.

Use the following command to install Python 3 with all required libraries with the default version:

```
sudo apt-get install python3 python3-dev python3-venv python3-setuptools python3-pip  
↪python-pip
```

Installing the DBMS

Install MySQL and required libraries with the following command:

```
sudo apt-get install mysql-server mysql-client libmysqlclient-dev
```

Note: We chose to use MySQL instead of MariaDB, because the standard version of MariaDB that comes with this Ubuntu distribution will not work with AA.

We need to apply a permission fix to mysql or you will get a warning with every startup:

```
sudo usermod -d /var/lib/mysql/ mysql
```

Start the mysql server

```
sudo service mysql start
```

Create database and user for AA

```
sudo mysql -u root
```

```
CREATE USER 'aa_dev'@'localhost' IDENTIFIED BY 'PASSWORD';
CREATE DATABASE aa_dev CHARACTER SET utf8mb4;
GRANT ALL PRIVILEGES ON aa_dev . * TO 'aa_dev'@'localhost';
CREATE DATABASE test_aa_dev CHARACTER SET utf8mb4;
GRANT ALL PRIVILEGES ON test_aa_dev . * TO 'aa_dev'@'localhost';
exit;
```

Add timezone info to mysql

```
sudo mysql_tzinfo_to_sql /usr/share/zoneinfo | sudo mysql -u root mysql
```

Install redis and other tools

```
sudo apt-get install unzip git redis-server curl libssl-dev libbz2-dev libffi-dev
```

Start redis

```
sudo redis-server --daemonize yes
```

Note: WSL does not have an init.d service, so it will not automatically start your services such as MySQL and Redis when you boot your Windows machine. For convenience we recommend putting the commands for starting these services in a bash script. Here is an example:

```
#!/bin/bash
# start services for AA dev
sudo service mysql start
sudo redis-server --daemonize yes
```

In addition it is possible to configure Windows to automatically start WSL services, but that procedure goes beyond the scopes of this guide.

Setup dev folder on WSL

Setup your folders on WSL bash for your dev project. Our approach will setup one AA project with one venv and multiple apps running under the same AA project, but each in their own folder and git.

A good location for setting up this folder structure is your home folder or a subfolder of your home:

```
~/aa-dev
|- venv
|- myauth
|- my_app_1
|- my_app_2
|- ...
```

Following this approach you can also setup additional AA projects, e.g. aa-dev-2, aa-dev-3 if needed.

Create the root folder aa-dev.

setup virtual Python environment for aa-dev

Create the virtual environment. Run this in your aa-dev folder:

```
python3 -m venv venv
```

And activate your venv:

```
source venv/bin/activate
```

install Python packages

```
pip install --upgrade pip
pip install wheel
```

Alliance Auth installation

Install and create AA instance

```
pip install allianceauth
```

Now we are ready to setup our AA instance. Make sure to run this command in your aa-dev folder:

```
allianceauth start myauth
```

Next we will setup our VSC project for aa-dev by starting it directly from the WSL bash:

```
code .
```

First you want to make sure exclude the venv folder from VSC as follows: Open settings and go to Files:Exclude Add pattern: `**/venv`

Update settings

Open the settings file with VSC. Its under myauth/myauth/settings/local.py

Make sure to have the settings of your Eve Online app ready.

Turn on DEBUG mode to ensure your static files get served by Django:

```
DEBUG = True
```

Update name, user and password of your DATABASE configuration.

```
DATABASES['default'] = {
    'ENGINE': 'django.db.backends.mysql',
    'NAME': 'aa_dev',
    'USER': 'aa_dev',
    'PASSWORD': 'PASSWORD',
    'HOST': '127.0.0.1',
    'PORT': '3306',
    'OPTIONS': {'charset': 'utf8mb4'},
}
```

For the Eve Online related setup you need to create a SSO app on the developer site:

- Create your Eve Online SSO App on the [Eve Online developer site](#)
- Add all ESI scopes
- Set callback URL to: `http://localhost:8000/sso/callback`

Then update local.py with your settings:

```
ESI_SSO_CLIENT_ID = 'YOUR-ID'
ESI_SSO_CLIENT_SECRET = 'YOUR_SECRET'
ESI_SSO_CALLBACK_URL = 'http://localhost:8000/sso/callback'
```

Disable email registration:

```
REGISTRATION_VERIFY_EMAIL = False
```

Migrations and superuser

Before we can start AA we need to run migrations:

```
cd myauth
python manage.py migrate
```

We also need to create a superuser for our AA installation:

```
python /home/allianceserver/myauth/manage.py createsuperuser
```

Running Alliance Auth

AA instance

We are now ready to run out AA instance with the following command:

```
python manage.py runserver
```

Once running you can access your auth site on the browser under `http://localhost:8000`. Or the admin site under `http://localhost:8000/admin`

Hint: You can start your AA server directly from a terminal window in VSC or with a VSC debug config (see chapter about debugging for details).

Note: Debug vs. Non-Debug mode Usually it is best to run your dev AA instance in debug mode, so you get all the detailed error messages that helps a lot for finding errors. But there might be cases where you want to test features that do not exist in debug mode (e.g. error pages) or just want to see how your app behaves in non-debug / production mode.

When you turn off debug mode you will see a problem though: Your pages will not render correctly. The reason is that Django will stop serving your static files in production mode and expect you to serve them from a real web server. Luckily, there is an option that forces Django to continue serving your static files directly even when not in debug mode. Just start your server with the following option: `python manage.py runserver --insecure`

Celery

In addition you can start a celery worker instance for myauth. For development purposed it makes sense to only start one instance and add some additional logging.

This can be done from the command line with the following command in the myauth folder (where `manage.py` is located):

```
celery -E -A myauth worker -l info -P solo
```

Same as AA itself you can start Celery from any terminal session, from a terminal window within VSC or as a debug config in VSC (see chapter about debugging for details). For convenience we recommend starting Celery as debug config.

Debugging setup

To be able to debug your code you need to add debugging configuration to VSC. At least one for AA and one for celery.

Breakpoints

By default VSC will break on any uncaught exception. Since every error raised by your tests will cause an uncaught exception we recommend to deactivate this feature.

To deactivate open click on the debug icon to switch to the debug view. Then un-check “Uncaught Exceptions” on the breakpoints window.

AA debug config

In VSC click on Debug / Add Configuration and choose “Django”. Should Django not appear as option make sure to first open a Django file (e.g. the local.py settings) to help VSC detect that you are using Django.

The result should look something like this:

```
{
  "name": "Python: Django",
  "type": "python",
  "request": "launch",
  "program": "${workspaceFolder}/myauth/manage.py",
  "args": [
    "runserver",
    "--noreload"
  ],
  "django": true
}
```

Debug celery

For celery we need another debug config, so that we can run it in parallel to our AA instance.

Here is an example debug config for Celery:

```
{
  "name": "Python: Celery",
  "type": "python",
  "request": "launch",
  "module": "celery",
  "cwd": "${workspaceFolder}/myauth",
  "console": "integratedTerminal",
  "args": [
    "-A",
    "myauth",
    "worker",
    "-l",
    "info",
    "-P",
    "solo",
  ],
  "django": true,
  "justMyCode": true,
},
```

Debug config for unit tests

Finally it makes sense to have a dedicated debug config for running unit tests. Here is an example config for running all tests of the app `example`.

```
{
  "name": "Python: myauth unit tests",
  "type": "python",
  "request": "launch",
  "program": "${workspaceFolder}/myauth/manage.py",
  "args": [
    "test",
    "--keepdb",
    "--debug-mode",
    "--failfast",
    "example",
  ],
  "django": true,
  "justMyCode": true
},
```

You can also specify to run just a part of your test suite down to a test method. Just give the full path to the test you want to run, e.g. `example.test.test_models.TestDemoModel.test_this_method`

Debugging normal python scripts

Finally you may also want to have a debug config to debug a non-Django Python script:

```
{
  "name": "Python: Current File",
  "type": "python",
  "request": "launch",
  "program": "${file}",
  "console": "integratedTerminal"
},
```

Additional tools

The following additional tools are very helpful when developing for AA with VS Code:

Pylance

Pylance is an extension that works alongside Python in Visual Studio Code to provide performant language support: [Pylance](#)

Code Spell Checker

Typos in your user facing comments can be quite embarrassing. This spell checker helps you avoid them: [Code Spell Checker](#)

markdownlint

Extension for Visual Studio Code - Markdown linting and style checking for Visual Studio Code: [markdownlint](#)

GitLens

Extension for Visual Studio Code - Supercharge the Git capabilities built into Visual Studio Code: [GitLens](#)

RST preview

A VS Code extension to preview restructured text and provide syntax highlighting: [RST Preview](#)

Django Template

This extension adds language colorization support and user snippets for the Django template language to VS Code: [Django Template](#)

DBeaver

DBeaver is a free universal database tool and works with many different kinds of databases include MySQL. It can be installed on Windows 10 and will be able to help manage your MySQL databases running on WSL.

