Knight Foundation School of Computing and Information Sciences

Fall 2022

Final Deliverable:

"Course Evaluation System"

Capstone 2: Gerardo Parra, Daniel Nunez Dominguez, Mariela Torres Zuniga, Jose Milanes Urquiza, Amani Hunt, Shaoting Wu, Arthur Vorojeykin

Capstone 1: Abel Aguilera, Gabriel Campillo, Saul Castillo, Schadrack Cineas, Blake Prieto, Ryan Rodriguez

Product Owner(s): Nagarajan Prabakar

Instructor: Masoud Sadjadi

Abstract

The Course Evaluation System (CES) is an online computer software system designed to conduct course evaluations at the end of every semester. The system provides students with a list of courses provided by the School of Computing and Information Sciences that are receiving evaluations and for which they are registered; after the evaluation period is over, the system generates reports maintaining the participant's anonymity, and then it allows administrators to look at the generated reports.

Table of Contents

Abstract ***2***
Current System *4*
Purpose of New System *4*
USER STORIES ***5***
Implemented User Stories *5*
Pending User Stories *9*
PROJECT PLAN ***10***
Hardware and Software Resources *11*
Sprints Plan *12*
SYSTEM DESIGN ***16***
Architectural Patterns To-do *16*
System and Subsystem Decomposition *18*
Deployment Diagram To-do *18*
Design Patterns To-do *19*
SYSTEM VALIDATION ***19***
GLOSSARY ***26***
APPENDIX ***26***
Appendix A - UML Diagrams *27*
1. Use Case Diagram 27
2. Sequence Diagram - Take Evaluation [Student] 28
Appendix B - User Interface Design [PENDING] *31*
Appendix C - Sprint Review Reports *31*
Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents 42
Wish List 46

INTRODUCTION

The Course Evaluation System (CES) is an online computer software system designed to conduct course evaluations at the end of every semester. The system provides students with a list of courses provided by the School of Computing and Information Sciences that are receiving evaluations and for which they are registered; after the evaluation period is over the system generates reports maintaining the participant's anonymity, and then it allows administrators to look at the generated reports.

Current System

The Course Evaluation System (CES) was first developed in 2006 by the KFSCIS web development team and was used for more than fifteen years at the end of each term. This classic package was built on a Postgres database with several Perl modules to collect student course evaluation data and to generate reports. This package is not modular enough to classify the collected data according to student majors and to generate reports specific for each KFSCIS degree program.

Purpose of New System

As such, for our Capstone 2 development, the purpose of the new system is to Design a new CES system with a modular design approach that would consist of:

- Secure web interface with FIU credentials for students (provide course outcome evaluation), faculty and administrators (access reports generated from the CES data).
- A database to host the student assessment data for every term- course-major without any inconsistencies and to support the report generation & ad hoc queries with GUI.
- Report generators to generate reports at various granular levels for a given group (term/year/course) with an extension feature to add new plugins.

- Documentation and help instructions for each group of users (student, faculty, administrator).

User Stories

The following sections discuss implemented and pending user stories Capstone 2.

Implemented User Stories

- **User Story #1**
  - As a product owner, I want to have everybody access to the resources needed so that we are ready to start to work on the project
  - Acceptance Criteria:
    - Set up a meeting with the product owner and go over project
    - Produce a product backlog

- **User Story #2**
  - As a developer, I want to familiarize myself with the tools needed so that we can effectively implement what is asked of us.
  - Acceptance Criteria:
    - Set up an environment (GitHub repository) for development.

- **User Story #3**
  - As a team player, I want to understand the scrum process so that I can collaborate with my team and produce the best product possible.
  - Acceptance Criteria:
    - Attend meetings and submit documents at their appropriate times.

- **User Story #4**
  - As a team player, I want to know the best time to congregate with the rest of my team so that we can all speak to each other and avoid rescheduling often.
  - Acceptance Criteria:
    - Set up a when2meet and have everyone input their available times.

- **User Story #5**
  - As a developer, I want to know my position in developing the product so that no redundant work is done.
  - Acceptance Criteria:
    - Assign roles and responsibilities

- **User Story #6**
  - As a developer, I want to review the current system.
  - Acceptance Criteria:
    - About 2 weeks to review the current system

- **User Story #7**
  - As a developer, I want to start designing the database.
  - Acceptance Criteria:
    - Implement ER diagram for the design of the database

- **User Story #8**
  - As a developer, I want to create a basic project in Django to start developing.
  - Acceptance Criteria:
    - Create a Django project.

- **User Story #9**
  - As a developer, I want to create the models.
  - Acceptance Criteria:
    - Create models for each table from the database design.

- **User Story #10**
  - As a developer, I want to create the respective serializers.
  - Acceptance Criteria:
    - Create serializers for each model.

- **User Story #11**
  - As a developer, I want to create the respective views.
  - Acceptance Criteria:
    - Create the views for each model.

- **User Story #12**
  - As a developer, I want to start creating the scripts.
  - Acceptance Criteria:
    - Create a modular CLI in Python
    - Create a modular CLI in Bash

- **User Story #13**
  - As a developer, I want to start creating authentication.
  - Acceptance criteria:
    - Add token based authentication to Django.

- **User Story #14**
  - As a developer, I want to create some rest actions.
  - Acceptance criteria:
    - Create possible actions needed in future development.
    - Create actions for report gathering..

- **User Story #15**
  - As a developer, I want to continue creating the scripts with the new database model.
  - Acceptance Criteria:
    - Modify and create scripts for new database design.

- **User Story #16**
  - As a developer, I want to finish creating authentication.
  - Acceptance criteria:
    - Terminate with token based authentication.

- **User Story #17**
  - As a developer, I want to continue creating more actions.
  - Acceptance criteria:
    - Create more actions for report gathering.

- **User Story #18**
  - As a developer, I want to create a frontend project.
  - Acceptance criteria:
    - Create a frontend project in React

- **User Story #19**
  - As a user (student), I want to be able to login.
  - Acceptance criteria:
    - Be able to enter credentials and being authenticated

- **User Story #20**
  - As a user (student), I want to be able to see available surveys.
  - Acceptance criteria:
    - Be able to see available surveys after the student is authenticated.
    - Be able to do not see surveys already evaluated
    - Be able to see some message saying that I completed all evaluations when this condition applies.

- **User Story #21**
  - As a user (student), I want to be able to take an evaluation.
  - Acceptance criteria:
    - Be able to click on an available survey and do the evaluation.
    - Be able to know if I missed some question with an error message in red color.
    - Be able to see some confirmation if the evaluation is submitted successfully.

- **User Story #22**
  - As a user (student), I want to be able to use available pages with no errors and with a good design.
  - Acceptance criteria:
    - Improve design in available pages.
    - Improve logic in available pages.

- **User Story #23**
  - As a user (faculty), I want to be able to login, and be authenticated and redirected to a faculty page.
  - Acceptance criteria:
    - Create a login page for the faculty site with authentication.

- **User Story #24**

    - As a user (faculty), I want to be able to gather reports by different ranges like semester, course, and year.
    - Acceptance criteria:
        - Create a faculty page to gather reports.
        - Gather reports by course, semester, and year.

- **User Story #25**
    - As a developer, I want to continue creating more actions for report gathering requested by frontend developers.
    - Acceptance criteria:
        - Create actions to gather reports by course, semester, and year.

- **User Story #26**
    - As a developer, I want to continue creating scripts for a faculty to enter data.
    - Acceptance criteria:
        - Create scripts for the admin to add data.

- **User Story #27**
    - As a developer , I want to make changes requested by PO in the faculty page.
    - Acceptance criteria:
        - Change report gathering for faculty.

- **User Story #28**
    - As a developer, I want to make changes requested by PO in the student page.
    - Acceptance criteria:
        - Change report gathering for students.

- **User Story #29**
    - As a user (admin), I want to be able to add data interactively to the database.
    - Acceptance criteria:
        - Create scripts for the admin to add data interactively.

- **User Story #30**
    - As a user (student, faculty, admin), I want to be able to use UI without bugs.

    - Acceptance criteria:
        - Wrap up the bugs.

- **User Story #31**
    - As a developer (future developers), I want to see all the history of development, and system architecture diagrams.
    - Acceptance criteria:
        - Document all development history and do formal documentation describing the whole system architecture.

- **User Story #32**
    - As a user (student, faculty, admin), I want to see video tutorials of how the system works.
    - Acceptance criteria:
        - Create videos showing all features of the application.

- **User Story #33**
    - As a developer, I want to run tests on the whole system.
    - Acceptance criteria:
        - Create test cases of the whole system.

Project Plan

In this section, the figures below illustrate the Gantt charts that show a summarized agenda of the meetings and tasks, along with their respective duration.

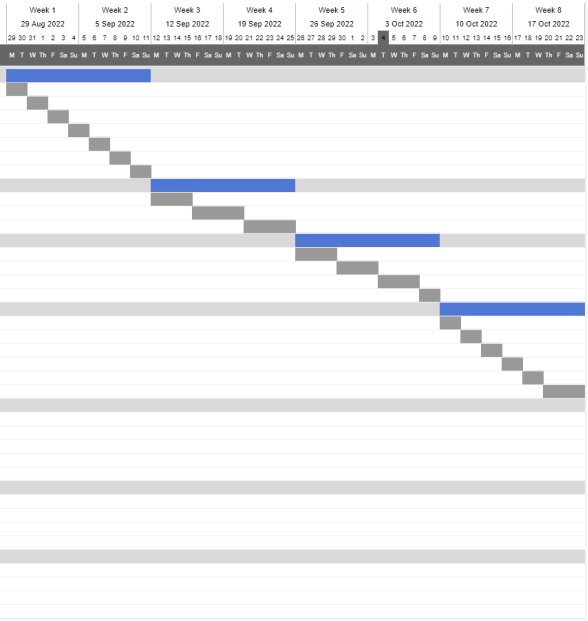
Course Evaluation System

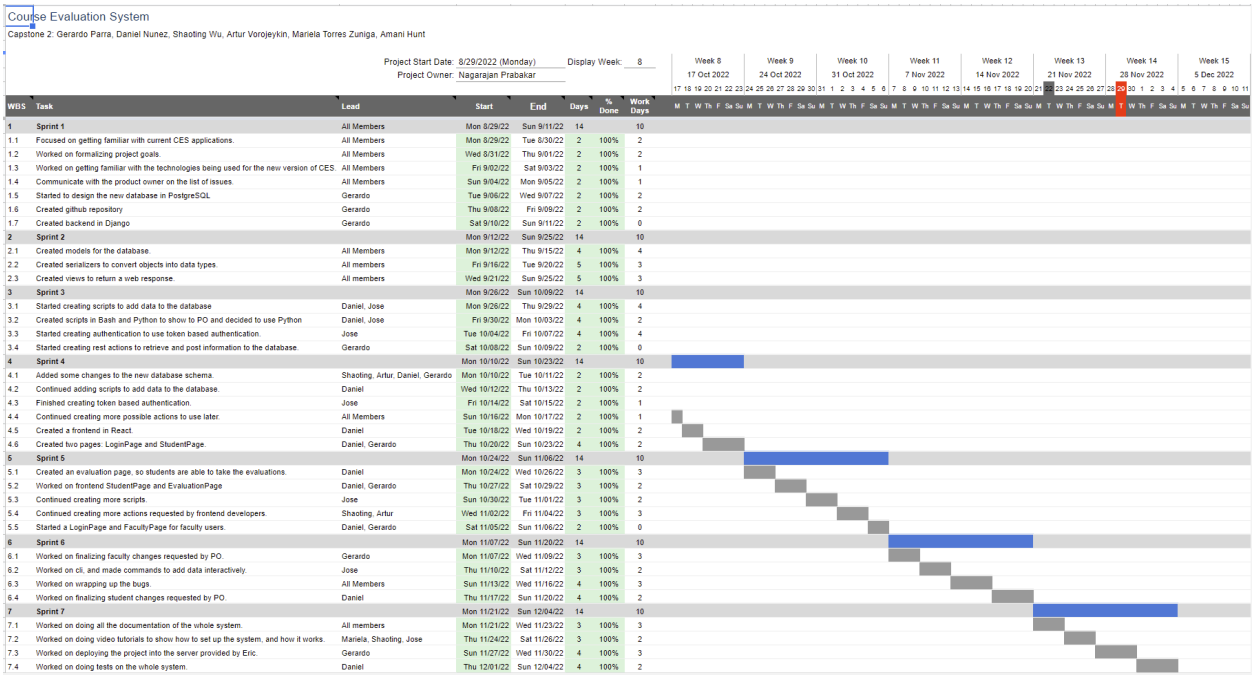
Capstone 2: Gerardo Parra, Daniel Nunez, Shaoting Wu, Artur Vorojeykin, Mariela Torres Zuniga, Amani Hunt

