Note: As of right now, we do not have this on a remote server, so the listed URL is the local server. This means you must locally run this server on your machine in order to access the API. The setup process for this is described in the [Setup and Installation Documentation](https://myxavier.sharepoint.com/sites/CS261Fall20222/Shared%20Documents/Web%20Backend/Setup%20and%20Installation%20Documentation%20for%20other%20teams.docx).

This documentation is based off the documentation for the SWAPI: <https://swapi.dev/documentation>

# A Note on User Tables

**User table:** A default Django table to store essential attributes pertaining to a user, such as username, password (hash-encoded automatically), email, first\_name, and last\_name.

**Profile table:** An additional table, 1:1 directly linked with User table. This stores additional attributes for users not in the user table, such as prefix, suffix, and role.

**Student/Professor/AdminAssistant tables:** Top-level tables pertaining to a user. These have a foreign-key relation to the profile table, and thus the underlying user table. These tables store role-specific attributes, such as Major/Minor for a Student, or Degree\_Desc for a Professor.

# API Resources (Model-Tables)

In **bold** are the names of the resources which have the names of their attributes in the bullet points. This is a representation of the format of the API.

**Department**

* id – *integer* the id of the department (auto generated, read only)
* Name – *string* the name of the department

**Major**

* id – *integer* the id for the Major’s entry, aka the index in the table (auto generated, read only)
* name – *string* the name of the major
* subject – *Departments ID* the ID for the department instance that the major is a part of

**Minor**

* id – *integer* the id for the Minor’s entry, aka the index in the table (auto generated, read only)
* name – *string* the name of the minor
* subject – *Departments ID* the ID for the department instance that the minor is a part of

**Student**

* id – *integer* the id for the Student’s entry, aka the index in the table (auto generated, read only)
* prof – *Profile instance* the profile table attributes for the Student
  + user – *User instance* the user table attributes for the Student
    - username – *string* the username for the user
    - email – *string* the email for the user
    - first\_name – *string* the first name of the user
    - last\_name – *string* the last name of the user
    - password – the password to set for the user on create/update calls. This will be hash-encoded when stored in the database table. This field will not be returned in a GET request.
  + prefix – string the prefix of the user, OPTIONAL (4 characters max)
  + suffix – *string* the suffix of the user, OPTIONAL (4 characters max)
  + role – *string* the abbreviation for the role of the user (auto generated, read only)
    - For reference-- (2 character abbreviations)
      * ‘ST’ -- the role abbreviation for STUDENT
      * ‘PR’ -- the role abbreviation for PROFESSOR
      * ‘AA’ -- the role abbreviation for ADMINASSISTANT
* major – *List of Major IDs* the list of major IDs for the major instances the student has
* minor – *List of Minor IDs* the list of minor IDs for the minor instances the student has
* schoolyear – *string* the year the student is in (2 characters max), possible schoolyears:
  + FRESHMAN = 'FR'
  + SOPHOMORE = 'SO'
  + JUNIOR = 'JR'
  + SENIOR = 'SR'
  + GRADUATE = 'GR'

**Professor**

* id – *integer* the id for the Professor’s entry, aka the index in the table. (auto generated, read only)
* prof – *Profile instance* the profile table attributes for the Professor
  + user – *User instance* the user table attributes for the Professor
    - username – *string* the username for the user
    - email – *string* the email for the user
    - first\_name – *string* the first name of the user
    - last\_name – *string* the last name of the user
    - password – the password to set for the user on create/update calls. This will be hash-encoded when stored in the database table. This field will not be returned in a GET request.
  + prefix – string the prefix of the user, OPTIONAL (4 characters max)
  + suffix – *string* the suffix of the user, OPTIONAL (4 characters max)
  + role – *string* the abbreviation for the role of the user. (auto generated, read only)
    - For reference-- (2 character abbreviations)
      * ‘ST’ -- the role abbreviation for STUDENT
      * ‘PR’ -- the role abbreviation for PROFESSOR
      * ‘AA’ -- the role abbreviation for ADMINASSISTANT
* Department – *Departments ID* the ID for the department instance that the professor belongs to
* degree\_desc – *string* the description of the professor’s degree(s)