Install from here. [DBeaver](#)

django-extensions

[django-extensions](#) is a swiss army knife for django developers with adds a lot of very useful features to your Django site. Here are a few highlights:

- `shell_plus` - An enhanced version of the Django shell. It will auto-load all your models at startup so you don't have to import anything and can use them right away.
- `graph_models` - Creates a dependency graph of Django models. Visualizing a model dependency structure can be very useful for trying to understand how an existing Django app works, or e.g. how all the AA models work together.
- `runserver_plus` - The standard runserver stuff but with the Werkzeug debugger baked in. This is a must have for any serious debugging.

Adding apps for development

The idea behind the particular folder structure of aa-dev is to have each and every app in its own folder and git repo. To integrate them with the AA instance they need to be installed once using the `-e` option that enabled editing of the package. And then added to the `INSTALLED_APPS` settings.

To demonstrate let's add the example plugin to our environment.

Open a WSL bash and navigate to the aa-dev folder. Make sure you have activate your virtual environment. (`source venv/bin/activate`)

Run these commands:

```
git clone https://gitlab.com/ErikKalkoken/allianceauth-example-plugin.git
pip install -e allianceauth-example-plugin
```

Add `'example'` to `INSTALLED_APPS` in your `local.py` settings.

Run migrations and restart your AA server, e.g.:

```
cd myauth
python manage.py migrate
```

6.4 Developing apps

In this section you find topics useful for app developers.

6.4.1 API

To reduce redundancy and help speed up development we encourage developers to utilize the following packages when developing apps for Alliance Auth.

Discord Client

AA contains a web client for interacting with the Discord API. This client can be used independently from an installed Discord service in AA.

Location: `allianceauth.services.modules.discord.discord_client`

- *client*
- *models*
- *exceptions*
- *settings*

client

Client for interacting with the Discord API.

class DiscordApiStatusCode (*value*)
Status code returned from the Discord API.

UNKNOWN_MEMBER = 10007

class DiscordClient (*access_token: str, redis: Optional[redis.client.Redis] = None, is_rate_limited: bool = True*)

This class provides a web client for interacting with the Discord API.

The client has rate limiting that supports concurrency. This means it is able to ensure the API rate limit is not violated, even when used concurrently, e.g. with multiple parallel celery tasks.

In addition the client support proper API backoff.

Synchronization of rate limit infos across multiple processes is implemented with Redis and thus requires Redis as Django cache backend.

The cache is shared across all clients and processes (also using Redis).

All durations are in milliseconds.

Most errors from the API will raise a `requests.HTTPError`.

Parameters

- **access_token** – Discord access token used to authenticate all calls to the API
- **redis** – Redis instance to be used.
- **is_rate_limited** – Set to False to turn off rate limiting (use with care). If not specified will try to use the Redis instance from the default Django cache backend.

Raises **ValueError** – No access token provided

__init__ (*access_token: str, redis: Optional[redis.client.Redis] = None, is_rate_limited: bool = True*)
→ **None**
Initialize self. See help(type(self)) for accurate signature.

property access_token
Discord access token.

add_guild_member (*guild_id: int, user_id: int, access_token: str, role_ids: Optional[list] = None, nick: Optional[str] = None*) → **Optional[bool]**
Adds a user to the guild.

Returns

- True when a new user was added
- None if the user already existed
- False when something went wrong or raises exception

add_guild_member_role (*guild_id: int, user_id: int, role_id: int*) → **Optional[bool]**
Adds a role to a guild member

Returns: - True when successful - None if member does not exist - False otherwise

create_guild_role (*guild_id: int, role_name: str, **kwargs*) → **Optional[allianceauth.services.modules.discord.discord_client.models.Role]**
Create a new guild role with the given name.

See official documentation for additional optional parameters.

Note that Discord allows the creation of multiple roles with the same name, so to avoid duplicates it's important to check existing roles before creating new one

Parameters

- **guild_id** – Discord ID of the guild
- **role_name** – Name of new role to create

Returns new role on success

current_user () → *allianceauth.services.modules.discord.discord_client.models.User*
Fetch user belonging to the current access_token.

delete_guild_role (guild_id: *int*, role_id: *int*) → *bool*
Delete a guild role.

guild_infos (guild_id: *int*) → *allianceauth.services.modules.discord.discord_client.models.Guild*
Fetch all basic infos about this guild.

Parameters **guild_id** – Discord ID of the guild

guild_member (guild_id: *int*, user_id: *int*) → *Optional[allianceauth.services.modules.discord.discord_client.models.GuildMember]*
Fetch info for a guild member.

Parameters

- **guild_id** – Discord ID of the guild
- **user_id** – Discord ID of the user

Returns guild member or *None* if the user is not a member of the guild

guild_member_roles (guild_id: *int*, user_id: *int*) → *Optional[allianceauth.services.modules.discord.discord_client.helpers.RolesSet]*
Fetch the current guild roles of a guild member.

Args: - guild_id: Discord guild ID - user_id: Discord user ID

Returns: - Member roles - *None* if user is not a member of the guild

guild_name (guild_id: *int*, use_cache: *bool* = *True*) → *str*
Fetch the name of this guild (cached).

Parameters

- **guild_id** – Discord ID of the guild
- **use_cache** – When set to *False* will force an API call to get the server name

Returns Name of the server or an empty string if something went wrong.

guild_roles (guild_id: *int*, use_cache: *bool* = *True*) → *Set[allianceauth.services.modules.discord.discord_client.models.Role]*
Fetch all roles for this guild.

Parameters

- **guild_id** – Discord ID of the guild
- **use_cache** – If is set to *False* it will always hit the API to retrieve fresh data and update the cache.

Returns:

property is_rate_limited
Wether this instance is rate limited.

match_or_create_role_from_name (*guild_id: int, role_name: str, guild_roles: Optional[allianceauth.services.modules.discord.discord_client.helpers.RolesSet] = None*) → *Tuple[allianceauth.services.modules.discord.discord_client.models.Role, bool]*

Fetch or create Discord role matching the given name.

Will try to match with existing roles names Non-existing roles will be created, then created flag will be True

Parameters

- **guild_id** – ID of guild
- **role_name** – strings defining name of a role
- **guild_roles** – All known guild roles as RolesSet object. Helps to void redundant lookups of guild roles when this method is used multiple times.

Returns Tuple of Role and created flag

match_or_create_roles_from_names (*guild_id: int, role_names: Iterable[str]*) → *List[Tuple[allianceauth.services.modules.discord.discord_client.models.Role, bool]]*

Fetch or create Discord roles matching the given names (cached).

Will try to match with existing roles names Non-existing roles will be created, then created flag will be True

Parameters

- **guild_id** – ID of guild
- **role_names** – list of name strings each defining a role

Returns List of tuple of Role and created flag

match_or_create_roles_from_names_2 (*guild_id: int, role_names: Iterable[str]*) → *allianceauth.services.modules.discord.discord_client.helpers.RolesSet*

Fetch or create Discord role matching the given name.

Wrapper for `match_or_create_role_from_name()`

Returns Roles as RolesSet object.

match_role_from_name (*guild_id: int, role_name: str*) → *Optional[allianceauth.services.modules.discord.discord_client.models.Role]*

Fetch Discord role matching the given name (cached).

Parameters

- **guild_id** – Discord ID of the guild
- **role_name** – Name of role

Returns Matching role or None if no match is found

modify_guild_member (*guild_id: int, user_id: int, role_ids: Optional[List[int]] = None, nick: Optional[str] = None*) → *Optional[bool]*

Set properties of a guild member.

Parameters

- **guild_id** – Discord ID of the guild
- **user_id** – Discord ID of the user
- **roles_id** – New list of role IDs (if provided)

- **nick** – New nickname (if provided)

Returns

- True when successful
- None if user is not a member of this guild
- False otherwise

remove_guild_member (*guild_id: int, user_id: int*) → Optional[bool]

Remove a member from a guild.

Parameters

- **guild_id** – Discord ID of the guild
- **user_id** – Discord ID of the user

Returns

- True when successful
- None if member does not exist
- False otherwise

remove_guild_member_role (*guild_id: int, user_id: int, role_id: int*) → Optional[bool]

Remove a role to a guild member

Parameters

- **guild_id** – Discord ID of the guild
- **user_id** – Discord ID of the user
- **role_id** – Discord ID of role to be removed

Returns

- True when successful
- None if member does not exist
- False otherwise

models

Implementation of Discord objects used by this client.

Note that only those objects and properties are implemented, which are needed by AA.