Project Start Date: 8/29/2022 (Monday) Display Week: 1

Project Owner: Nagarajan Prabakar

WBS	Task	Lead	Start	End	Days	% Done	Work Days
1	Sprint 1	All Members	Mon 8/29/22	Sun 9/11/22	14		10
1.1	Focused on getting familiar with current CES applications.	All Members	Mon 8/29/22	Tue 8/30/22	2	100%	2
1.2	Worked on formalizing project goals.	All Members	Wed 8/31/22	Thu 9/01/22	2	100%	2
1.3	Worked on getting familiar with the technologies being used for the new version of CES.	All Members	Fri 9/02/22	Sat 9/03/22	2	100%	1
1.4	Communicate with the product owner on the list of issues.	All Members	Sun 9/04/22	Mon 9/05/22	2	100%	1
1.5	Started to design the new database in PostgreSQL	Gerardo	Tue 9/06/22	Wed 9/07/22	2	100%	2
1.6	Created github repository	Gerardo	Thu 9/08/22	Fri 9/09/22	2	100%	2
1.7	Created backend in Django	Gerardo	Sat 9/10/22	Sun 9/11/22	2	100%	0
2	Sprint 2		Mon 9/12/22	Sun 9/25/22	14		10
2.1	Created models for the database.	All Members	Mon 9/12/22	Thu 9/15/22	4	100%	4
2.2	Created serializers to convert objects into data types.	All members	Fri 9/16/22	Tue 9/20/22	5	100%	3
2.3	Created views to return a web response.	All members	Wed 9/21/22	Sun 9/25/22	5	100%	3
3	Sprint 3		Mon 9/26/22	Sun 10/09/22	14		10
3.1	Started creating scripts to add data to the database	Daniel, Jose	Mon 9/26/22	Thu 9/29/22	4	100%	4
3.2	Created scripts in Bash and Python to show to PO and decided to use Python	Daniel, Jose	Fri 9/30/22	Mon 10/03/22	4	100%	2
3.3	Started creating authentication to use token based authentication.	Jose	Tue 10/04/22	Fri 10/07/22	4	100%	4
3.4	Started creating rest actions to retrieve and post information to the database.	Gerardo	Sat 10/08/22	Sun 10/09/22	2	100%	0
4	Sprint 4		Mon 10/10/22	Sun 10/23/22	14		10
4.1	Added some changes to the new database schema.	Shaoting, Artur, Daniel, Gerardo	Mon 10/10/22	Tue 10/11/22	2	100%	2
4.2	Continued adding scripts to add data to the database.	Daniel	Wed 10/12/22	Thu 10/13/22	2	100%	2
4.3	Finished creating token based authentication.	Jose	Fri 10/14/22	Sat 10/15/22	2	100%	1
4.4	Continued creating more possible actions to use later.	All Members	Sun 10/16/22	Mon 10/17/22	2	100%	1
4.5	Created a frontend in React.	Daniel	Tue 10/18/22	Wed 10/19/22	2	100%	2
4.6	Created two pages: LoginPage and StudentPage.	Daniel, Gerardo	Thu 10/20/22	Sun 10/23/22	4	100%	2
5	Sprint 5		Mon 10/24/22	Sun 11/06/22	14		10
5.1	Created an evaluation page, so students are able to take the evaluations.	Daniel	Mon 10/24/22	Wed 10/26/22	3	100%	3
5.2	Worked on frontend StudentPage and EvaluationPage	Daniel, Gerardo	Thu 10/27/22	Sat 10/29/22	3	100%	2
5.3	Continued creating more scripts.	Jose	Sun 10/30/22	Tue 11/01/22	3	100%	2
5.4	Continued creating more actions requested by frontend developers.	Shaoting, Artur	Wed 11/02/22	Fri 11/04/22	3	100%	3
5.5	Started a LoginPage and FacultyPage for faculty users.	Daniel, Gerardo	Sat 11/05/22	Sun 11/06/22	2	100%	0
6	Sprint 6		Mon 11/07/22	Sun 11/20/22	14		10
6.1	Worked on finalizing faculty changes requested by PO.	Gerardo	Mon 11/07/22	Wed 11/09/22	3	100%	3
6.2	Worked on cli, and made commands to add data interactively.	Jose	Thu 11/10/22	Sat 11/12/22	3	100%	2
6.3	Worked on wrapping up the bugs.	All Members	Sun 11/13/22	Wed 11/16/22	4	100%	3
6.4	Worked on finalizing student changes requested by PO.	Daniel	Thu 11/17/22	Sun 11/20/22	4	100%	2
7	Sprint 7		Mon 11/21/22	Sun 12/04/22	14		10
7.1	Worked on doing all the documentation of the whole system.	All members	Mon 11/21/22	Wed 11/23/22	3	100%	3
7.2	Worked on doing video tutorials to show how to set up the system, and how it works.	Mariela, Shaoting, Jose	Thu 11/24/22	Sat 11/26/22	3	100%	2
7.3	Worked on deploying the project into the server provided by Eric.	Gerardo	Sun 11/27/22	Wed 11/30/22	4	100%	3
7.4	Worked on doing tests on the whole system.	Daniel	Thu 12/01/22	Sun 12/04/22	4	100%	2





Course Evaluation System

Capstone 2: Gerardo Parra, Daniel Nunez, Shaoting Wu, Artur Vorojeykin, Mariela Torres Zuniga, Amani Hunt

Project Start Date: 8/29/2022 (Monday) Display Week: 8

Project Owner: Nagarajan Prabakar

WBS	Task	Lead	Start	End	Days	% Done	Work Days
1	Sprint 1	All Members	Mon 8/29/22	Sun 9/11/22	14		10
1.1	Focused on getting familiar with current CES applications.	All Members	Mon 8/29/22	Tue 8/30/22	2	100%	2
1.2	Worked on formalizing project goals.	All Members	Wed 8/31/22	Thu 9/01/22	2	100%	2
1.3	Worked on getting familiar with the technologies being used for the new version of CES.	All Members	Fri 9/02/22	Sat 9/03/22	2	100%	1
1.4	Communicate with the product owner on the list of issues.	All Members	Sun 9/04/22	Mon 9/05/22	2	100%	1
1.5	Started to design the new database in PostgreSQL	Gerardo	Tue 9/06/22	Wed 9/07/22	2	100%	2
1.6	Created github repository	Gerardo	Thu 9/08/22	Fri 9/09/22	2	100%	2
1.7	Created backend in Django	Gerardo	Sat 9/10/22	Sun 9/11/22	2	100%	0
2	Sprint 2		Mon 9/12/22	Sun 9/25/22	14		10
2.1	Created models for the database.	All Members	Mon 9/12/22	Thu 9/15/22	4	100%	4
2.2	Created serializers to convert objects into data types.	All members	Fri 9/16/22	Tue 9/20/22	5	100%	3
2.3	Created views to return a web response.	All members	Wed 9/21/22	Sun 9/25/22	5	100%	3
3	Sprint 3		Mon 9/26/22	Sun 10/09/22	14		10
3.1	Started creating scripts to add data to the database	Daniel, Jose	Mon 9/26/22	Thu 9/29/22	4	100%	4
3.2	Created scripts in Bash and Python to show to PO and decided to use Python	Daniel, Jose	Fri 9/30/22	Mon 10/03/22	4	100%	2
3.3	Started creating authentication to use token based authentication.	Jose	Tue 10/04/22	Fri 10/07/22	4	100%	4
3.4	Started creating rest actions to retrieve and post information to the database.	Gerardo	Sat 10/08/22	Sun 10/09/22	2	100%	0
4	Sprint 4		Mon 10/10/22	Sun 10/23/22	14		10
4.1	Added some changes to the new database schema.	Shaoting, Artur, Daniel, Gerardo	Mon 10/10/22	Tue 10/11/22	2	100%	2
4.2	Continued adding scripts to add data to the database.	Daniel	Wed 10/12/22	Thu 10/13/22	2	100%	2
4.3	Finished creating token based authentication.	Jose	Fri 10/14/22	Sat 10/15/22	2	100%	1
4.4	Continued creating more possible actions to use later.	All Members	Sun 10/16/22	Mon 10/17/22	2	100%	1
4.5	Created a frontend in React.	Daniel	Tue 10/18/22	Wed 10/19/22	2	100%	2
4.6	Created two pages: LoginPage and StudentPage.	Daniel, Gerardo	Thu 10/20/22	Sun 10/23/22	4	100%	2
5	Sprint 5		Mon 10/24/22	Sun 11/06/22	14		10
5.1	Created an evaluation page, so students are able to take the evaluations.	Daniel	Mon 10/24/22	Wed 10/26/22	3	100%	3
5.2	Worked on frontend StudentPage and EvaluationPage	Daniel, Gerardo	Thu 10/27/22	Sat 10/29/22	3	100%	2
5.3	Continued creating more scripts.	Jose	Sun 10/30/22	Tue 11/01/22	3	100%	2
5.4	Continued creating more actions requested by frontend developers.	Shaoting, Artur	Wed 11/02/22	Fri 11/04/22	3	100%	3
5.5	Started a LoginPage and FacultyPage for faculty users.	Daniel, Gerardo	Sat 11/05/22	Sun 11/06/22	2	100%	0
6	Sprint 6		Mon 11/07/22	Sun 11/20/22	14		10
6.1	Worked on finalizing faculty changes requested by PO.	Gerardo	Mon 11/07/22	Wed 11/09/22	3	100%	3
6.2	Worked on cli, and made commands to add data interactively.	Jose	Thu 11/10/22	Sat 11/12/22	3	100%	2
6.3	Worked on wrapping up the bugs.	All Members	Sun 11/13/22	Wed 11/16/22	4	100%	3
6.4	Worked on finalizing student changes requested by PO.	Daniel	Thu 11/17/22	Sun 11/20/22	4	100%	2
7	Sprint 7		Mon 11/21/22	Sun 12/04/22	14		10
7.1	Worked on doing all the documentation of the whole system.	All members	Mon 11/21/22	Wed 11/23/22	3	100%	3
7.2	Worked on doing video tutorials to show how to set up the system, and how it works.	Mariela, Shaoting, Jose	Thu 11/24/22	Sat 11/26/22	3	100%	2
7.3	Worked on deploying the project into the server provided by Eric.	Gerardo	Sun 11/27/22	Wed 11/30/22	4	100%	3
7.4	Worked on doing tests on the whole system.	Daniel	Thu 12/01/22	Sun 12/04/22	4	100%	2

Hardware and Software Resources

The following is a list of all hardware and software resources that were used for this project:

- **Visual Studio IDE** with Python extension installed to develop and modify Python code and scripts.
- **Github**, a version control site, to maintain changes to our project.
- **Mozilla Firefox, Microsoft Edge, Chrome, and Safari** to explore the CES user interfaces.

- **Trello**, a calendar board site, helped the team understand what was expected of them and by when (with Power-Ups in Trello you can add user story points and have a Gantt chart for all team members to see).
- **PostgreSQL**, used to store all the data for CES.
- **pgAdmin**, used as a user interface to see all data in the database.
- **React**, used as a frontend framework for CES.
- **Django**, used as a backend framework for CES.
- Any x86 or arm based computer.

Sprints Plan

The following section shows all of our sprint plans from both Capstone 1 and 2's development periods.

Capstone 2 Development

Sprint 1

During Sprint 1, we:

- Focused on getting familiar with current CES applications.
- Worked on formalizing project goals.
- Worked on getting familiar with the technologies being used for the new version of CES.
- Communicate with the product owner on the list of issues.
- Started to design the new database in PostgreSQL
- Created github repository
- Created backend in Django

The product owner requested the following user stories to be completed this sprint. They are ordered by priority:

- **User Story #1**
  - As a product owner, I want to have everybody access to the resources needed so that we are ready to start to work on the project
- **User Story #2**
  - As a developer, I want to familiarize myself with the tools needed so that we can effectively implement what is asked of us.
- **User Story #3**
  - As a team player, I want to understand the scrum process so that I can collaborate with my team and produce the best product possible.
- **User Story #4**
  - As a team player, I want to know the best time to congregate with the rest of my team so that we can all speak to each other and avoid rescheduling often.

- **User Story #5**
  - As a developer, I want to know my position in developing the product so that no redundant work is done.
- **User Story #6**
  - As a developer, I want to review the current system.
- **User Story #7**
  - As a developer, I want to design the database.
- **User Story #8**
  - As a developer, I want to create a basic project in Django to start developing.

Sprint 2

During Sprint 2, we:

- Created models for the database.
- Created serializers to convert objects into data types.
- Created views to return a web response.