**AdminAssistant – Department Admins, such as Donna Wallace for CompSci dept.**

* id – *integer* the id for the AdminAssistant’s entry, aka the index in the table. (auto generated, read only)
* prof – *Profile instance* the profile table attributes for the AdminAssistant
  + user – *User instance* the user table attributes for the AdminAssistant
    - username – *string* the username for the user
    - email – *string* the email for the user
    - first\_name – *string* the first name of the user
    - last\_name – *string* the last name of the user
    - password – the password to set for the user on create/update calls. This will be hash-encoded when stored in the database table. This field will not be returned in a GET request.
  + prefix – string the prefix of the user, OPTIONAL (4 characters max)
  + suffix – *string* the suffix of the user, OPTIONAL (4 characters max)
  + role – *string* the abbreviation for the role of the user. (auto generated, read only)
    - For reference-- (2 character abbreviations)
      * ‘ST’ -- the role abbreviation for STUDENT
      * ‘PR’ -- the role abbreviation for PROFESSOR
      * ‘AA’ -- the role abbreviation for ADMINASSISTANT
* Department – *Departments ID* the ID for the department instance that the AdminAssistant belongs to

**HighImpactExperience**

* id – *integer* the id of the event (7 digits max)
* name – *string* the title of the experience
* RTX\_name – *string* a shorter description of the experience
* Freshman\_desc – *string* description of experiences that are relevant to Freshmen
* Sophomore\_desc – *string* description of experiences that are relevant to Sophomores
* Junior\_desc – *string* description of experiences that are relevant to Juniors
* Senior\_desc – *string* description of experiences that are relevant to Seniors
* creation\_date – *dateTime* when the experience was added to the table
* area – *Departments ID* the ID for the department instance that the experience is associated with
* advisor – *User ID* the ID for the user instance that the experience is associated with (can be null)

**Course**

* crn – *integer* the id of the course (5 digits max)
* title – *string* the name of the course
* desc\_text – *string* the description of the course
* course\_num – *integer* the 3-digit course number of the course
* subject – *Departments ID* the ID for the department instance that the Course is under
* instructor – *Professor ID* the ID for the professor instance that teaches the Course
* credit\_hours – *integer* the number of credit hours the course is worth

**Event – Events that are in EngageXU**

* id – *integer* the id of the event and PK in the event table (7 digits max). User-specified.
* name – *string* the name of the event
* start\_time – *dateTime* the start time of the event
* end\_time – *dateTime* the end time of the event
* creation\_time – *dateTime* when the event was created. Auto-generated by Backend, READ-ONLY.
* modified\_time – *dateTime* when the event was last modified.
* url – *string* the EngageXU URL of the event
* location – *string* the location of the event
* categories – *string* the event type and event tags of the event
* organizer – User *ID* the ID for the user instance that the Event is organized by
* description – *string* a longer description of the event
* hie – *HighImpactExperience ID* the ID for the HighImpactExperience instance that the Event may be linked to

# URL Root

<http://127.0.0.1:8000/> is the root URL for the API. This should be the local IP for your computer if you are running the server. You should also be able to access it via <http://localhost:8000/.> Every time you make an API request, the URL should start with one of a URL root. If you receive a 404 error, try calling the root and see if you get one to make sure that the server itself is not down.

# Requests Library

If you are using Python, we recommend using the [Requests Library](https://requests.readthedocs.io/en/latest/) to do API requests programmatically. This is just one library that can be used to make calls within Python. This also has options for Authentication (username:password) and Data to pass with the request.

Example:  
requests.post("http://127.0.0.1:8000/api/Departments/", data = {"name": "Test7"})

Notice that the data field is a dictionary, with the keys being the field names, and the values being the values in the fields.

# HTTPie

[httpie](https://httpie.io/) is a command line tool used to make API calls. It can be installed using pip install httpie (assuming you have Python installed). Generally, this would just be used for testing purposes when you don’t want to use the Requests library for whatever reason.

# Return Format

Per the [Django Rest Framework Documentation](https://www.django-rest-framework.org/topics/browsable-api/#formats), By default, the API will return the format specified by the headers. The format can be specified using <?format=> in the request, so you can make sure you return a JSON object by adding <?format=json> to the end of the URL.

This would generally only need to be used if you are using a tool that can’t convert what is being returned to a JSON (or another desired format). So this isn’t necessary when using the Requests library since a request can be returned in a JSON format using the .json() function.

# Referencing other Resource Instances

Sometimes you may need to reference instances of other resources, namely when creating or updating a resource that has a field which is related to a different resource instance. Such fields are indicated above in the resource definitions.

This is done by simply setting the field to the index (ID/primary key) of the resource instance. These ID’s can be found in returned data for List or Retrieve actions, as well as in the response data after a successful Create or Update action.

For example, if we want to set the subject field of a Major instance to our first Department instance, we will create the following JSON object specifying Subject (department) ID value of 1.  
{"name": "New Major", "subject": 1}

# Permissions

There are 3 permissions groups for users: **Students**, **Professors**, and **AdminAssistants**. These are separate from the Student/Professor/AdminAssistant models, which are just for storing unique characteristics to each user role. Whether or not a requesting user (via passing credentials/authentication in the request) can do an action depends on the permissions of the user, meaning the permission group they are in.