Names and types are mirrored from the API whenever possible. Discord's snowflake type (used by Discord IDs) is implemented as int.

class User (*id: int, username: str, discriminator: str*)

A user on Discord.

id: int

username: str

discriminator: str

classmethod from_dict (*data: dict*) → *allianceauth.services.modules.discord.discord_client.models.User*

Create object from dictionary as received from the API.


```

__init__ (id: int, username: str, discriminator: str) → None
    Initialize self. See help(type(self)) for accurate signature.

class Role (id: int, name: str, managed: bool = False)
    A role on Discord.

    id: int

    name: str

    managed: bool = False

    asdict () → dict
        Convert object into a dictionary representation.

    classmethod from_dict (data: dict) → allianceauth.services.modules.discord.discord_client.models.Role
        Create object from dictionary as received from the API.

    classmethod sanitize_name (role_name: str) → str
        Shorten too long names if necessary.

__init__ (id: int, name: str, managed: bool = False) → None
    Initialize self. See help(type(self)) for accurate signature.

class Guild (id: int, name: str, roles: FrozenSet[allianceauth.services.modules.discord.discord_client.models.Role])
    A guild on Discord.

    id: int

    name: str

    roles: FrozenSet[allianceauth.services.modules.discord.discord_client.models.Role]

    classmethod from_dict (data: dict) → allianceauth.services.modules.discord.discord_client.models.Guild
        Create object from dictionary as received from the API.

__init__ (id: int, name: str, roles: FrozenSet[allianceauth.services.modules.discord.discord_client.models.Role])
    → None
    Initialize self. See help(type(self)) for accurate signature.

class GuildMember (roles: FrozenSet[int], nick: Optional[str] = None, user: Optional[allianceauth.services.modules.discord.discord_client.models.User] = None)
    A member of a guild on Discord.

    roles: FrozenSet[int]

    nick: str = None

    user: allianceauth.services.modules.discord.discord_client.models.User = None

    classmethod from_dict (data: dict) → allianceauth.services.modules.discord.discord_client.models.GuildMember
        Create object from dictionary as received from the API.

    classmethod sanitize_nick (nick: str) → str
        Sanitize a nick, i.e. shorten too long strings if necessary.

__init__ (roles: FrozenSet[int], nick: Optional[str] = None, user: Optional[allianceauth.services.modules.discord.discord_client.models.User] = None) →
    None
    Initialize self. See help(type(self)) for accurate signature.

```

exceptions

Custom exceptions for the Discord Client package.

exception `DiscordApiBackoff` (*retry_after: int*)

Exception signaling we need to backoff from sending requests to the API for now.

Parameters `retry_after` – time to retry after in milliseconds

`__init__` (*retry_after: int*)

Initialize self. See `help(type(self))` for accurate signature.

property `retry_after_seconds`

Time to retry after in seconds.

exception `DiscordClientException`

Base Exception for the Discord client.

exception `DiscordRateLimitExhausted` (*retry_after: int*)

Exception signaling that the total number of requests allowed under the current rate limit have been exhausted and need to wait until next reset.

exception `DiscordTooManyRequestsError` (*retry_after: int*)

API has responded with a 429 Too Many Requests Error. Need to backoff for now.

settings

Settings for the Discord client.

To overwrite a default set the variable in your local Django settings, e.g:

```
DISCORD_GUILD_NAME_CACHE_MAX_AGE = 7200
```

`DISCORD_API_BASE_URL = 'https://discord.com/api/'`

Base URL for all API calls. Must end with /.

`DISCORD_API_TIMEOUT_CONNECT = 5`

Low level connect timeout for requests to the Discord API in seconds.

`DISCORD_API_TIMEOUT_READ = 30`

Low level read timeout for requests to the Discord API in seconds.

`DISCORD_DISABLE_ROLE_CREATION = False`

Turns off creation of new roles. In case the rate limit for creating roles is exhausted, this setting allows the Discord service to continue to function and wait out the reset. Rate limit is about 250 per 48 hrs.

`DISCORD_GUILD_NAME_CACHE_MAX_AGE = 86400`

How long the Discord guild names retrieved from the server are caches locally in seconds.

`DISCORD_OAUTH_BASE_URL = 'https://discord.com/api/oauth2/authorize'`

Base authorization URL for Discord OAuth.

`DISCORD_OAUTH_TOKEN_URL = 'https://discord.com/api/oauth2/token'`

Base authorization URL for Discord OAuth.

`DISCORD_ROLES_CACHE_MAX_AGE = 3600`

How long Discord roles retrieved from the server are caches locally in seconds.

Discord Service

This page contains the technical documentation for the Discord service.

Location: `allianceauth.services.modules.discord`

- *api*
- *settings*

api

Public interface for community apps who want to interact with the Discord server of the current Alliance Auth instance.

Example

Here is an example for using the api to fetch the current roles from the configured Discord server.

```
from allianceauth.services.modules.discord.api import create_bot_client, discord_
    ↪ guild_id

client = create_bot_client() # create a new Discord client
guild_id = discord_guild_id() # get the ID of the configured Discord server
roles = client.guild_roles(guild_id) # fetch the roles from our Discord server
```

See also:

The docs for the client class can be found here: *DiscordClient*

class `DiscordUser` (*args, **kwargs)

The Discord user account of an Auth user.

Parameters

- **uid** (*BigIntegerField*) – Uid. user’s ID on Discord
- **username** (*CharField*) – Username. user’s username on Discord
- **discriminator** (*CharField*) – Discriminator. user’s discriminator on Discord
- **activated** (*DateTimeField*) – Activated. Date & time this service account was activated

Relationship fields:

Parameters **user** (*OneToOneField* to *User*) – Primary key: User. Auth user owning this Discord account (related name: discord)

class `Role` (*id: int, name: str, managed: bool = False*)

A role on Discord.

__init__ (*id: int, name: str, managed: bool = False*) → *None*

Initialize self. See help(type(self)) for accurate signature.

asdict () → *dict*

Convert object into a dictionary representation.

classmethod **from_dict** (*data: dict*) → *allianceauth.services.modules.discord.discord_client.models.Role*

Create object from dictionary as received from the API.

classmethod `sanitize_name` (*role_name: str*) → *str*

Shorten too long names if necessary.

create_bot_client (*is_rate_limited: bool = True*) → *allianceauth.services.modules.discord.discord_client.client.DiscordClient*

Create new bot client for accessing the configured Discord server.

Parameters `is_rate_limited` – Set to False to turn off rate limiting (use with care).

Returns Discord client instance

group_to_role (*group: [django.contrib.auth.models.Group](#)*) → *Optional[[allianceauth.services.modules.discord.discord_client.models.Role](#)]* Op-

Fetch the Discord role matching the given Django group by name.

Returns Discord role or None if no matching role exist

server_name (*use_cache: bool = True*) → *str*

Fetches the name of the current Discord server.

Parameters `use_cache` – When set False will force an API call to get the server name

Returns Server name or an empty string if the name could not be retrieved

settings

DISCORD_APP_ID = `'appid'`

App ID for the AA bot on Discord. Needs to be set.

DISCORD_APP_SECRET = `'secret'`

App secret for the AA bot on Discord. Needs to be set.

DISCORD_BOT_TOKEN = `'bottoken'`

Token used by the AA bot on Discord. Needs to be set.

DISCORD_CALLBACK_URL = `'http://example.com/discord/callback'`

Callback URL for OAuth with Discord. Needs to be set.

DISCORD_GUILD_ID = `'0118999'`

ID of the Discord Server. Needs to be set.

DISCORD_SYNC_NAMES = `False`

Automatically sync Discord users names to user's main character name when created.

DISCORD_TASKS_MAX_RETRIES = `3`

Max retries of tasks after an error occurred.

DISCORD_TASKS_RETRY_PAUSE = `60`

Pause in seconds until next retry for tasks after the API returned an error.

django-esi

The django-esi package provides an interface for easy access to the ESI.

This is an external package. Please see [here](#) for it's documentation.

evelinks

This package generates profile URLs for eve entities on 3rd party websites like evewho and zKillboard.

Location: `allianceauth.eveonline.evelinks`

eveimageserver

alliance_logo_url (*alliance_id: int, size: int = 32*) → str
image URL for the given alliance ID

character_portrait_url (*character_id: int, size: int = 32*) → str
image URL for the given character ID

corporation_logo_url (*corporation_id: int, size: int = 32*) → str
image URL for the given corporation ID

type_icon_url (*type_id: int, size: int = 32*) → str
icon image URL for the given type ID

type_render_url (*type_id: int, size: int = 32*) → str
render image URL for the given type ID

dotlan

alliance_url (*name: str*) → str
url for page about given alliance on dotlan

corporation_url (*name: str*) → str
url for page about given corporation on dotlan

region_url (*name: str*) → str
url for page about given region on dotlan

solar_system_url (*name: str*) → str
url for page about given solar system on dotlan

eveho

alliance_url (*eve_id: int*) → str
url for page about given alliance on evewho

character_url (*eve_id: int*) → str
url for page about given character on evewho

corporation_url (*eve_id: int*) → str
url for page about given corporation on evewho

zkillboard

alliance_url (*eve_id: int*) → str
url for page about given alliance on zKillboard

character_url (*eve_id: int*) → str
url for page about given character on zKillboard

corporation_url (*eve_id: int*) → str
url for page about given corporation on zKillboard

region_url (*eve_id: int*) → str
url for page about given region on zKillboard

solar_system_url (*eve_id: int*) → str

eveonline

The eveonline package provides models for commonly used Eve Online entities like characters, corporations and alliances. All models have the ability to be loaded from ESI.