The product owner requested the following user stories to be completed this sprint. They are ordered by priority:

- **User Story #9**
  - As a developer, I want to create the models.
- **User Story #10**
  - As a developer, I want to create the respective serializers.
- **User Story #11**
  - As a developer, I want to create the respective views.

Sprint 3

During Sprint 3, we:

- Started creating scripts to add data to the database.
- Created scripts in Bash and Python to show to PO and decided to use Python, eventually.
- Started creating authentication to use token based authentication.
- Started creating rest actions to retrieve and post information to the database.

The product owner requested the following user stories to be completed this sprint. They are ordered by priority:

- **User Story #12**
  - As a developer, I want to start creating the scripts.
- **User Story #13**
  - As a developer, I want to start creating authentication.
- **User Story #14**
  - As a developer, I want to create some rest actions.

Sprint 4

During Sprint 4, we:

- Added some changes to the new database schema.
- Continued adding scripts to add data to the database.
- Finished creating token based authentication.
- Continued creating more possible actions to use later.
- Created a frontend in React.
- Created two pages: LoginPage and StudentPage.

The product owner requested the following user stories to be completed this sprint. They are ordered by priority:

- **User Story #15**
  - As a developer, I want to continue creating the scripts with the new database model.
- **User Story #16**
  - As a developer, I want to finish creating authentication.
- **User Story #17**
  - As a developer, I want to continue creating more actions.
- **User Story #18**
  - As a developer, I want to create a frontend project.
- **User Story #19**
  - As a user (student), I want to be able to login.

- **User Story #20**
  - As a user (student), I want to be able to see available surveys.

Sprint 5

During Sprint 5, we:

- Created an evaluation page, so students are able to take the evaluations.
- Worked on frontend StudentPage and EvaluationPage, especially in adding more logic and styles.
- Continued creating more scripts.
- Continued creating more actions requested by frontend developers.
- Started a LoginPage and FacultyPage for faculty users.

The product owner requested the following user stories to be completed this sprint. They are ordered by priority:

- **User Story #21**
  - As a user (student), I want to be able to take an evaluation.
- **User Story #22**
  - As a user (student), I want to be able to use available pages with no errors and with a good design.
- **User Story #23**
  - As a user (faculty), I want to be able to login, and be authenticated and redirected to a faculty page.
- **User Story #24**
  - As a user (faculty), I want to be able to gather reports by different ranges like semester, course, and year.
- **User Story #25**
  - As a developer, I want to continue creating more actions for report gathering requested by frontend developers.
- **User Story #26**
  - As a developer, I want to continue creating scripts for a faculty to enter data.

Sprint 6

During Sprint 6, we:

- Worked on finalizing faculty changes requested by PO.
- Worked on cli, and made commands to add data interactively.
- Worked on wrapping up the bugs.
- Worked on finalizing student changes requested by PO.

The product owner requested the following user stories to be completed this sprint. They are ordered by priority:

- **User Story #27**
  - As a developer , I want to make changes requested by PO in the faculty page.
- **User Story #28**
  - As a developer, I want to make changes requested by PO in the student page.
- **User Story #29**
  - As a user (admin), I want to be able to add data interactively to the database.
- **User Story #30**
  - As a user (student, faculty, admin), I want to be able to use UI without bugs.

Sprint 7

During Sprint 7, we:

- Worked on doing all the documentation of the whole system.
- Worked on doing video tutorials to show how to set up the system, and how it works.
- Worked on deploying the project into the server provided by Eric.
- Worked on doing tests on the whole system.

The product owner requested the following user stories to be completed this sprint. They are ordered by priority:

- **User Story #31**

    - As a developer (future developers), I want to see all the history of development, and system architecture diagrams.
- **User Story #32**
    - As a user (student, faculty, admin), I want to see video tutorials of how the system works.
- **User Story #33**
    - As a developer, I want to run tests on the whole system.

System Design

This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

Architectural Patterns

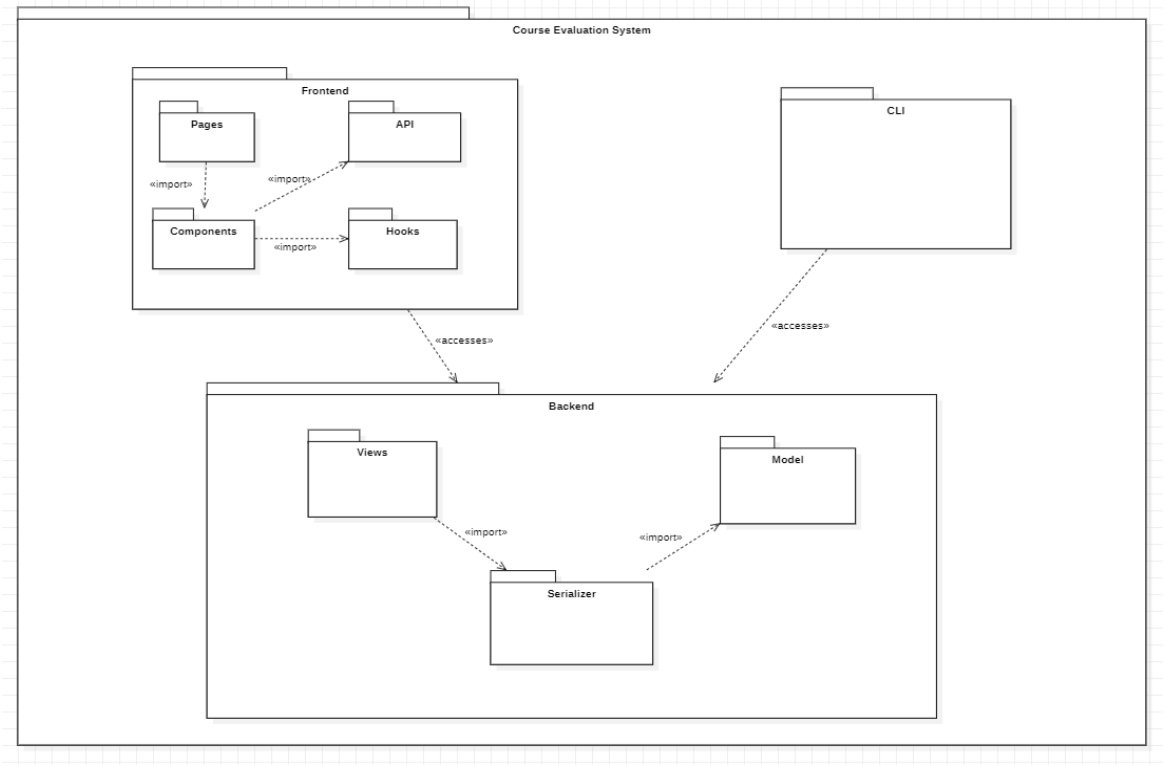
The architecture of the Course Evaluation System consists of three main components, the frontend, CLI, and the backend.

- The backend component is a REST API which is hosted by a web server. Because of this feature, a frontend can be developed in any framework/language that can do web requests. The backend reacts to different requests (GET, PUT, PATCH, OPTIONS) and depending on the type of the request, it generates different SQL queries that can create, get, or modify entries in the PostgreSQL database. The different endpoints created by the backend represent a table (or a combination of tables) on the database: /faculty, /courses, /outcomes, /sections, /students_in_section, etc. Some of these endpoints also contain actions which are used to query the database. For example, to find out if a faculty member is a subject area coordinator, we would send a GET request to the endpoint /faculty/<username>/is_area_coordinator/. The returned payload will contain true or false as a JSON object.
- The frontend component uses React to allow for rapid development of the UX. It communicates with the backend using the fetch API, which hits the specific endpoints

depending on what needs to be rendered. Some components prefetch from the REST API before being rendered using React hooks (useEffect). Since the backend implemented a token authentication system, every request has to include this token (which was exchanged as the user logged in).

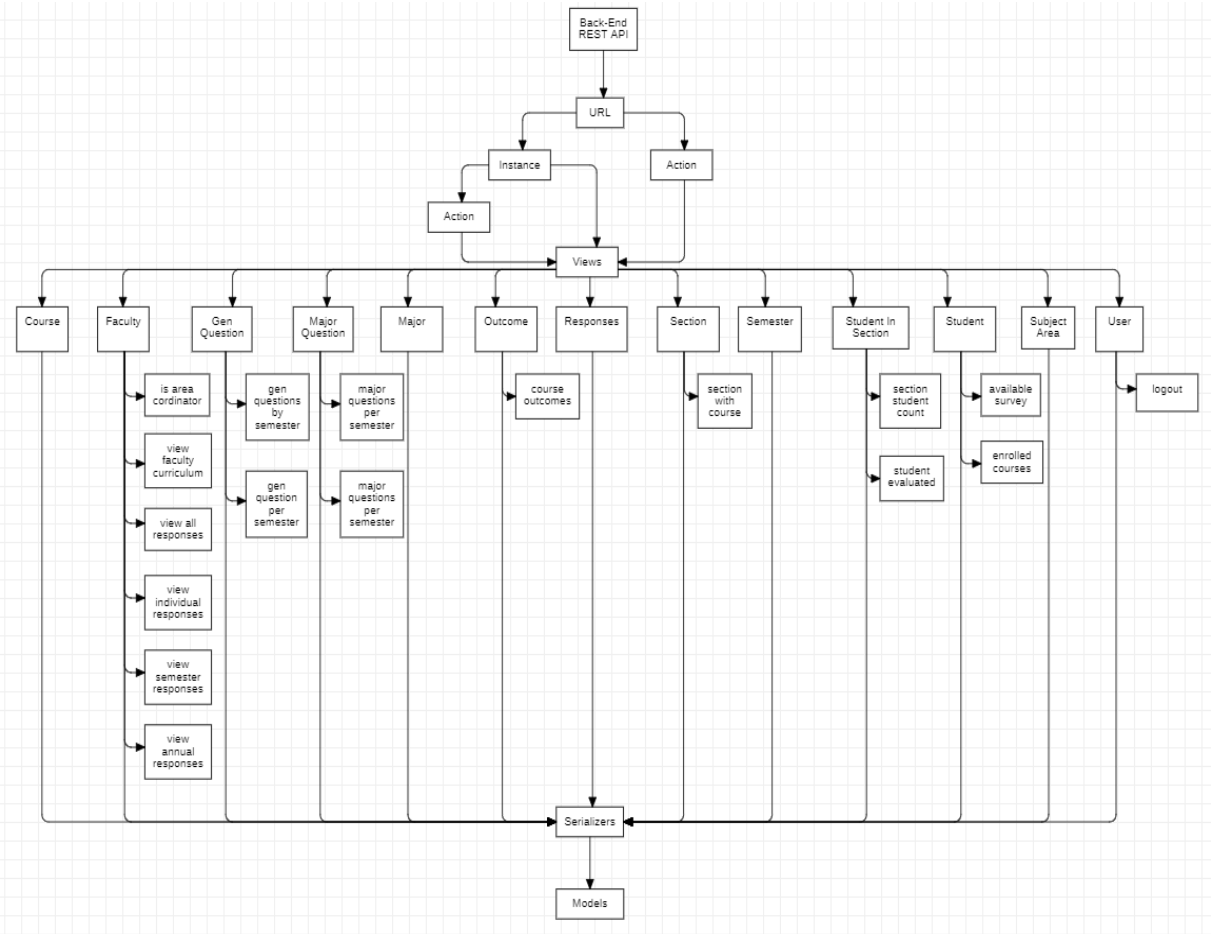
- The CLI component is simply a Python script that uses the requests package to communicate with the backend. The input to the CLI will depend on the desired command from the admin; creating faculty, courses, sections, majors and students, will require a CSV file with the proper format as input; modifying faculty, courses, and sections, require the correct string as input.