**Students** (permission group) can do the following actions to the following resources (models/tables):

* LIST / RETRIEVE: Departments, Majors, Minors, Courses, HIEs, Events, Students, Professors
  + **Students resource:** Requesting user is limited in scope to their own entry (i.e., can’t retrieve a different Student’s information).
* CREATE: Students
* UPDATE / PARTIAL\_UPDATE: Students
  + **Students resource:** Requesting user is limited in scope to their own entry (i.e., can’t Update a different Student’s information).
* DELETE: Students
  + **Students resource:** Requesting user is limited in scope to their own entry (i.e., can’t Delete a different Student).

**Professors** (permission group) can do the following actions to the following resources (models/tables):

* LIST / RETRIEVE: Departments, Majors, Minors, Courses, HIEs, Events, Students, Professors, AdminAssistants
* CREATE: Courses, HIEs, Events, Students
* UPDATE / PARTIAL\_UPDATE: Courses, HIEs, Events, Professors
  + **Professors resource:** Requesting user is limited in scope to their own entry (i.e., can’t Update a different Professor’s information).
* DELETE: Courses, HIEs, Events, Professors
  + **Professors resource:** Requesting user is limited in scope to their own entry (i.e., can’t Delete a different Professor).

**AdminAssistants** (permission group) can do the following actions to the following resources (models/tables):

* LIST / RETRIEVE: Departments, Majors, Minors, Courses, HIEs, Events, Students, Professors, AdminAssistants
* CREATE: Courses, HIEs, Events, Majors, Minors, Students, Professors, AdminAssistants
* UPDATE / PARTIAL\_UPDATE: Courses, HIEs, Events, Majors, Minors, Students, Professors, AdminAssistants
  + **AdminAssistants resource:** Requesting user is limited in scope to their own entry (i.e., can’t Update a different AdminAssistant).
* DELETE: Courses, HIEs, Events, Majors, Minors, Students, Professors, AdminAssistants
  + **AdminAssistants resource:** Requesting user is limited in scope to their own entry (i.e., can’t Delete a different AdminAssistant).

Superusers can do all actions on all resources. Here are the unique permissions exclusive to superusers:

* CREATE: Departments, AdminAssistants
* UPDATE / PARTIAL\_UPDATE: Departments, AdminAssistants
* DELETE: Departments, AdminAssistants

A superuser is created when running the startup script as described in the [Setup and Installation Documentation](https://myxavier.sharepoint.com/sites/CS261Fall20222/Shared%20Documents/Web%20Backend/Setup%20and%20Installation%20Documentation%20for%20other%20teams.docx).

# Authentication

We expect that when making API calls/requests, the frontend will pass the authentication request and credentials from the user to the API and the backend will do the rest.

If the user does not have the appropriate permissions required to access a resource, an error will be returned stating so. See [the Permissions section](#_Permissions) for information on specific resource permissions.

When using [httpie](#_HTTPie), you can authenticate your request using the -a or --auth flag, and then authenticating using the format username:password. For example:  
http -a admin:password http://127.0.0.1:8000/api/Departments/

When using the [Requests library](#_Requests_Library), you can authenticate using an HTTPBasicAuth object as a parameter of a request. For example:   
from requests.auth import HTTPBasicAuth  
userAuth = HTTPBasicAuth(‘admin’, ‘password’)  
requests.get(‘http://127.0.0.1:8000/api/Departments/’, auth=userAuth)  
# alternatively  
requests.get(‘http://127.0.0.1:8000/api/Departments/’, auth=(‘admin’, ‘password’))

However you end up making a request, authentication credentials must be sent with each request for which they are required. They do not persist through requests.

# List

This will return a list of JSON objects.



# Retrieve

This will retrieve a specific resource instance based on the requested PK/ID.

Pk is the Primary Key of the instance of the resource. This is generally the id field, or an equivalent. If this is not the case for a resource, it is notated in [the API Resources section](#_API_Resources).