Location: `allianceauth.eveonline`

models

class EveAllianceInfo (*id, alliance_id, alliance_name, alliance_ticker, executor_corp_id*)

Parameters

- **id** (*AutoField*) – Primary key: ID
- **alliance_id** (*PositiveIntegerField*) – Alliance id
- **alliance_name** (*CharField*) – Alliance name
- **alliance_ticker** (*CharField*) – Alliance ticker
- **executor_corp_id** (*PositiveIntegerField*) – Executor corp id

Reverse relationships:

Parameters

- **state** (Reverse *ManyToManyField* from *State*) – All states of this eve alliance info (related name of `member_alliances`)
- **evecorporationinfo** (Reverse *ForeignKey* from *EveCorporationInfo*) – All eve corporation infos of this eve alliance info (related name of `alliance`)
- **managedalliancegroup** (Reverse *ForeignKey* from *ManagedAllianceGroup*) – All managed alliance groups of this eve alliance info (related name of `alliance`)

static generic_logo_url (*alliance_id: int, size: int = 32*) → str
image URL for the given alliance ID

logo_url (*size: int = 32*) → str
image URL of this alliance

property logo_url_128
image URL for this alliance

property logo_url_256
image URL for this alliance

property logo_url_32
image URL for this alliance

property logo_url_64
image URL for this alliance

class EveCharacter (*args, **kwargs)
Character in Eve Online

Parameters

- **id** (*AutoField*) – Primary key: ID
- **character_id** (*PositiveIntegerField*) – Character id
- **character_name** (*CharField*) – Character name
- **corporation_id** (*PositiveIntegerField*) – Corporation id
- **corporation_name** (*CharField*) – Corporation name
- **corporation_ticker** (*CharField*) – Corporation ticker
- **alliance_id** (*PositiveIntegerField*) – Alliance id
- **alliance_name** (*CharField*) – Alliance name
- **alliance_ticker** (*CharField*) – Alliance ticker
- **faction_id** (*PositiveIntegerField*) – Faction id
- **faction_name** (*CharField*) – Faction name

Reverse relationships:

Parameters

- **state** (Reverse *ManyToManyField* from State) – All states of this eve character (related name of member_characters)
- **userprofile** (Reverse *OneToOneField* from UserProfile) – The user profile of this eve character (related name of main_character)
- **character_ownership** (Reverse *OneToOneField* from CharacterOwnership) – The character ownership of this eve character (related name of character)
- **ownership_records** (Reverse *ForeignKey* from OwnershipRecord) – All ownership records of this eve character (related name of character)
- **application** (Reverse *ForeignKey* from Application) – All applications of this eve character (related name of reviewer_character)
- **timer** (Reverse *ForeignKey* from Timer) – All timers of this eve character (related name of eve_character)
- **srpfleetmain** (Reverse *ForeignKey* from SrpFleetMain) – All srp fleet mains of this eve character (related name of fleet_commander)
- **srpuserrequest** (Reverse *ForeignKey* from SrpUserRequest) – All srp user requests of this eve character (related name of character)
- **optimizer** (Reverse *ForeignKey* from OpTimer) – All op timers of this eve character (related name of eve_character)

- **fat** (Reverse `ForeignKey` from Fat) – All fats of this eve character (related name of character)

property alliance

Pseudo foreign key from `alliance_id` to `EveAllianceInfo` :raises: `EveAllianceInfo.DoesNotExist` :return: `EveAllianceInfo` or `None`

alliance_logo_url (*size=32*) → *str*

image URL for alliance of this character or empty string

property alliance_logo_url_128

image URL for alliance of this character or empty string

property alliance_logo_url_256

image URL for alliance of this character or empty string

property alliance_logo_url_32

image URL for alliance of this character or empty string

property alliance_logo_url_64

image URL for alliance of this character or empty string

property corporation

Pseudo foreign key from `corporation_id` to `EveCorporationInfo` :raises: `EveCorporationInfo.DoesNotExist` :return: `EveCorporationInfo`

corporation_logo_url (*size=32*) → *str*

image URL for corporation of this character

property corporation_logo_url_128

image URL for corporation of this character

property corporation_logo_url_256

image URL for corporation of this character

property corporation_logo_url_32

image URL for corporation of this character

property corporation_logo_url_64

image URL for corporation of this character

property faction

Pseudo foreign key from `faction_id` to `EveFactionInfo` :raises: `EveFactionInfo.DoesNotExist` :return: `EveFactionInfo`

faction_logo_url (*size=32*) → *str*

image URL for alliance of this character or empty string

property faction_logo_url_128

image URL for alliance of this character or empty string

property faction_logo_url_256

image URL for alliance of this character or empty string

property faction_logo_url_32

image URL for alliance of this character or empty string

property faction_logo_url_64

image URL for alliance of this character or empty string

static generic_portrait_url (*character_id: int, size: int = 32*) → *str*

image URL for the given character ID

property is_biomassed

Whether this character is dead or not.

portrait_url (*size=32*) → str

image URL for this character

property portrait_url_128

image URL for this character

property portrait_url_256

image URL for this character

property portrait_url_32

image URL for this character

property portrait_url_64

image URL for this character

```
class EveCorporationInfo(id, corporation_id, corporation_name, corporation_ticker, member_count, ceo_id, alliance)
```

Parameters

- **id** (*AutoField*) – Primary key: ID
- **corporation_id** (*PositiveIntegerField*) – Corporation id
- **corporation_name** (*CharField*) – Corporation name
- **corporation_ticker** (*CharField*) – Corporation ticker
- **member_count** (*IntegerField*) – Member count
- **ceo_id** (*PositiveIntegerField*) – Ceo id

Relationship fields:

Parameters alliance (*ForeignKey* to *EveAllianceInfo*) – Alliance (related name: evecorporationinfo)

Reverse relationships:

Parameters

- **state** (Reverse *ManyToManyField* from *State*) – All states of this eve corporation info (related name of member_corporations)
- **managedcorpgroup** (Reverse *ForeignKey* from *ManagedCorpGroup*) – All managed corp groups of this eve corporation info (related name of corp)
- **applicationform** (Reverse *OneToOneField* from *ApplicationForm*) – The application form of this eve corporation info (related name of corp)
- **timer** (Reverse *ForeignKey* from *Timer*) – All timers of this eve corporation info (related name of eve_corp)
- **corpstats** (Reverse *OneToOneField* from *CorpStats*) – The corp stats of this eve corporation info (related name of corp)

static generic_logo_url (*corporation_id: int, size: int = 32*) → str

image URL for the given corporation ID

logo_url (*size: int = 32*) → str

image URL for this corporation

property logo_url_128

image URL for this corporation

property logo_url_256
image URL for this corporation

property logo_url_32
image URL for this corporation

property logo_url_64
image URL for this corporation

class EveFactionInfo (*id, faction_id, faction_name*)

Parameters

- **id** (*AutoField*) – Primary key: ID
- **faction_id** (*PositiveIntegerField*) – Faction id
- **faction_name** (*CharField*) – Faction name

Reverse relationships:

Parameters state (Reverse *ManyToManyField* from State) – All states of this eve faction
info (related name of member_factions)

static generic_logo_url (*faction_id: int, size: int = 32*) → *str*
image URL for the given faction ID

logo_url (*size: int = 32*) → *str*
image URL of this faction

property logo_url_128
image URL for this faction

property logo_url_256
image URL for this faction

property logo_url_32
image URL for this faction

property logo_url_64
image URL for this faction

notifications

The notifications package has an API for sending notifications.

Location: `allianceauth.notifications`

models

class Notification (**args, **kwargs*)

Notification to a user within Auth

Parameters

- **id** (*AutoField*) – Primary key: ID
- **level** (*CharField*) – Level
- **title** (*CharField*) – Title
- **message** (*TextField*) – Message

- **timestamp** (*DateTimeField*) – Timestamp
- **viewed** (*BooleanField*) – Viewed

Relationship fields:

Parameters **user** (*ForeignKey* to *User*) – User (related name: *notification*)

class **Level** (*value*)

A notification level.