Package Diagram

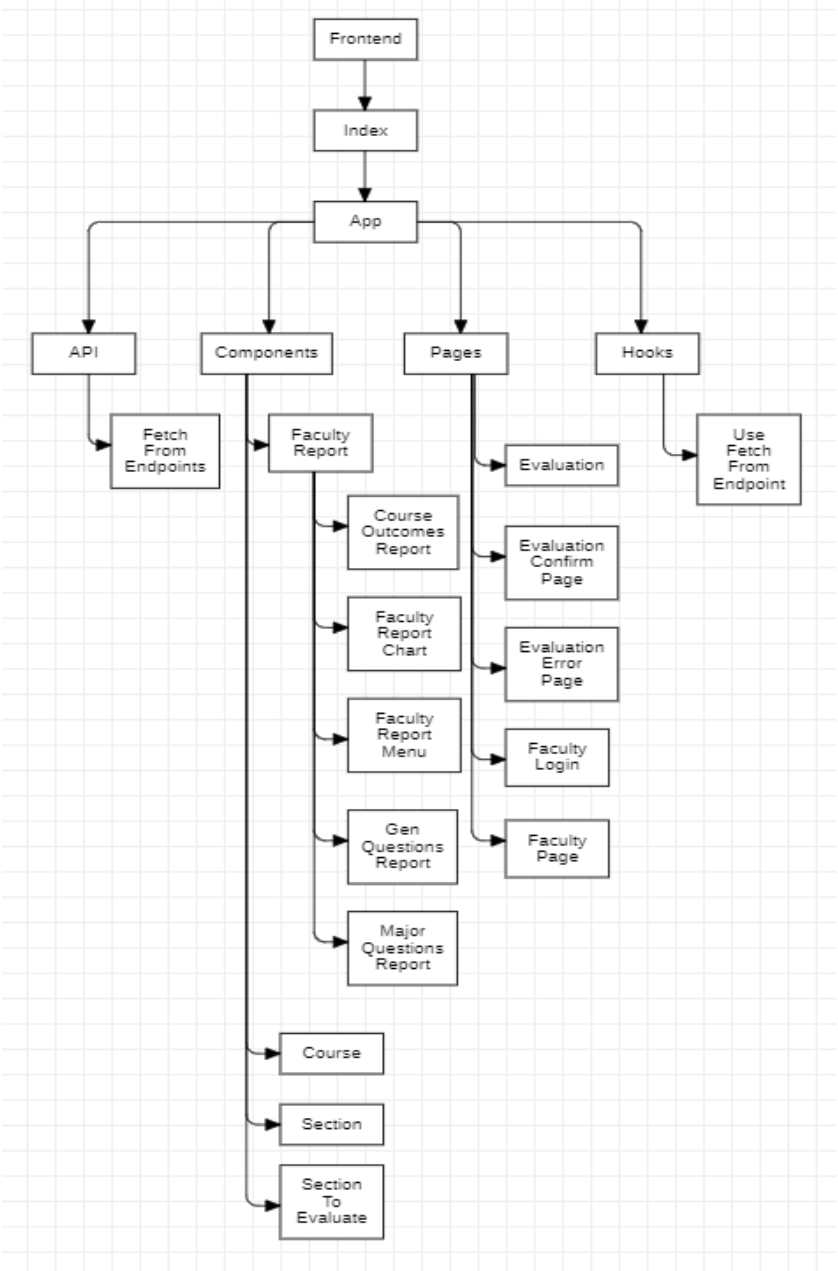


System and Subsystem Decomposition

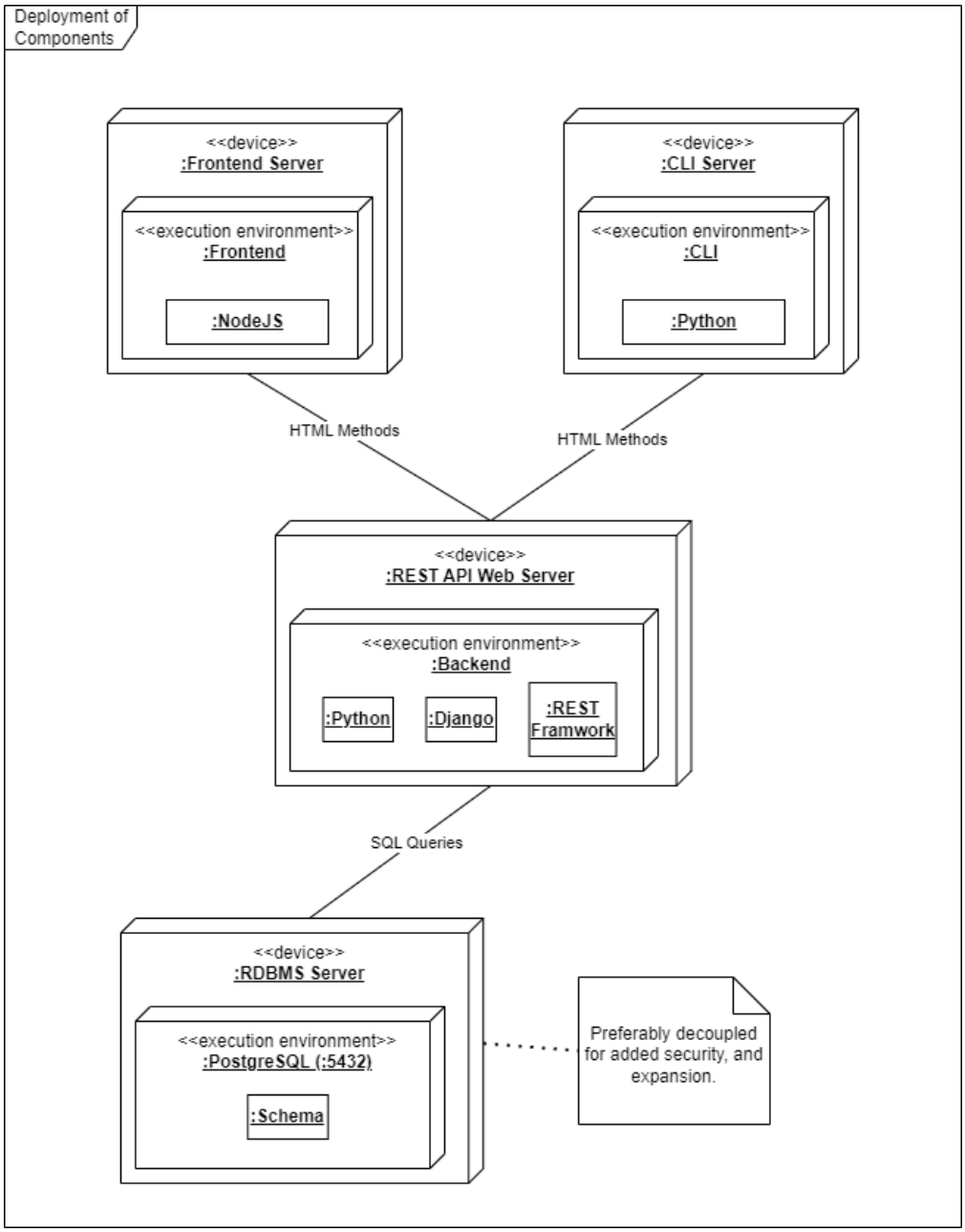
Backend



Frontend


Frontend
Index
App
API
Components
Pages
Hooks
Fetch From Endpoints
Faculty Report
Evaluation
Use Fetch From Endpoint
Course Outcomes Report
Evaluation Confirm Page
Faculty Report Chart
Evaluation Error Page
Faculty Report Menu
Faculty Login
Gen Questions Report
Faculty Page
Major Questions Report
Course
Section
Section To Evaluate

