# General Functionality of Project

The goal of this project is to provide a web API for use by frontends that seek to create interfaces for applications centered on high impact experiences at Xavier. An API necessitates a database to expose, and we chose Django as our web server for that purpose. Specifically, we decided to use the Django REST framework which is a separate library from Django itself that allows easy REST api creation.

We use an object relational mapping database that is reflected in Django ‘models’, which are classes that map directly to database objects. These models are essentially database tables, and they’re serialized into JSON to then be exposed at API endpoints, which can then be used by various frontends to populate their programs with data.

* Manage.py contains most command line tools in order to test the backend & database.
* API Documentation.docx contains the tentative API documentation

Everything mentioned here can be found in either the Django or Django REST framework documentation. Keep in mind that the REST framework will override the documentation of Django itself wherever there is conflict.

<https://docs.djangoproject.com/en/4.1/>

<https://www.django-rest-framework.org/>

# File Descriptions

* Note: If a file isn’t described here, it probably won’t need to be modified unless a major change is introduced. In this case, refer to in-file documentation.

## data folder (this is a Django “app” called “data”)

**models.py**

* Models are defined here, which are mapped to database tables.

**views.py**

* Since we use the Django REST Framework, we decided on viewsets rather than class-based views. Generally, they are used for defining API functionality

**fetchEvents.py**

* Example for local additions to the database; can be used in webscraping scripts. Scrapes ics events from Campus Groups (EngageXU) and places the data into the events model. As of writing, unused.

**Serializers.py**

* Serializers are defined for each model and are essentially what convert the models into JSON for the API.

## BackendDev folder

**settings.py**

**urls.py**

* Routes the url scheme, eg the database to urls that expose api endpoints.

**views.py**

* Differs from the data/views.py file.

# Environment Setup

See [Setup and Installation Documentation](https://myxavier.sharepoint.com/:w:/r/sites/CS261Fall20222/Shared%20Documents/Web%20Backend/Setup%20and%20Installation%20Documentation%20for%20other%20teams.docx?d=wef2f65f093f64aaab1950e5e2fff6eb4&csf=1&web=1&e=GLkSWG).

# Outstanding Issues

* We have been uncertain about how to implement temporary authentication because we have been left waiting on IT for access to the two authentication systems they use (azure for one). For now, the backend will handle authentication itself based on passed username:password credentials in the request.
* All automated tests have been completed and should pass in the current state of the repo, except for tests for the Events endpoint.
* We were going to write a parser for the ICS file in Engage, but we never got to it. That is what was supposed to go into the Events endpoint. What we have of the parser is in a file called fetchEvents.py. Also, we were thinking that the two auto generated fields (modified\_time and creation\_time) should not be autogenerated and should come from the parsed ICS file instead.
* We were going to attempt to revise the Serializer handling to allow sending and receiving `flattened` request data passed (as opposed to a nested JSON)
  + Specifically within the custom serializer’s Update and Create functions, which actually do the creation of the entry in the models.
  + All ModelSerializers have update/create functions inherited, which are linked to a singular model. When trying to engage with multiple models, we override the update/create functions within the serializer.
  + In other words, the API request would have all of the fields on the same level for both sending and receiving requests.
  + For example, for the student resource, the Django model would be the same as it is now, with the user field being nested within the profile field, but from the perspective of the API request, the username, email, name, and password fields would be on the same level as ID, major, minor, and schoolyear.
* We were going to put our existing local server on a hosting service (we were thinking AWS, since there is a free tier). This would be helpful for testing issues you might encounter in a more realistic environment with the server running longer term.