DANGER = 'danger'

INFO = 'info'

SUCCESS = 'success'

WARNING = 'warning'

classmethod **from_old_name** (*name: str*) → *object*

Map old name to enum.

Raises *ValueError* for invalid names.

mark_viewed () → *None*

Mark notification as viewed.

set_level (*level_name: str*) → *None*

Set notification level according to old level name, e.g. 'CRITICAL'.

Raises *ValueError* on invalid level names.

managers

class **NotificationManager** (**args, **kwargs*)

notify_user (*user: object, title: str, message: Optional[str] = None, level: str = 'info'*) → *object*

Sends a new notification to user. Returns newly created notification object.

user_unread_count (*user_pk: int*) → *int*

returns the cached unread count for a user given by user PK

Will return -1 if user can not be found

tests

Here you find utility functions and classes, which can help speed up writing test cases for AA.

Location: `allianceauth.tests.auth_utils`

auth_utils

class AuthUtils

Utilities for making it easier to create tests for Alliance Auth

```
classmethod add_main_character (user, name, character_id, corp_id=2345, corp_name="",  
                                corp_ticker="", alliance_id=None, alliance_name="", fac-  
                                tion_id=None, faction_name="")
```

```
classmethod add_main_character_2 (user, name, character_id, corp_id=2345,  
                                corp_name="", corp_ticker="", alliance_id=None,  
                                alliance_name="", disconnect_signals=False) →  
                                allianceauth.eveonline.models.EveCharacter
```

new version that works in all cases

```
classmethod add_permission_to_user_by_name (perm, user, disconnect_signals=True)  
→ django.contrib.auth.models.User
```

returns permission specified by qualified name

perm: Permission name as 'app_label.codename'

user: user object

disconnect_signals: whether to run process without signals

```
classmethod add_permissions_to_groups (perms, groups, disconnect_signals=True)
```

```
classmethod add_permissions_to_state (perms, states, disconnect_signals=True)
```

```
classmethod add_permissions_to_user (perms, user, disconnect_signals=True) →  
                                django.contrib.auth.models.User
```

add list of permissions to user

perms: list of Permission objects

user: user object

disconnect_signals: whether to run process without signals

```
classmethod add_permissions_to_user_by_name (perms: List[str], user:  
                                django.contrib.auth.models.User,  
                                disconnect_signals: bool = True) →  
                                django.contrib.auth.models.User
```

Add permissions given by name to a user

Parameters

- **perms** – List of permission names as 'app_label.codename'
- **user** – user object
- **disconnect_signals** – whether to run process without signals

Returns Updated user object

```
classmethod assign_state (user, state, disconnect_signals=False)
```

```
classmethod connect_signals ()
```

```
classmethod create_member (username)
```

```
classmethod create_state (name, priority, member_characters=None, mem-  
                                ber_corporations=None, member_alliances=None, public=False,  
                                disconnect_signals=False)
```

```

classmethod create_user (username, disconnect_signals=False)
    create a new user

    username: Name of the user

    disconnect_signals: whether to run process without signals

classmethod disconnect_signals ()

classmethod get_guest_state ()

classmethod get_member_state ()

static get_permission_by_name (perm: str) → django.contrib.auth.models.Permission
    returns permission specified by qualified name

    perm: Permission name as 'app_label.codename'

    Returns: Permission object or throws exception if not found

class BaseViewTestCase (methodName='runTest')

    login ()

    setUp ()
        Hook method for setting up the test fixture before exercising it.

```

utils

Utilities and helper functions.

Location: `allianceauth.utils`

- [cache](#)
- [testing](#)

cache

get_redis_client () → `redis.client.Redis`

Get the configured redis client used by Django for caching.

This function is a wrapper designed to work for both AA2 and AA3 and should always be used to ensure backwards compatibility.

testing

class NoSocketsTestCase (*methodName='runTest'*)

Variation of Django's `TestCase` class that prevents any network use.

Example

```
class TestMyStuff(NoSocketsTestCase):
    def test_should_do_what_i_need(self):
        ...
```

exception SocketAccessError

Error raised when a test script accesses the network

6.4.2 Celery FAQ

Alliance Auth uses Celery for asynchronous task management. This page aims to give developers some guidance on how to use Celery when developing apps for Alliance Auth.

For a complete documentation of Celery please refer to the [official Celery documentation](#).

When should I use Celery in my app?

There are two main cases for using celery. Long duration of a process and recurrence of a process.

Duration

Alliance Auth is an online web application and as such the user expects fast and immediate responses to any of his clicks or actions. Same as with any other good web site. Good response times are measures in ms and a user will perceive everything that takes longer than 1 sec as an interruption of his flow of thought (see also [Response Times: The 3 Important Limits](#)).

As a rule of thumb we therefore recommend to use celery tasks for every process that can take longer than 1 sec to complete (also think about how long your process might take with large amounts of data).

Note: Another solution for dealing with long response time in particular when loading pages is to load parts of a page asynchronously, for example with AJAX.

Recurrence

Another case for using celery tasks is when you need recurring execution of tasks. For example you may want to update the list of characters in a corporation from ESI every hour.

These are called periodic tasks and Alliance Auth uses [celery beat](#) to implement them.

What is a celery task?

For the most part a celery task is a Python functions that is configured to be executed asynchronously and controlled by Celery. Celery tasks can be automatically retried, executed periodically, executed in work flows and much more. See the [celery docs](#) for a more detailed description.

How should I use Celery in my app?

Please use the following approach to ensure your tasks are working properly with Alliance Auth:

- All tasks should be defined in a module of your app's package called `tasks.py`
- Every task is a Python function with has the `@shared_task` decorator.
- Task functions and the tasks module should be kept slim, just like views by mostly utilizing business logic defined in your models/managers.
- Tasks should always have logging, so their function and potential errors can be monitored properly

Here is an example implementation of a task:

```
import logging
from celery import shared_task

logger = logging.getLogger(__name__)

@shared_task
def example():
    logger.info('example task started')
```

This task can then be started from any another Python module like so:

```
from .tasks import example

example.delay()
```

How should I use celery tasks in the UI?

There is a well established pattern for integrating asynchronous processes in the UI, for example when the user asks your app to perform a longer running action:

1. Notify the user immediately (with a Django message) that the process for completing the action has been started and that he will receive a report once completed.
2. Start the celery task
3. Once the celery task is completed it should send a notification containing the result of the action to the user. It's important to send that notification also in case of errors.

Can I use long running tasks?

Long running tasks are possible, but in general Celery works best with short running tasks. Therefore we strongly recommend to try and break down long running tasks into smaller tasks if possible.

If contextually possible try to break down your long running task in shorter tasks that can run in parallel.

However, many long running tasks consist of several smaller processes that need to run one after the other. For example you may have a loop where you perform the same action on hundreds of objects. In those cases you can define each of the smaller processes as it's own task and then link them together, so that they are run one after the other. That is called chaining in Celery and is the preferred approach for implementing long running processes.

Example implementation for a celery chain:

```
import logging
from celery import shared_task, chain

logger = logging.getLogger(__name__)

@shared_task
def example():
    logger.info('example task')

@shared_task
def long_runner():
    logger.info('started long runner')
    my_tasks = list()
    for _ in range(10):
        task_signature = example.si()
        my_task.append(task_signature)

    chain(my_tasks).delay()
```

In this example we first add 10 example tasks that need to run one after the other to a list. This can be done by creating a so called signature for a task. Those signature are a kind of wrapper for tasks and can be used in various ways to compose work flow for tasks.

The list of task signatures is then converted to a chain and started asynchronously.

Hint: In our example we use `si()`, which is a shortcut for “immutable signatures” and prevents us from having to deal with result sharing between tasks.

For more information on signature and work flows see the official documentation on [Canvas](#).

In this context please note that Alliance Auth currently only supports chaining, because all other variants require a so called results back, which Alliance Auth does not have.

How can I define periodic tasks for my app?

Periodic tasks are normal celery tasks which are added the scheduler for periodic execution. The convention for defining periodic tasks for an app is to define them in the local settings. So user will need to add those settings manually to his local settings during the installation process.

Example setting:

```
CELERYBEAT_SCHEDULE['structures_update_all_structures'] = {
    'task': 'structures.tasks.update_all_structures',
    'schedule': crontab(minute='*/30'),
}
```

- `structures_update_all_structures` is the name of the scheduling entry. You can chose any name, but the convention is name of your app plus name of the task.
- `'task':` Name of your task (full path)
- `'schedule':` Schedule definition (see Celery documentation on [Periodic Tasks](#) for details)

How can I use priorities for tasks?