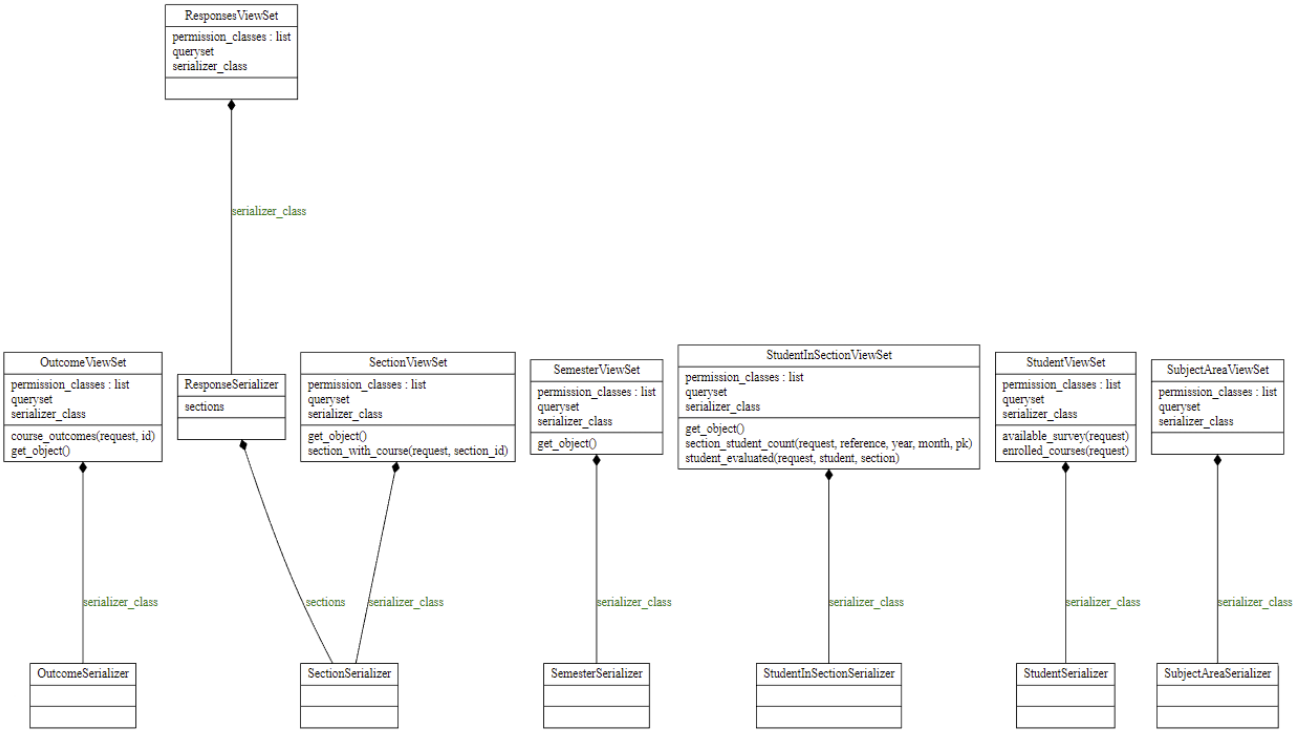
Deployment Diagram

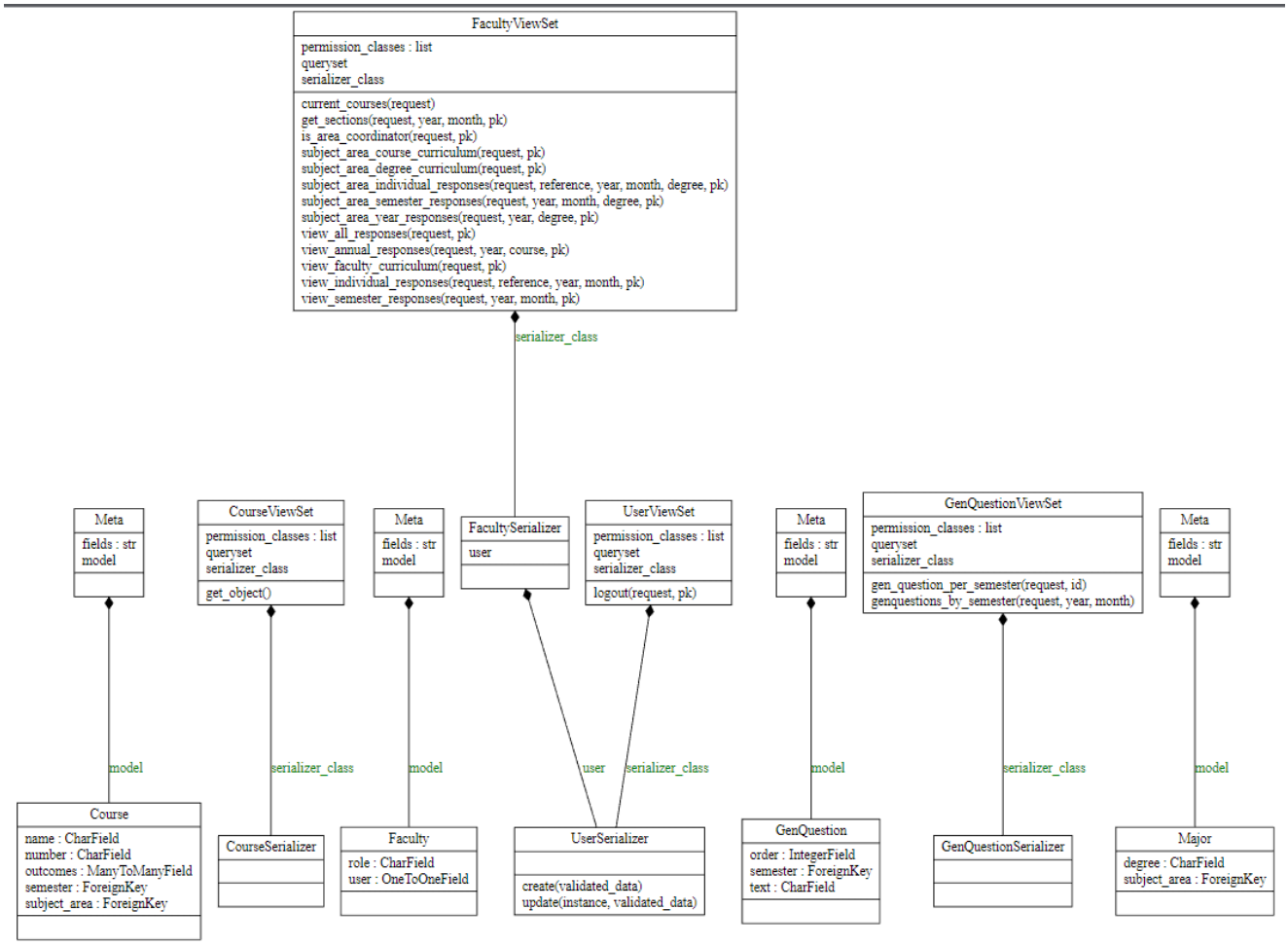


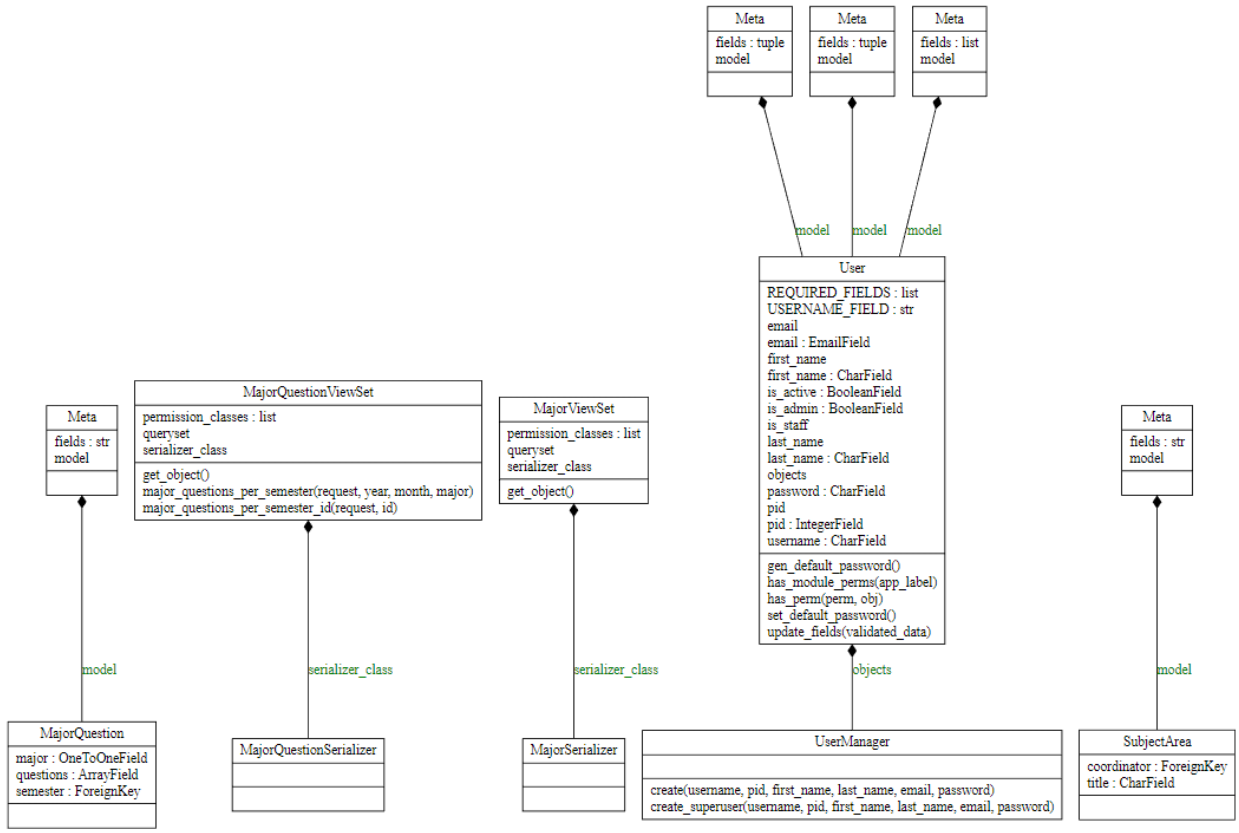
Design Patterns

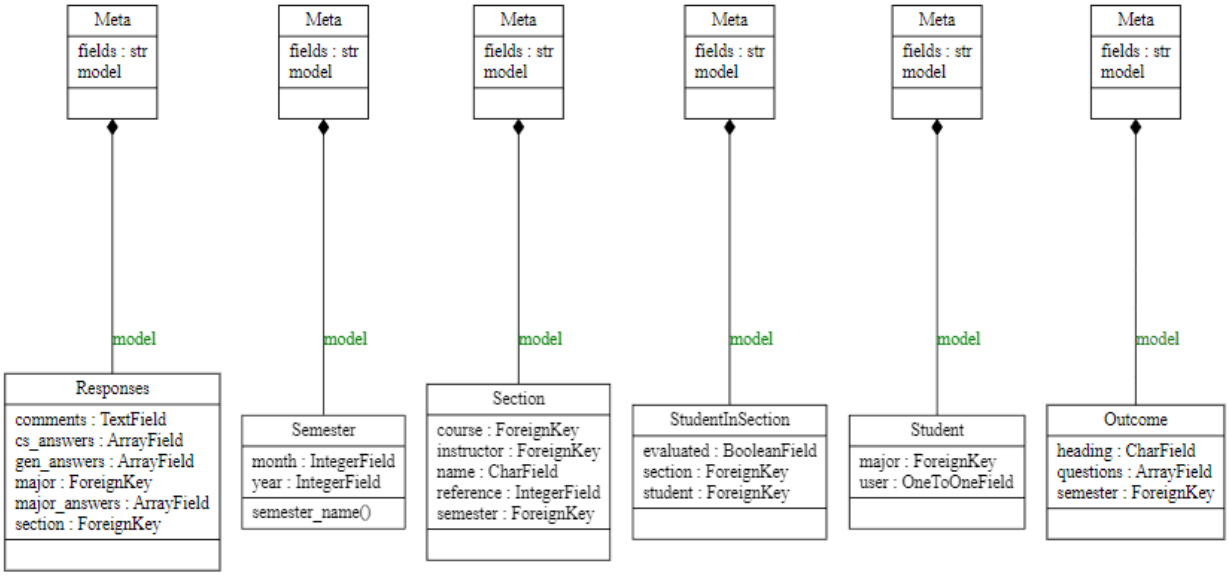
- The backend component follows the MVC architecture. Views take care of displaying and collecting data from the user or other system components. Views also provide other system components with actions in order to collect and analyze data requested, therefore they can be considered partial “Controllers”. Serializers (or “Controllers”) perform data conversion into native Python objects. Serializers also validate and sanitize this data. Models are the representation of database tables that will be used collectively by the Serializers and Views in order for the REST API to correctly generate SQL queries.
- The frontend package follows a Component architecture structure. Because we used React as our framework, functional components are the logical answer to using it. By separating individual pieces of renderable Javascript code in separate files, the framework can become more modular and flexible when adding new parts to the UX.
- The CLI is a Singleton package architecture. It is composed of an individual Python script that reacts to specific commands that will then interact with the REST API. The input to this program is in the format of a CSV file that will be of a different format depending on the command that is required.

Class Diagrams (backend)




FacultyViewSet
permission_classes : list
queryset
serializer_class
current_courses(request)
get_sections(request, year, month, pk)
is_area_coordinator(request, pk)
subject_area_course_curriculum(request, pk)
subject_area_degree_curriculum(request, pk)
subject_area_individual_responses(request, reference, year, month, degree, pk)
subject_area_semester_responses(request, year, month, degree, pk)
subject_area_year_responses(request, year, degree, pk)
view_all_responses(request, pk)
view_annual_responses(request, year, course, pk)
view_faculty_curriculum(request, pk)
view_individual_responses(request, reference, year, month, pk)
view_semester_responses(request, year, month, pk)
serializer_class
Meta
fields : str
model
CourseViewSet
permission_classes : list
queryset
serializer_class
get_object()
Meta
fields : str
model
FacultySerializer
user
UserViewSet
permission_classes : list
queryset
serializer_class
logout(request, pk)
Meta
fields : str
model
GenQuestionViewSet
permission_classes : list
queryset
serializer_class
gen_question_per_semester(request, id)
genquestions_by_semester(request, year, month)
Meta
fields : str
model
model
serializer_class
model
user
serializer_class
model
serializer_class
model
Course
name : CharField
number : CharField
outcomes : ManyToManyField
semester : ForeignKey
subject_area : ForeignKey
CourseSerializer
Faculty
role : CharField
user : OneToOneField
UserSerializer
create(validated_data)
update(instance, validated_data)
GenQuestion
order : IntegerField
semester : ForeignKey
text : CharField
GenQuestionSerializer
Major
degree : CharField
subject_area : ForeignKey


Meta
fields : tuple
model
Meta
fields : tuple
model
Meta
fields : list
model
model
model
model
User
REQUIRED_FIELDS : list
USERNAME_FIELD : str
email
email : EmailField
first_name
first_name : CharField
is_active : BooleanField
is_admin : BooleanField
is_staff
last_name
last_name : CharField
objects
password : CharField
pid
pid : IntegerField
username : CharField
gen_default_password()
has_module_perms(app_label)
has_perm(perm, obj)
set_default_password()
update_fields(validated_data)
Meta
fields : str
model
MajorQuestionViewSet
permission_classes : list
queryset
serializer_class
get_object()
major_questions_per_semester(request, year, month, major)
major_questions_per_semester_id(request, id)
MajorViewSet
permission_classes : list
queryset
serializer_class
get_object()
Meta
fields : str
model
model
serializer_class
serializer_class
objects
model
MajorQuestion
major : OneToOneField
questions : ArrayField
semester : ForeignKey
MajorQuestionSerializer
MajorSerializer
UserManager
create(username, pid, first_name, last_name, email, password)
create_superuser(username, pid, first_name, last_name, email, password)
SubjectArea
coordinator : ForeignKey
title : CharField


Meta
fields : str
model
model
Responses
comments : TextField
cs_answers : ArrayField
gen_answers : ArrayField
major : ForeignKey
major_answers : ArrayField
section : ForeignKey
Meta
fields : str
model
model
Semester
month : IntegerField
year : IntegerField
semester_name()
Meta
fields : str
model
model
Section
course : ForeignKey
instructor : ForeignKey
name : CharField
reference : IntegerField
semester : ForeignKey
Meta
fields : str
model
model
StudentInSection
evaluated : BooleanField
section : ForeignKey
student : ForeignKey
Meta
fields : str
model
model
Student
major : ForeignKey
user : OneToOneField
Meta
fields : str
model
model
Outcome
heading : CharField
questions : ArrayField
semester : ForeignKey

System Validation

In this section, test cases for all developed features are presented.

Capstone 2 Development

Test CES GUI for Students

Test Case ID: Display_Login_Student

- **Purpose:** To display CES GUI login page.
- **Setup:** Activate virtual environment, run server and frontend, and access localhost:3000 on a web browser.
- **Input:** Home.js file containing the basic GUI.
- **Expected Output:** Login GUI on the localhost:3000 server.
- **Status**: Pass

Test Case ID: Authenticate_Login_Student

- **Purpose:** To authenticate a student through CES GUI login page.
- **Setup:** Activate virtual environment, run server and frontend, and access localhost:3000 on a web browser.
- **Input:**
  username: *teststudent1*
  password: *password1*
- **Expected Output:** Http status response 200**.**
- **Status**: Pass

Test Case ID: Display_StudentPage_Student

- **Purpose:** To display CES GUI student page.
- **Setup:** Activate virtual environment, run server and frontend, access localhost:3000 on a web browser, enter valid credentials for a student, and click login button.