Example using [httpie](#_HTTPie):

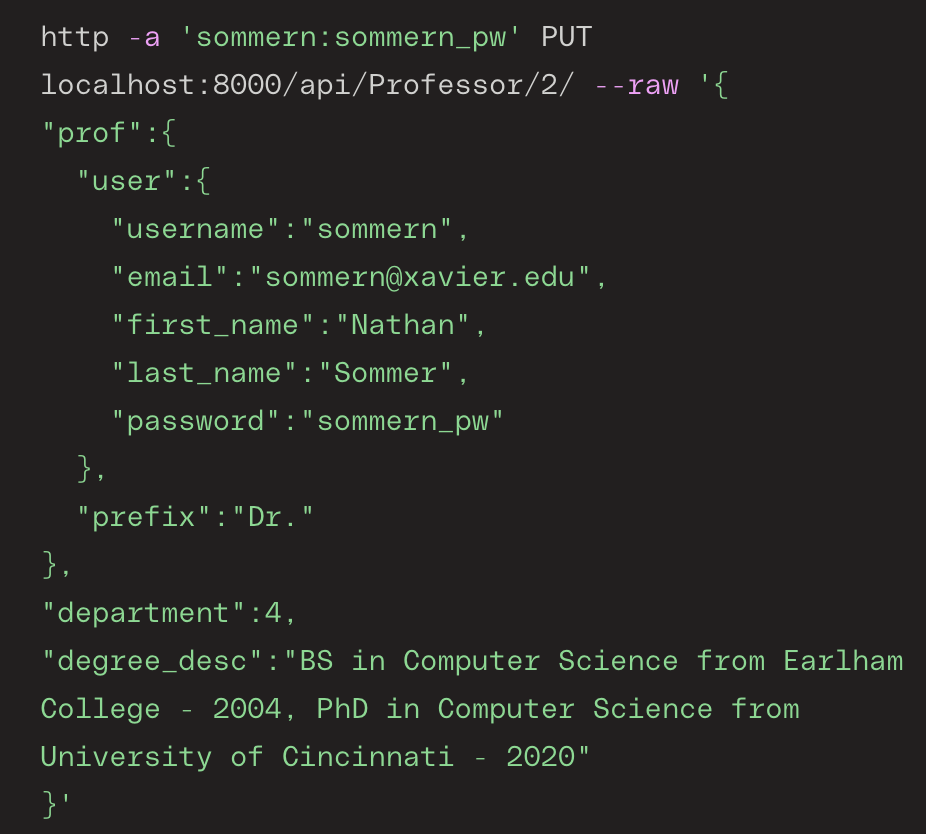
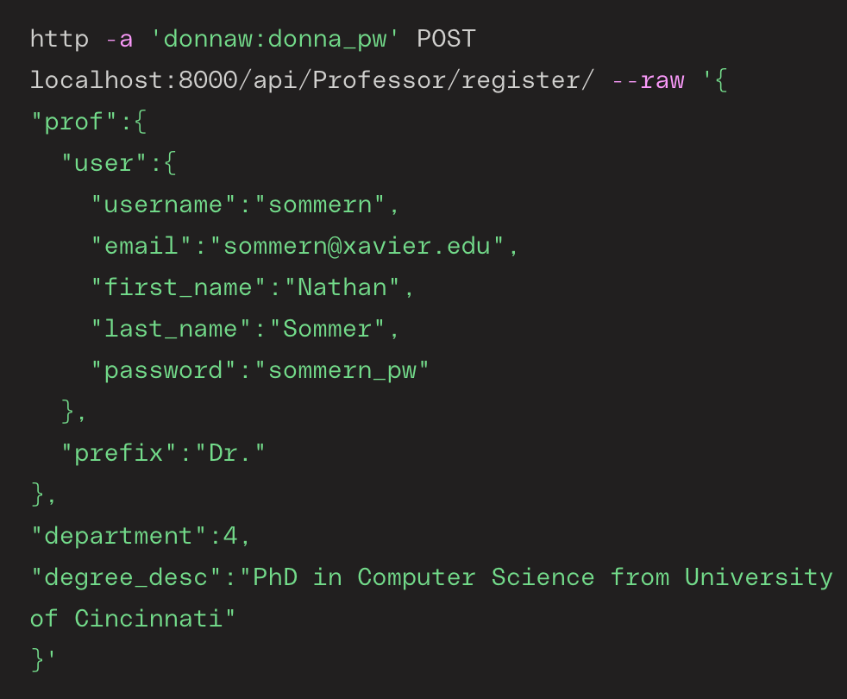


# Create/Update

In order to create a new instance of a resource, or update an existing instance of a resource, you must make a POST or PUT/PATCH request, respectively.

POST is used to create a new instance. PUT and PATCH are used for updating an existing instance. PUT requires that you specify all fields in your request, even if they are not the ones being updated. PATCH is a partial update which allows you to only specify the fields you are changing.