In Alliance Auth we have defined task priorities from 0 - 9 as follows:

| Number | Priority | Description |
|--------|----------|--|
| 0 | Reserved | Reserved for Auth and may not be used by apps |
| 1, 2 | Highest | Needs to run right now |
| 3, 4 | High | needs to run as soon as practical |
| 5 | Normal | default priority for most tasks |
| 6, 7 | Low | needs to run soonish, but is less urgent than most tasks |
| 8, 9 | Lowest | not urgent, can be run whenever there is time |

Warning: Please make sure to use task priorities with care and especially do not use higher priorities without a good reason. All apps including Alliance Auth share the same task queues, so using higher task priorities excessively can potentially prevent more important tasks (of other apps) from completing on time.

You also want to make sure to run use lower priorities if you have a large amount of tasks or long running tasks, which are not super urgent. (e.g. the regular update of all Eve characters from ESI runs with priority 7)

Hint: If no priority is specified all tasks will be started with the default priority, which is 5.

To run a task with a different priority you need to specify it when starting it.

Example for starting a task with priority 3:

```
example.apply_async(priority=3)
```

Hint: For defining a priority to tasks you can not use the convenient shortcut `delay()`, but instead need to start a task with `apply_async()`, which also requires you to pass parameters to your task function differently. Please check out the [official docs](#) for details.

What special features should I be aware of?

Every Alliance Auth installation will come with a couple of special celery related features “out-of-the-box” that you can make use of in your apps.

celery-once

Celery-once is a celery extension “that allows you to prevent multiple execution and queuing of celery tasks”. What that means is that you can ensure that only one instance of a celery task runs at any given time. This can be useful for example if you do not want multiple instances of you task to talk to the same external service at the same time.

We use a custom backend for celery_once in Alliance Auth defined [here](#) You can import it for use like so:

```
from allianceauth.services.tasks import QueueOnce
```

An example of AllianceAuth’s use within the @sharedtask decorator, can be seen [here](#) in the discord module You can use it like so:

```
@shared_task(bind=True, name='your_modules.update_task', base=QueueOnce)
```

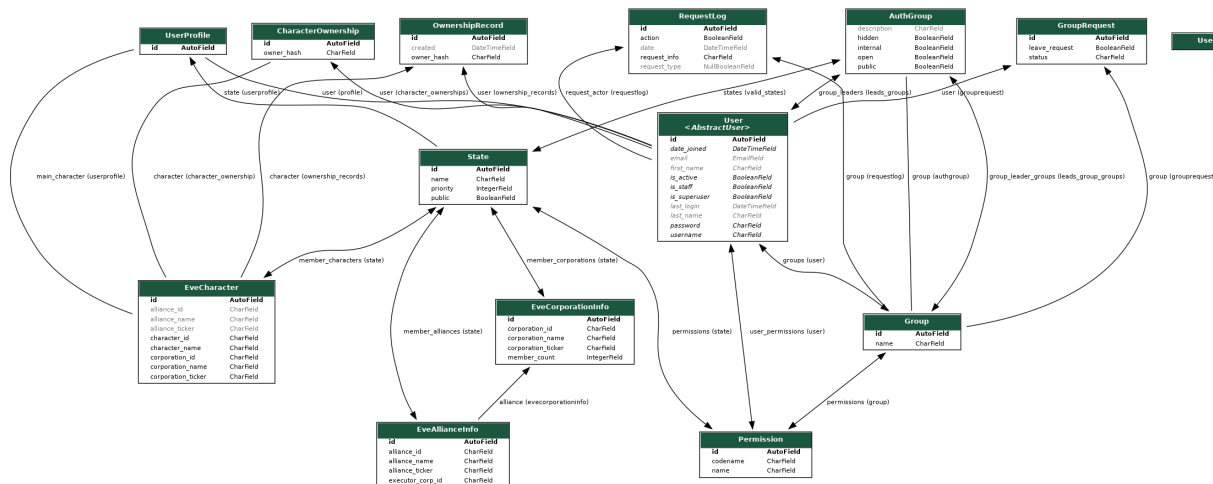
Please see the [official documentation](#) of celery-once for details.

task priorities

Alliance Auth is using task priorities to enable priority based scheduling of task execution. Please see [How can I use priorities for tasks?](#) for details.

6.4.3 Core models

The following diagram shows the core models of AA and Django and their relationships:



6.4.4 Template tags & filters

The following template tags and filters are available to be used by all apps. To use them just load them into your template like so:

```
{% load evelinks %}
```

Template Filters

evelinks

Example for using an evelinks filter to render an alliance logo:

```

```

alliance_logo_url (*eve_obj: object, size: int = 32*) → str

generates image URL for the given object Works with EveCharacter, EveAllianceInfo objects or alliance IDs
Returns URL or empty string

character_portrait_url (*eve_obj: object, size: int = 32*) → str

generates an image URL for the given object Works with EveCharacter objects or character IDs Returns URL or empty string

corporation_logo_url (*eve_obj: object, size: int = 32*) → str

generates image URL for the given object Works with EveCharacter, EveCorporationInfo objects or corporation IDs Returns URL or empty string

dotlan_alliance_url (*eve_obj: object*) → str

generates a dotlan URL for the given object Works with allianceauth.eveonline objects and eve entity names
Returns URL or empty string

dotlan_corporation_url (*eve_obj: object*) → str

generates a dotlan URL for the given object Works with allianceauth.eveonline objects and eve entity names
Returns URL or empty string

dotlan_region_url (*eve_obj: object*) → str

generates a dotlan URL for the given object Works with eve entity names Returns URL or empty string

dotlan_solar_system_url (*eve_obj: object*) → str

generates a dotlan URL for the given object Works with eve entity names Returns URL or empty string

evewho_alliance_url (*eve_obj: object*) → str

generates an evewho URL for the given object Works with allianceauth.eveonline objects and eve entity IDs
Returns URL or empty string

evewho_character_url (*eve_obj: allianceauth.eveonline.models.EveCharacter*) → str

generates an evewho URL for the given object Works with allianceauth.eveonline objects and eve entity IDs
Returns URL or empty string

evewho_corporation_url (*eve_obj: object*) → str

generates an evewho URL for the given object Works with allianceauth.eveonline objects and eve entity IDs
Returns URL or empty string

type_icon_url (*type_id: int, size: int = 32*) → str

generates a icon image URL for the given type ID Returns URL or empty string

type_render_url (*type_id: int, size: int = 32*) → str

generates a render image URL for the given type ID Returns URL or empty string

zkillboard_alliance_url (*eve_obj: object*) → str

generates a zkillboard URL for the given object Works with allianceauth.eveonline objects and eve entity IDs
Returns URL or empty string

zkillboard_character_url (*eve_obj: allianceauth.eveonline.models.EveCharacter*) → str

generates a zkillboard URL for the given object Works with allianceauth.eveonline objects and eve entity IDs
Returns URL or empty string

zkillboard_corporation_url (*eve_obj: object*) → str

generates a zkillboard URL for the given object Works with allianceauth.eveonline objects and eve entity IDs
Returns URL or empty string

zkillboard_region_url (*eve_obj: object*) → str

generates a zkillboard URL for the given object Works with eve entity IDs Returns URL or empty string

zkillboard_solar_system_url (*eve_obj: object*) → str

generates zkillboard URL for the given object Works with eve entity IDs Returns URL or empty string

CONTRIBUTING

Alliance Auth is developed by the community and we are always looking to welcome new contributors. If you are interested in contributing, here are some ideas where to start:

7.1 Publish a new community app or service

One great way to contribute is to develop and publish your own community app or service for Alliance Auth. By design Auth only comes with some basic features and therefore heavily relies on the community to provide apps to extend Auth with additional features.

To publish your app make sure it can be installed from a public repo or PyPI. Once it's ready, you can inform everybody about your new app by posting it to our [list of community apps](#).

If you are looking for ideas on what to make, you can check out Auth's [issue list](#). Many of those issues are feature requests that will probably never make into Auth core, but would be awesome to have as community app or service. You could also ask the other devs on our Discord server for ideas or to help you get a feeling about which new features might be in higher demand than others.

7.2 Help to maintain an existing community app or service

There are quite a few great community apps that need help from additional maintainers. Often the initial author has no time anymore to support his app or would just appreciate some support for working on new features or to fix bugs.

Sometimes original app owners may even be looking to completely hand over their apps to a new owner.

If you are interested to help maintain an existing community app or service you can just start working on open issues and create merge requests. Or just ask other devs on our Discord.

7.3 Help with improving Auth documentation

Auth has an extensive [documentation](#), but there are always things to improve and add. If you notice any errors or see something to improve or add please feel free to issue a change for the documentation (via MRs same as code changes).

7.4 Help with support questions on Discord

One of the main functions of the Auth Discord server is to help the community with any support question they may have when installing or running an Auth installation.

Note that you do not need to be part of any official group to become a supporter. Just jump in and help with answering new questions from the community if you know how to help.