- **Input:**
  username: *teststudent1*
  password: *password1*
- **Expected Output:** StudentPage GUI on the address *localhost:3000/user* with all evaluations rendered.
- **Status**: Pass

Test Case ID: StudentPage_Select_Evaluation

- **Purpose:** To select an evaluation in StudentPage and redirect to EvaluationPage successfully.
- **Setup:** Activate virtual environment, run server and frontend, access localhost:3000 on a web browser, enter valid credentials for a student, and click login button to be redirected to StudentPage.
- **Input:** StudentPage.js file containing the GUI and logic.
- **Expected Output:** EvaluationPage GUI on the address *localhost:3000/evaluation*.
- **Status**: Pass

Test Case ID: EvaluationPage_Submit_CorrectForm

- **Purpose:** To fill evaluation form in EvaluationPage and redirect to ConfirmationPage successfully.
- **Setup:** Activate virtual environment, run server and frontend, access localhost:3000 on a web browser, enter valid credentials for a student, click login button to be redirected to StudentPage, select an evaluation to be redirected to EvaluationPage, and answer all questions.
- **Input:** Evaluation.js file containing the GUI and logic.
- **Expected Output:** ConfirmationPage GUI on the address *localhost:3000/evaluation/confirm* and response entered in database.
- **Status**: Pass

Test Case ID:
EvaluationPage_Submit_MissingFields

- **Purpose:** To fill evaluation form in EvaluationPage and submit it with missed fields.
- **Setup:** Activate virtual environment, run server and frontend, access localhost:3000 on a web browser, enter valid credentials for a student, click login button to be redirected to StudentPage, select an evaluation to be redirected to EvaluationPage, answer some questions, and leave at least one question (radio button) unanswered.
- **Input:** Evaluation.js file containing the GUI and logic.
- **Expected Output:** Error message indicating all missing questions in the same page.
- **Status**: Pass

Test Case ID: EvaluationPage_Submit_Error

- **Purpose:** To fill evaluation form in EvaluationPage and submit it, being the submission request unsuccessful.
- **Setup:** Activate virtual environment, run server and frontend, access localhost:3000 on a web browser, enter valid credentials for a student, click login button to be redirected to StudentPage, select an evaluation to be redirected to EvaluationPage, answer all questions and submit.
- **Input:** Simulate a different code than 201 in Evaluation.js.
- **Expected Output:** EvaluationErrorPage GUI on the address *localhost:3000/evaluation/error*.
- **Status**: Pass

Test CES API

Test Case ID: Test_All_CES_API

- **Purpose:** To test all CES API calls, since some API calls are required before testing GUI features.

- **Setup:** Activate virtual environment, run server, and access localhost:8000 on a web browser. Choose all GET API endpoints and call them. Choose all POST APIs endpoints and call them. Choose all PATCH APIs endpoints and call them. Choose all DELETE APIs endpoints and call them.
- **Input:** JSON files for GET, POST, PATCH, DELETE calls.
- **Expected Output:** Respective successful code response on each API call
- **Status**: Pass

Test CES GUI for Faculty

Test Case ID: Display_Login_Faculty

- **Purpose:** To display CES GUI login page for faculty.
- **Setup:** Activate virtual environment, run server and frontend, and access localhost:3000/faculty/login on a web browser.
- **Input:** FacultyLogin.js file containing the basic GUI.
- **Expected Output:** Login GUI on the localhost:3000/faculty/login address.
- **Status**: Pass

Test Case ID: Authenticate_Login_Faculty

- **Purpose:** To authenticate a faculty through CES GUI login page for faculty.
- **Setup:** Activate virtual environment, run server and frontend, and access localhost:3000/faculty/login on a web browser.
- **Input:**
  username: *testfaculty1*
  password: *password1*
- **Expected Output:** Http status response 200**.**
- **Status**: Pass

Test Case ID: Display_StudentPage_Student

- **Purpose:** To display CES GUI faculty page.

- **Setup:** Activate virtual environment, run server and frontend, access localhost:3000/faculty/login on a web browser, enter valid credentials for a faculty, and click login button.
- **Input:**
  username: *testfaculty1*
  password: *password1*
- **Expected Output:** FacultyPage GUI on the address *localhost:3000/faculty*.
- **Status**: Pass

Test Case ID: Faculty_Get_Report

- **Purpose:** To get a report from the faculty page by using the filters.
- **Setup:** Activate virtual environment, run server and frontend, access localhost:3000/faculty/login on a web browser, enter valid credentials for a faculty, click login button, and filter a report.
- **Input:** FacultyPage.js file containing the basic GUI.
- **Expected Output:** FacultyPage GUI on the address *localhost:3000/faculty* with report generated for selected filters.
- **Status**: Pass

Test CES CLI for Admin

Test Case ID: Add_Faculty

- **Purpose:** To enter a faculty through the CLI
- **Setup:** Activate virtual environment, run server, and run CLI with “ias” command [`python3 ces.py iaf`].
- **Input:**
  username: *testfaculty*
  pid: *1234567*
  first_name: *John*
  last_name: *Paul*
  password: *password1*

role: *Instructor*

- **Expected Output:** Faculty data entered successfully in the database.
- **Status**: Pass

Test Case ID: Add_Subject_Area

- **Purpose:** To enter a subject area through the CLI
- **Setup:** Activate virtual environment, run server, and run CLI with "iasa" command [`python3 ces.py iafa`].
- **Input:**
  title: *Computer Science*
  cordinator: *testfaculty*
- **Expected Output:** Subject Area data entered successfully in the database.
- **Status**: Pass

Test Case ID: Add_Courses

- **Purpose:** To enter courses through the CLI with a csv file.
- **Setup:** Activate virtual environment, run server, and run CLI with "bac" command [`python3 ces.py bac`].
- **Input:** CSV file
- **Expected Output:** Courses data entered successfully in the database.
- **Status**: Pass

Test Case ID: Add_Students

- **Purpose:** To enter students through the CLI with a csv file.
- **Setup:** Activate virtual environment, run server, and run CLI with "bas" command [`python3 ces.py bas`].
- **Input:** CSV file
- **Expected Output:** Students data entered successfully in the database.
- **Status**: Pass

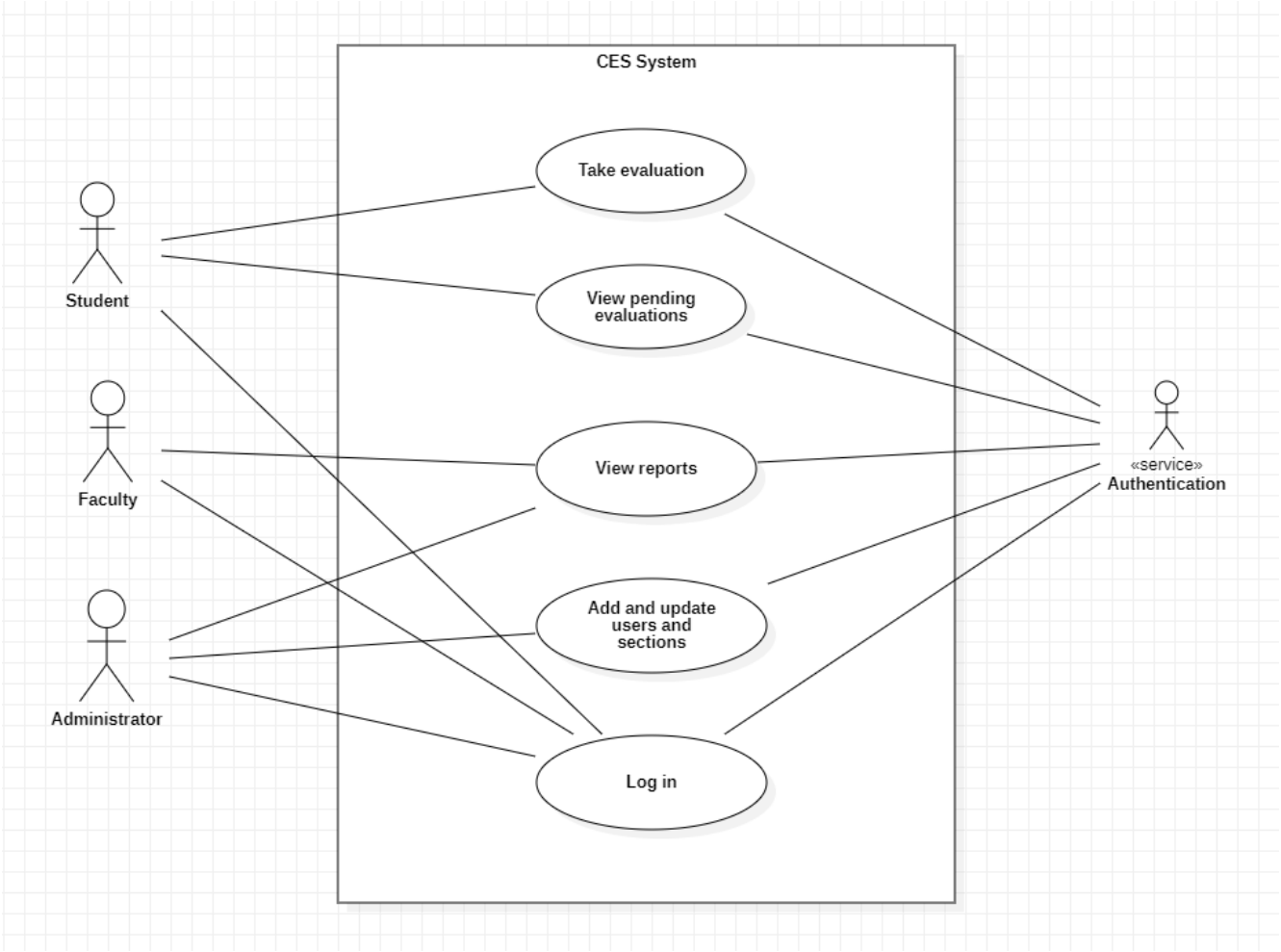
Glossary

Appendix

This appendix contains the UML diagrams generated for our system.