Examples for POST (Create), PUT (Update), PATCH (Partial Update) using [httpie](#_HTTPie):

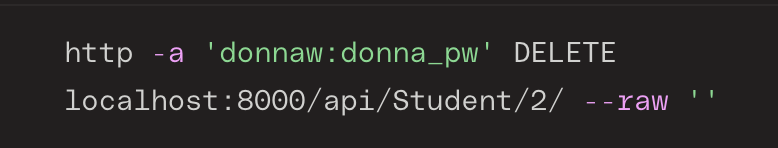


# Destroy

In order to delete an existing instance of a resource, you must make a DELETE request.

Deleting an instance *does not shift the indices/ids of the instances after the deleted instance*. Once an instance is deleted, it is not possible to make a PUT or PATCH request such that the index of the deleted instance has data put into it again.

Example using [httpie](#_HTTPie):



API Endpoints

These are callable paths created to access various endpoints of the API. These are all relative paths, where the “.” should be replaced with the URL Root, as described above.  
For example: *localhost:8000/api/Student/register/***Note that all URLs should end with a / and are case-sensitive.**

This first set of endpoints **can only be used on users**. There are three types of users: Student, Professor, and AdminAssistant. This distinction is further explained in [the A Note on User Tables section](#_Note_on_User). <userType> in the endpoint URL means to pass in the name of the type of user.  
For example: *./api/Student/register/*

* **./api/<userType>/register/**
  + POST request: This triggers the CREATE action. Must pass in JSON data for creating the userType instance. Authentication is required in this request for Professor and AdminAssistant. Authentication is not required for Student.
    - If the action is successful, and, if authentication is required, the request passed in valid credentials for a (USER) with appropriate permission for the request, this will return the newly created userType instance’s data.
* **./api/<userType>/list/**
  + GET request: This triggers the LIST action. Requires authentication passed in the request.
    - If the action is successful, and the request passed in valid credentials for a (USER) with appropriate permission for the request, this will return a list of userType instances that the authorized user has access to view.

This endpoint can be used for any resource type that is **not a user**. For a full list of resources, see [the Resources section](#_API_Resources_(Model-Tables)). <resourceName> in the endpoint URL means to pass in the name of the resource.  
For example: *./api/HighImpactExperiences/*

* **./api/<resourceName>/**
  + GET request: This triggers the LIST action. Requires authentication passed in the request.
    - If the action is successful, and the request passed in valid credentials for a (USER), this will return a list of the resource instances that the authorized user has access to view.
  + POST request: This triggers the CREATE action. Requires authentication passed in the request. Must pass in JSON for the attributes of the new resource.
    - If action successful, and request passed in valid credentials for a (USER), this will return the newly created resource’s data.

This endpoint can be used for any resource type, **including users**. For a full list of resources, see [the Resources section](#_API_Resources). <resourceName> in the endpoint URL means to pass in the name of the resource. <int:pk> in the endpoint URL means to pass in the Primary Key (or ID) for the instance of the resource.  
For example: *./api/HighImpactExperiences/1/*

* **./api/<resourceName>/<int:pk>/**
  + GET request: This triggers the RETRIEVE action. Requires authentication passed in the request. Must specify the resource given by ID in the path.
    - If the action is successful, and the request passed in valid credentials for a (USER) with appropriate permission for the request, this will return the resource’s data.
  + PUT request: This triggers the UPDATE action. Requires authentication passed in the request. Must pass in JSON data for updating all attributes of the resource instance given by ID in the path.
    - If the action is successful, and the request passed in valid credentials for a (USER) with appropriate permission for the request, this will return the updated resource’s data.
  + PATCH request: This triggers the PARTIAL\_UPDATE action. Requires authentication passed in the request. Must pass in JSON data for updating select attributes of the resource instance given by ID in the path.
    - If action successful, and request passed in valid credentials for a (USER) with appropriate permission for the request, this will return the updated resource’s data.
  + DELETE request: This triggers the DESTROY action. Requires authentication passed in the request. This deletes the specified resource instance from the database table.
    - If the action is successful, and the request passed in valid credentials for a (USER) with appropriate permission for the request, this will return the deletion success code.

These endpoints are mainly for debugging and other miscellaneous internal purposes.

* **./ or ./home/**
  + GET request: This will return a raw text list of all of the server endpoint paths.
* **./api/**
  + GET request: This will return an HTML formatted list of the number of tables that are user-facing, non-user-facing, and the number of entries in each table. It is recommended that you view this endpoint in a web browser.
* **./api/docs/**
  + GET request: This will display an HTML formatted documentation page using [drf-spectacular](https://drf-spectacular.readthedocs.io/en/latest/index.html). It is highly recommended that you view this endpoint in a web browser.