7.5 Help to improve Alliance Auth core

Alliance Auth has an issue list, which is usually the basis for all maintenance activities for Auth core. That means that bug fixes and new features are primarily delivered based on existing open issues.

We usually have a long list of open issues and very much welcome every help to fix existing bugs or work on new features for Auth.

Before starting to code on any topic we'd suggest talking to the other devs on Discord to make sure your issue is not already being worked on. Also, some feature request may be better implemented in a community app. Another aspect, which is best clarified by talking with the other devs.

If you like to contribute to Auth core, but are unsure where to start, we have a dedicated label for issues that are suitable for beginners: [beginner friendly](#).

7.6 Additional Resources

For more information on how to create community apps or how to setup a developer environment for Auth, please see our official [developer documentation](#).

For getting in touch with other contributors please feel free to join the [Alliance Auth Discord server](#).

PYTHON MODULE INDEX

a

- `allianceauth.analytics.tasks`, [31](#)
- `allianceauth.eveonline.evelinks.dotlan`,
[129](#)
- `allianceauth.eveonline.evelinks.eveimageserver`,
[129](#)
- `allianceauth.eveonline.evelinks.evewho`,
[129](#)
- `allianceauth.eveonline.evelinks.zkillboard`,
[130](#)
- `allianceauth.eveonline.models`, [130](#)
- `allianceauth.eveonline.templatetags.evelinks`,
[143](#)
- `allianceauth.notifications.__init__`, [134](#)
- `allianceauth.services.modules.discord.api`,
[127](#)
- `allianceauth.services.modules.discord.app_settings`,
[128](#)
- `allianceauth.services.modules.discord.discord_client.app_settings`,
[126](#)
- `allianceauth.services.modules.discord.discord_client.client`,
[121](#)
- `allianceauth.services.modules.discord.discord_client.exceptions`,
[126](#)
- `allianceauth.services.modules.discord.discord_client.models`,
[124](#)
- `allianceauth.tests.auth_utils`, [136](#)
- `allianceauth.utils.cache`, [137](#)
- `allianceauth.utils.testing`, [137](#)

Symbols

`__init__()` (*DiscordApiBackoff* method), 126
`__init__()` (*DiscordClient* method), 121
`__init__()` (*Guild* method), 125
`__init__()` (*GuildMember* method), 125
`__init__()` (*Role* method), 125, 127
`__init__()` (*User* method), 124

A

`access_token()` (*DiscordClient* property), 121
`add_guild_member()` (*DiscordClient* method), 121
`add_guild_member_role()` (*DiscordClient* method), 121
`add_main_character()` (*AuthUtils* class method), 136
`add_main_character_2()` (*AuthUtils* class method), 136
`add_permission_to_user_by_name()` (*AuthUtils* class method), 136
`add_permissions_to_groups()` (*AuthUtils* class method), 136
`add_permissions_to_state()` (*AuthUtils* class method), 136
`add_permissions_to_user()` (*AuthUtils* class method), 136
`add_permissions_to_user_by_name()` (*AuthUtils* class method), 136
`alliance()` (*EveCharacter* property), 132
`alliance_logo_url()` (*EveCharacter* method), 132
`alliance_logo_url()` (in module *allianceauth.eveonline.evelinks.eveimageserver*), 129
`alliance_logo_url()` (in module *allianceauth.eveonline.templatetags.evelinks*), 143
`alliance_logo_url_128()` (*EveCharacter* property), 132
`alliance_logo_url_256()` (*EveCharacter* property), 132
`alliance_logo_url_32()` (*EveCharacter* property), 132

`alliance_logo_url_64()` (*EveCharacter* property), 132
`alliance_url()` (in module *allianceauth.eveonline.evelinks.dotlan*), 129
`alliance_url()` (in module *allianceauth.eveonline.evelinks.evewho*), 129
`alliance_url()` (in module *allianceauth.eveonline.evelinks.zkillboard*), 130
`allianceauth.analytics.tasks` module, 31
`allianceauth.eveonline.evelinks.dotlan` module, 129
`allianceauth.eveonline.evelinks.eveimageserver` module, 129
`allianceauth.eveonline.evelinks.evewho` module, 129
`allianceauth.eveonline.evelinks.zkillboard` module, 130
`allianceauth.eveonline.models` module, 130
`allianceauth.eveonline.templatetags.evelinks` module, 143
`allianceauth.notifications.__init__` module, 134
`allianceauth.services.modules.discord.api` module, 127
`allianceauth.services.modules.discord.app_settings` module, 128
`allianceauth.services.modules.discord.discord_client` module, 126
`allianceauth.services.modules.discord.discord_client` module, 121
`allianceauth.services.modules.discord.discord_client` module, 126
`allianceauth.services.modules.discord.discord_client` module, 124
`allianceauth.tests.auth_utils` module, 136
`allianceauth.utils.cache` module, 137
`allianceauth.utils.testing`

module, 137
analytics_event() (in module *allianceauth.analytics.tasks*), 31
asdict() (*Role* method), 125, 127
assign_state() (*AuthUtils* class method), 136
AuthUtils (class in *allianceauth.tests.auth_utils*), 136

B

BaseViewTestCase (class in *allianceauth.tests.auth_utils*), 137

C

character_portrait_url() (in module *allianceauth.eveonline.evelinks.eveimageserver*), 129
character_portrait_url() (in module *allianceauth.eveonline.templatetags.evelinks*), 143
character_url() (in module *allianceauth.eveonline.evelinks.evewho*), 129
character_url() (in module *allianceauth.eveonline.evelinks.zkillboard*), 130
connect_signals() (*AuthUtils* class method), 136
corporation() (*EveCharacter* property), 132
corporation_logo_url() (*EveCharacter* method), 132
corporation_logo_url() (in module *allianceauth.eveonline.evelinks.eveimageserver*), 129
corporation_logo_url() (in module *allianceauth.eveonline.templatetags.evelinks*), 143
corporation_logo_url_128() (*EveCharacter* property), 132
corporation_logo_url_256() (*EveCharacter* property), 132
corporation_logo_url_32() (*EveCharacter* property), 132
corporation_logo_url_64() (*EveCharacter* property), 132
corporation_url() (in module *allianceauth.eveonline.evelinks.dotlan*), 129
corporation_url() (in module *allianceauth.eveonline.evelinks.evewho*), 129
corporation_url() (in module *allianceauth.eveonline.evelinks.zkillboard*), 130
create_bot_client() (in module *allianceauth.services.modules.discord.api*), 128
create_guild_role() (*DiscordClient* method), 121
create_member() (*AuthUtils* class method), 136

create_state() (*AuthUtils* class method), 136
create_user() (*AuthUtils* class method), 136
current_user() (*DiscordClient* method), 122

D

DANGER (*Notification.Level* attribute), 135
delete_guild_role() (*DiscordClient* method), 122
disconnect_signals() (*AuthUtils* class method), 137
DISCORD_API_BASE_URL (in module *allianceauth.services.modules.discord.discord_client.app_settings*), 126
DISCORD_API_TIMEOUT_CONNECT (in module *allianceauth.services.modules.discord.discord_client.app_settings*), 126
DISCORD_API_TIMEOUT_READ (in module *allianceauth.services.modules.discord.discord_client.app_settings*), 126
DISCORD_APP_ID (in module *allianceauth.services.modules.discord.app_settings*), 128
DISCORD_APP_SECRET (in module *allianceauth.services.modules.discord.app_settings*), 128
DISCORD_BOT_TOKEN (in module *allianceauth.services.modules.discord.app_settings*), 128
DISCORD_CALLBACK_URL (in module *allianceauth.services.modules.discord.app_settings*), 128
DISCORD_DISABLE_ROLE_CREATION (in module *allianceauth.services.modules.discord.discord_client.app_settings*), 126
DISCORD_GUILD_ID (in module *allianceauth.services.modules.discord.app_settings*), 128
DISCORD_GUILD_NAME_CACHE_MAX_AGE (in module *allianceauth.services.modules.discord.discord_client.app_settings*), 126
DISCORD_OAUTH_BASE_URL (in module *allianceauth.services.modules.discord.discord_client.app_settings*), 126
DISCORD_OAUTH_TOKEN_URL (in module *allianceauth.services.modules.discord.discord_client.app_settings*), 126
DISCORD_ROLES_CACHE_MAX_AGE (in module *allianceauth.services.modules.discord.discord_client.app_settings*), 126
DISCORD_SYNC_NAMES (in module *allianceauth.services.modules.discord.app_settings*), 128

DISCORD_TASKS_MAX_RETRIES (in module *allianceauth.services.modules.discord.app_settings*), 128

DISCORD_TASKS_RETRY_PAUSE (in module *allianceauth.services.modules.discord.app_settings*), 128

DiscordApiBackoff, 126

DiscordApiStatusCode (class in *allianceauth.services.modules.discord.discord_client.client*), 121