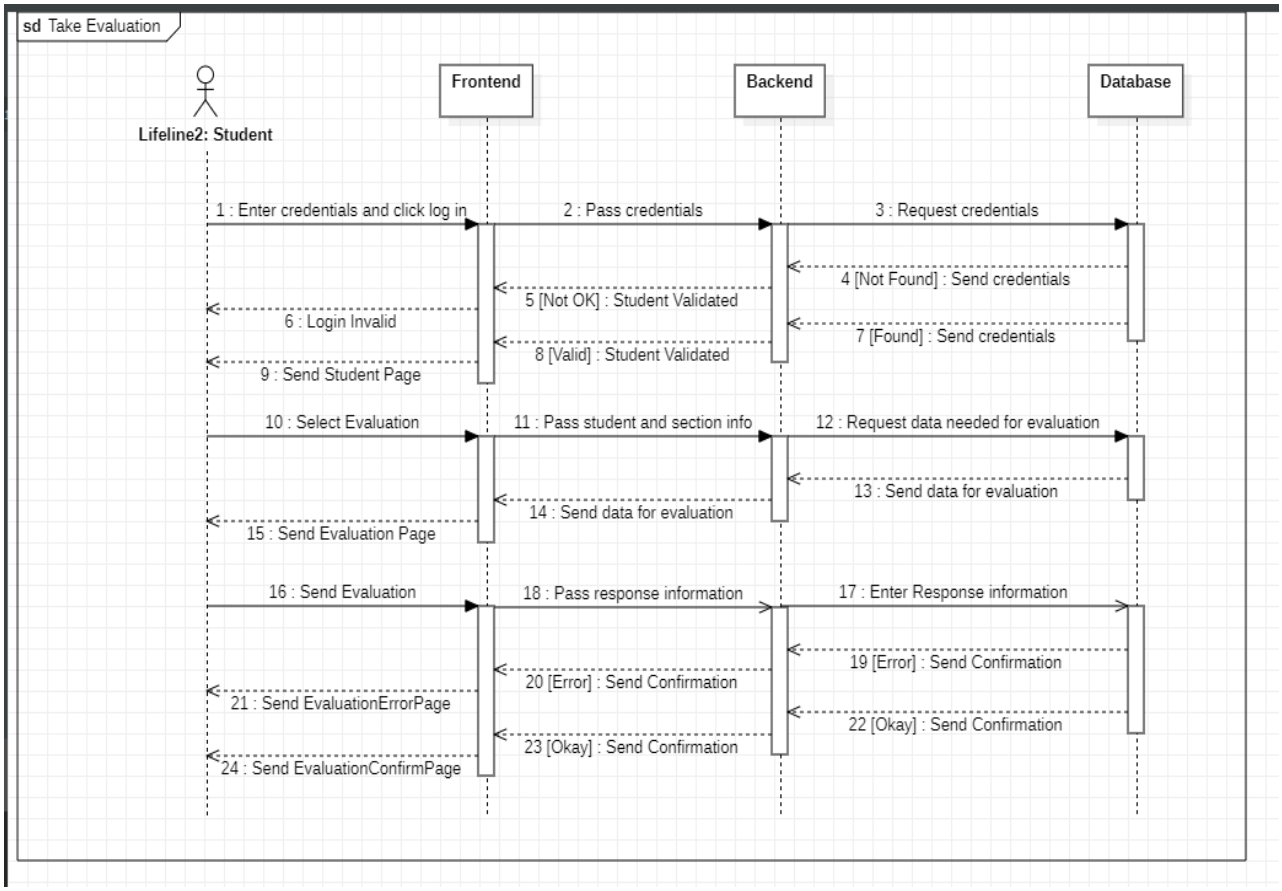
Appendix A - UML Diagrams

1. Use Case Diagram

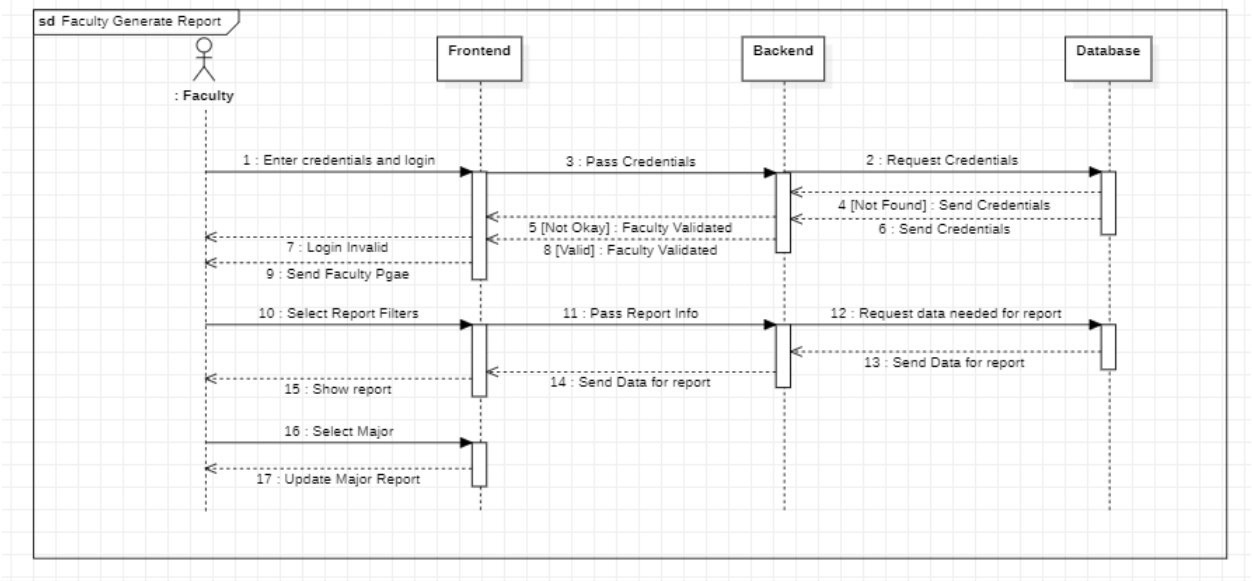


Use Case Diagram

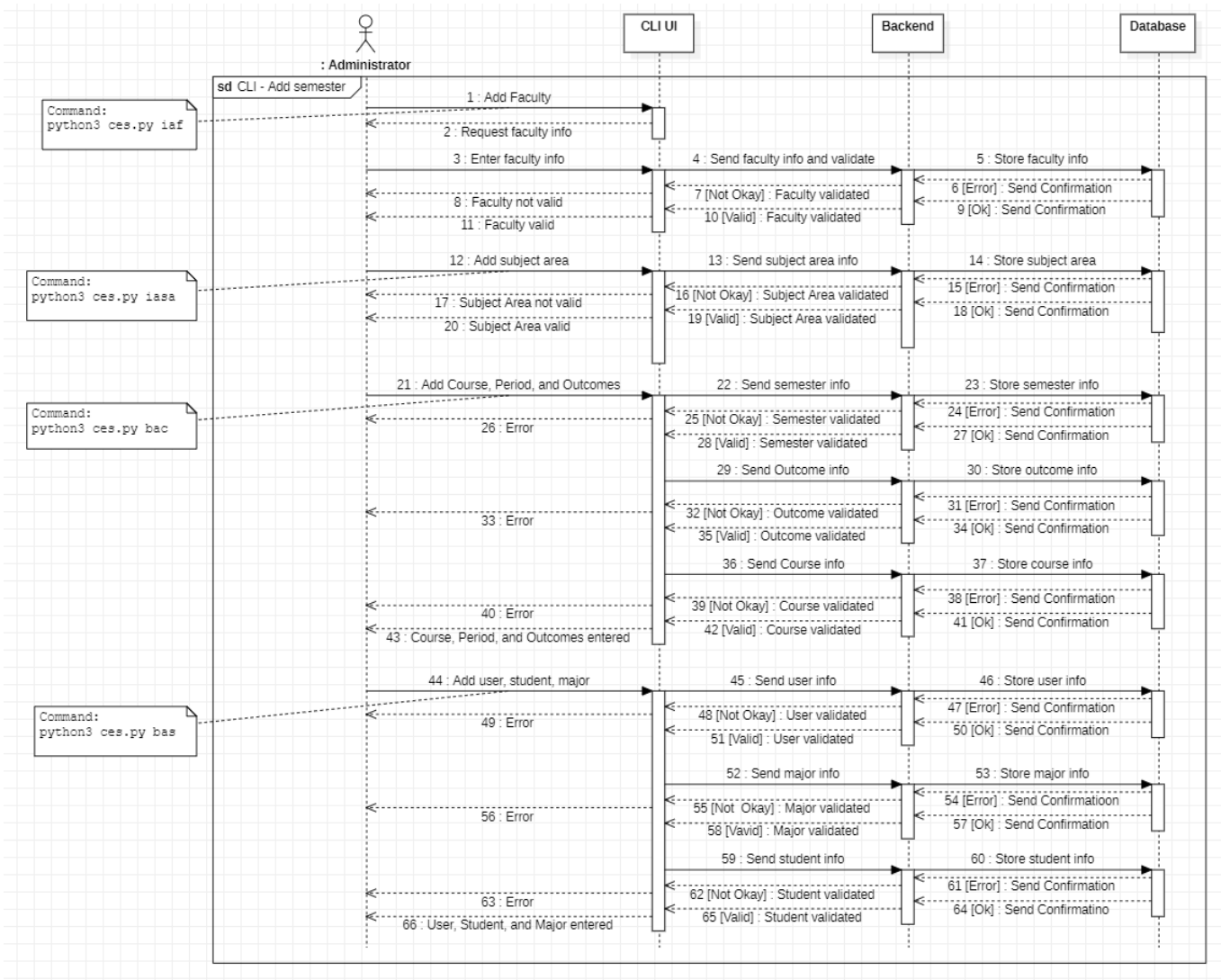
2. Sequence Diagram - Take Evaluation [Student]