DiscordClient (class in *allianceauth.services.modules.discord.discord_client.client*), 121

DiscordClientException, 126

DiscordRateLimitExhausted, 126

DiscordTooManyRequestsError, 126

DiscordUser (class in *allianceauth.services.modules.discord.api*), 127

discriminator (User attribute), 124

dotlan_alliance_url() (in module *allianceauth.eveonline.templatetags.evelinks*), 143

dotlan_corporation_url() (in module *allianceauth.eveonline.templatetags.evelinks*), 143

dotlan_region_url() (in module *allianceauth.eveonline.templatetags.evelinks*), 143

dotlan_solar_system_url() (in module *allianceauth.eveonline.templatetags.evelinks*), 143

faction_logo_url() (EveCharacter method), 132

faction_logo_url_128() (EveCharacter property), 132

faction_logo_url_256() (EveCharacter property), 132

faction_logo_url_32() (EveCharacter property), 132

faction_logo_url_64() (EveCharacter property), 132

from_dict() (Guild class method), 125

from_dict() (GuildMember class method), 125

from_dict() (Role class method), 125, 127

from_dict() (User class method), 124

from_old_name() (Notification.Level class method), 135

G

generic_logo_url() (EveAllianceInfo static method), 130

generic_logo_url() (EveCorporationInfo static method), 133

generic_logo_url() (EveFactionInfo static method), 134

generic_portrait_url() (EveCharacter static method), 132

get_guest_state() (AuthUtils class method), 137

get_member_state() (AuthUtils class method), 137

get_permission_by_name() (AuthUtils static method), 137

get_redis_client() (in module *allianceauth.utils.cache*), 137

group_to_role() (in module *allianceauth.services.modules.discord.api*), 128

E

EveAllianceInfo (class in *allianceauth.eveonline.models*), 130

EveCharacter (class in *allianceauth.eveonline.models*), 131

EveCorporationInfo (class in *allianceauth.eveonline.models*), 133

EveFactionInfo (class in *allianceauth.eveonline.models*), 134

evewho_alliance_url() (in module *allianceauth.eveonline.templatetags.evelinks*), 143

evewho_character_url() (in module *allianceauth.eveonline.templatetags.evelinks*), 143

evewho_corporation_url() (in module *allianceauth.eveonline.templatetags.evelinks*), 143

Guild (class in *allianceauth.services.modules.discord.discord_client.models*), 125

guild_infos() (DiscordClient method), 122

guild_member() (DiscordClient method), 122

guild_member_roles() (DiscordClient method), 122

guild_name() (DiscordClient method), 122

guild_roles() (DiscordClient method), 122

GuildMember (class in *allianceauth.services.modules.discord.discord_client.models*), 125

id (Guild attribute), 125

id (Role attribute), 125

id (User attribute), 124

INFO (Notification.Level attribute), 135

is_biomassed() (EveCharacter property), 132

is_rate_limited() (DiscordClient property), 122

F

faction() (EveCharacter property), 132

L

login() (*BaseViewTestCase* method), 137
 logo_url() (*EveAllianceInfo* method), 130
 logo_url() (*EveCorporationInfo* method), 133
 logo_url() (*EveFactionInfo* method), 134
 logo_url_128() (*EveAllianceInfo* property), 130
 logo_url_128() (*EveCorporationInfo* property), 133
 logo_url_128() (*EveFactionInfo* property), 134
 logo_url_256() (*EveAllianceInfo* property), 130
 logo_url_256() (*EveCorporationInfo* property), 134
 logo_url_256() (*EveFactionInfo* property), 134
 logo_url_32() (*EveAllianceInfo* property), 131
 logo_url_32() (*EveCorporationInfo* property), 134
 logo_url_32() (*EveFactionInfo* property), 134
 logo_url_64() (*EveAllianceInfo* property), 131
 logo_url_64() (*EveCorporationInfo* property), 134
 logo_url_64() (*EveFactionInfo* property), 134

M

managed (*Role* attribute), 125
 mark_viewed() (*Notification* method), 135
 match_or_create_role_from_name() (*DiscordClient* method), 122
 match_or_create_roles_from_names() (*DiscordClient* method), 123
 match_or_create_roles_from_names_2() (*DiscordClient* method), 123
 match_role_from_name() (*DiscordClient* method), 123
 modify_guild_member() (*DiscordClient* method), 123
 module
 allianceauth.analytics.tasks, 31
 allianceauth.eveonline.evelinks.dotlan, 129
 allianceauth.eveonline.evelinks.eveimageserver, 129
 allianceauth.eveonline.evelinks.evewho, 129
 allianceauth.eveonline.evelinks.zkillboard, 130
 allianceauth.eveonline.models, 130
 allianceauth.eveonline.template_tags.evelinks, 143
 allianceauth.notifications.__init__, 134
 allianceauth.services.modules.discord.api, 127
 allianceauth.services.modules.discord.app_settings, 128
 allianceauth.services.modules.discord.discord_client, 126
 allianceauth.services.modules.discord.discord_client_models, 121

allianceauth.services.modules.discord.discord_client, 126
 allianceauth.services.modules.discord.discord_client_models, 124
 allianceauth.tests.auth_utils, 136
 allianceauth.utils.cache, 137
 allianceauth.utils.testing, 137

N

name (*Guild* attribute), 125
 name (*Role* attribute), 125
 nick (*GuildMember* attribute), 125
 NoSocketsTestCase (class in *allianceauth.utils.testing*), 137
 Notification (class in *allianceauth.notifications.models*), 134
 Notification.Level (class in *allianceauth.notifications.models*), 135
 NotificationManager (class in *allianceauth.notifications.managers*), 135
 notify_user() (*NotificationManager* method), 135

P

portrait_url() (*EveCharacter* method), 133
 portrait_url_128() (*EveCharacter* property), 133
 portrait_url_256() (*EveCharacter* property), 133
 portrait_url_32() (*EveCharacter* property), 133
 portrait_url_64() (*EveCharacter* property), 133

R

region_url() (in module *allianceauth.eveonline.evelinks.dotlan*), 129
 region_url() (in module *allianceauth.eveonline.evelinks.zkillboard*), 130
 remove_guild_member() (*DiscordClient* method), 124
 remove_guild_member_role() (*DiscordClient* method), 124
 retry_after_seconds() (*DiscordApiBackoff* property), 126
 Role (class in *allianceauth.services.modules.discord.api*), 127
 Role (class in *allianceauth.services.modules.discord.discord_client.models*), 125
 roles (*Guild* attribute), 125
 roles (*GuildMember* attribute), 125

S

sanitize_name() (*Role* class method), 125, 127
 sanitize_name() (*GuildMember* class method), 125
 server_name() (in module *allianceauth.services.modules.discord.api*), 128

[set_level\(\)](#) (*Notification method*), [135](#)
[setUp\(\)](#) (*BaseViewTestCase method*), [137](#)
[SocketAccessError](#), [138](#)
[solar_system_url\(\)](#) (in module *allianceauth.eveonline.evelinks.dotlan*), [129](#)
[solar_system_url\(\)](#) (in module *allianceauth.eveonline.evelinks.zkillboard*),
[130](#)
[SUCCESS](#) (*Notification.Level attribute*), [135](#)

T

[type_icon_url\(\)](#) (in module *allianceauth.eveonline.evelinks.eveimageserver*),
[129](#)
[type_icon_url\(\)](#) (in module *allianceauth.eveonline.templatetags.evelinks*),
[143](#)
[type_render_url\(\)](#) (in module *allianceauth.eveonline.evelinks.eveimageserver*),
[129](#)
[type_render_url\(\)](#) (in module *allianceauth.eveonline.templatetags.evelinks*),
[143](#)

U

[UNKNOWN_MEMBER](#) (*DiscordApiStatusCode attribute*),
[121](#)
[User](#) (*class in allianceauth.services.modules.discord.discord_client.models*),
[124](#)
[user](#) (*GuildMember attribute*), [125](#)
[user_unread_count\(\)](#) (*NotificationManager method*), [135](#)
[username](#) (*User attribute*), [124](#)

W

[WARNING](#) (*Notification.Level attribute*), [135](#)

Z

[zkillboard_alliance_url\(\)](#) (in module *allianceauth.eveonline.templatetags.evelinks*),
[143](#)
[zkillboard_character_url\(\)](#) (in module *allianceauth.eveonline.templatetags.evelinks*),
[144](#)
[zkillboard_corporation_url\(\)](#) (in module *allianceauth.eveonline.templatetags.evelinks*),
[144](#)
[zkillboard_region_url\(\)](#) (in module *allianceauth.eveonline.templatetags.evelinks*),
[144](#)
[zkillboard_solar_system_url\(\)](#) (in module *allianceauth.eveonline.templatetags.evelinks*),
[144](#)