3. Sequence Diagram - Retrieve Report [Faculty]

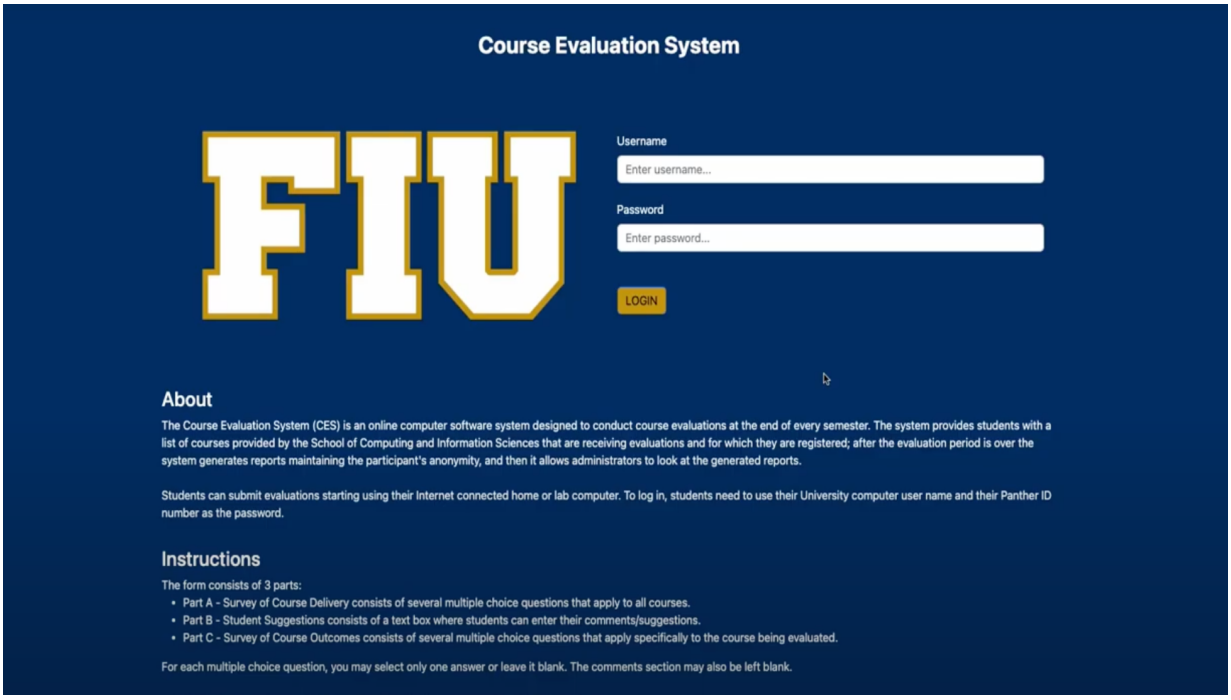


4. Sequence Diagram - Enter a semester [Admin]

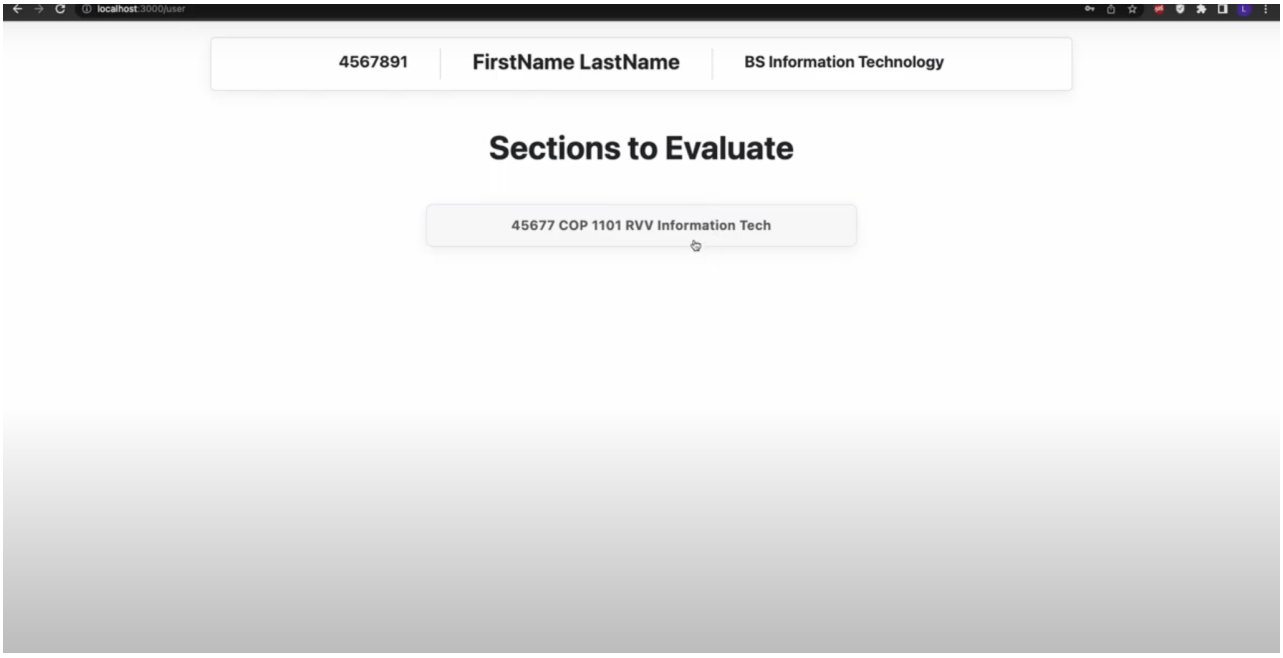


Appendix B - User Interface

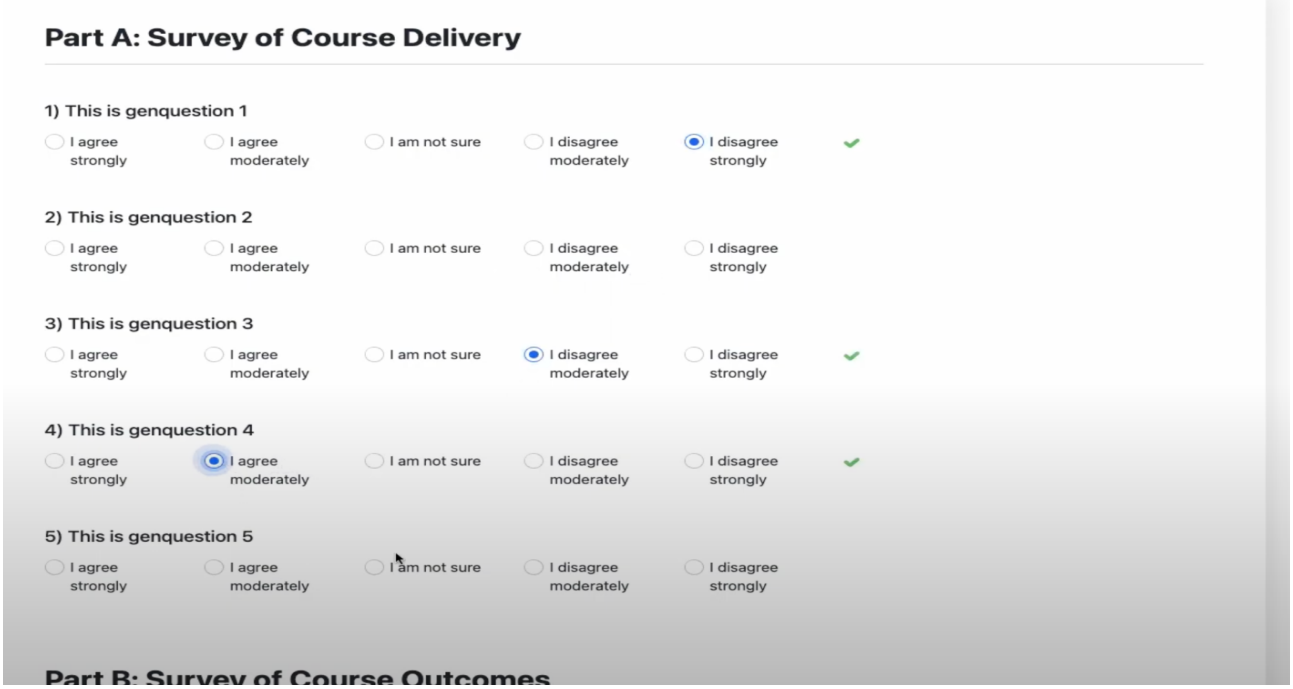
Login



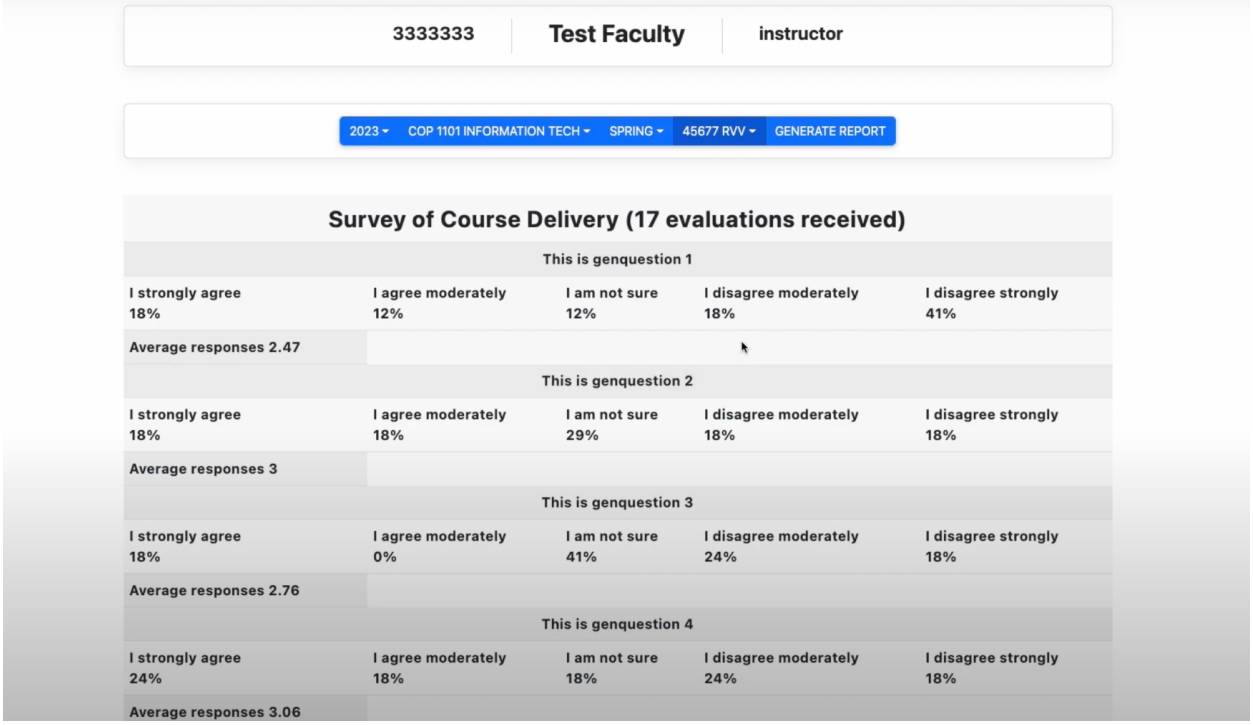
Student


localhost:3000/user
4567891
FirstName LastName
BS Information Technology
Sections to Evaluate
45677 COP 1101 RVV Information Tech

Evaluation Page



Faculty Reports



Appendix C - Sprint Review Reports

This section contains the sprints reviews and retrospectives logs for both Capstone 1 and 2 development periods.

Capstone 1 and 2 Development

Sprint 1

Review - Sprint 1
Date: September 9th, 2022
Time: 8:00pm
Participants: Artur Vorojeykin, Gerardo Parra, Amani Hunt, Jose Milanes Urquiza, Daniel Nunez Dominguez, Mariela Torres Zuniga, Shaoting Wu, Abel Aguilera, Gabriel Campillo, Saul Castillo, Ruben Cerna Serna, Schadrack Cineas, Fernando Collante, Blake Prieto, Ryan Rodriguez

Sprint Goals:

- Understand the SCRUM process.
- Figuring out the best time to meet every week.
- Assign roles and responsibilities.
- Start a testing API to start learning about the REST Framework.
- Set up communication channels.
- Create documentation for the different tools we will be using.

Completed Tasks:

- ~~Understand the SCRUM process.~~
- ~~Figuring out the best time to meet every week.~~
- Assign roles and responsibilities.
- ~~Start a testing API to start learning about the REST Framework.~~
- ~~Set up communication channels.~~
- ~~Create documentation for the different tools we will be using.~~

Challenges:

- Inability to split the work because we did not have access to some of the requirements for the initial database.
- Coordinate a time where the most members could be present for meetings.

Successes:

- Able to demo an initial mock REST API to the PO and got his approval to continue using the tools we selected.
- Everyone was able to successfully set up their development environment.

Retrospective - Sprint 1
Date: September 9th, 2022
Time: 8:00pm
Participants: Artur Vorojeykin, Gerardo Parra, Amani Hunt, Jose Milanes Urquiza, Daniel Nunez Dominguez, Mariela Torres Zuniga, Shaoting Wu, Abel Aguilera, Gabriel Campillo, Saul Castillo, Ruben Cerna Serna, Schadrack Cineas, Fernando Collante, Blake Prieto, Ryan Rodriguez

Completed/Approved User Stories:

- ~~Created Documentation for setting up the development environment.~~
- ~~Created templates for the required documents on the upcoming Sprints.~~
- ~~Successfully setup the development environment for team members.~~
- ~~Created a Trello board to keep track of the due dates and development progress.~~
- ~~Successfully gathered required database schema and requirements from the PO.~~
- ~~Created all communication channels for the semester.~~

What went well:

- We were able to set up the development environment and get some practice with the tools we will be using for development.

- We were able to create a testing API for learning and use different methods of interacting with it.
- We were able to demonstrate to get approval from the PO on the different tools we will be using throughout the semester.

What did not go so well:

- Trying to find a date for meetings that would accommodate the most members.
- Trying to split the work because we had not received the correct input fields for the initial database.

Sprint 2

Review - Sprint 2
Date: September 23rd, 2022
Time: 8:00pm
Participants: Artur Vorojeykin, Gerardo Parra, Amani Hunt, Jose Milanes Urquiza, Daniel Nunez Dominguez, Mariela Torres Zuniga, Shaoting Wu, Abel Aguilera, Gabriel Campillo, Saul Castillo, Ruben Cerna Serna, Schadrack Cineas, Fernando Collante, Blake Prieto, Ryan Rodriguez

Completed Goals:

- ~~Database Design~~
    - Minimum required models.
- ~~Serializers and Viewsets~~
    - Minimum required models.

Feedback from PO:

- Rework the Section model with an appropriate Foreign Key to a User model.
- Potentially add another model for grouping courses by areas.
- Add a role called “area coordinator” to further filter courses.

Backlog:

1. Improving Models.
2. Authentication.
3. Database Actions.

Retrospective - Sprint 2
Date: September 23rd, 2022
Time: 8:00pm
Participants: Artur Vorojeykin, Gerardo Parra, Amani Hunt, Jose Milanes Urquiza, Daniel Nunez Dominguez, Mariela Torres Zuniga, Shaoting Wu, Abel Aguilera, Gabriel Campillo, Saul Castillo, Ruben Cerna Serna, Schadrack Cineas, Fernando Collante, Blake Prieto, Ryan Rodriguez

What went well:

- Able to adapt to new changes introduced by different branches.
- Hit the due date for all assigned stories.
- Great teamwork.

Needs to change:

- Better communication.
- Set up more accurate user stories for development.
- Spread the work more evenly.
- Have more reasonable deadlines.

Concerns:

- Not having clear requirements from the PO.

Sprint 3

Review - Sprint 3
Date: October 7th, 2022
Time: 8:00pm
Participants: Artur Vorojeykin, Gerardo Parra, Amani Hunt, Jose Milanes Urquiza, Daniel Nunez Dominguez, Mariela Torres Zuniga, Shaoting Wu, Abel Aguilera, Gabriel Campillo, Saul Castillo, Ruben Cerna Serna, Schadrack Cineas, Fernando Collante, Blake Prieto, Ryan Rodriguez

Completed Goals:

- ~~New schema based on new requirements.~~

Feedback from PO:

- Full freedom to create a new schema.
- Add a role called “area coordinator” to further filter courses.

Backlog:

1. Authentication.
2. Actions.

Retrospective - Sprint 3
Date: October 7th, 2022
Time: 8:00pm
Participants: Artur Vorojeykin, Gerardo Parra, Amani Hunt, Jose Milanes Urquiza, Daniel Nunez Dominguez, Mariela Torres Zuniga, Shaoting Wu, Abel Aguilera, Gabriel Campillo, Saul Castillo, Ruben Cerna Serna, Schadrack Cineas, Fernando Collante, Blake Prieto, Ryan Rodriguez

What went well:

- Able to create a new schema quickly given the new details from PO and Eric Jonhson.

Needs to change:

- More communication between members.
- Being able to gather requirements more efficiently.

Concerns:

- Not having clear requirements from the PO.
- Not having enough time to complete the assigned user stories because of a change in requirements

Sprint 4

Review - Sprint 4
Date: October 21st, 2022
Time: 8:00pm
Participants: Artur Vorojeykin, Gerardo Parra, Amani Hunt, Jose Milanes Urquiza, Daniel Nunez Dominguez, Mariela Torres Zuniga, Shaoting Wu, Abel Aguilera, Gabriel Campillo, Saul Castillo, Ruben Cerna Serna, Schadrack Cineas, Fernando Collante, Blake Prieto, Ryan Rodriguez

Completed Goals:

- ~~Login page~~
- ~~Faculty page~~
- ~~Evaluation page~~
- ~~Token-based authentication~~

- ~~Action for faculty and students~~

Feedback from PO:

- Everything looked on time and okay.

Backlog:

1. Report gathering.
2. Extra actions.

Retrospective - Sprint 4
Date: October 21st, 2022
Time: 8:00pm
Participants: Artur Vorojeykin, Gerardo Parra, Amani Hunt, Jose Milanes Urquiza, Daniel Nunez Dominguez, Mariela Torres Zuniga, Shaoting Wu, Abel Aguilera, Gabriel Campillo, Saul Castillo, Ruben Cerna Serna, Schadrack Cineas, Fernando Collante, Blake Prieto, Ryan Rodriguez

What went well:

- Great communication.
- Good progress between teammates.

Needs to change:

- Work faster.
- Being able to gather requirements more efficiently.

Concerns:

- Not having enough time to finish all the requirements.
- Not being able to implement extra requirements from the PO.

Sprint 5

Review - Sprint 5
Date: November 6th, 2022
Time: 8:00pm
Participants: Artur Vorojeykin, Gerardo Parra, Amani Hunt, Jose Milanes Urquiza, Daniel Nunez Dominguez, Mariela Torres Zuniga, Shaoting Wu, Abel Aguilera, Gabriel Campillo, Saul Castillo, Ruben Cerna Serna, Schadrack Cineas, Fernando Collante, Blake Prieto, Ryan Rodriguez

Completed Goals:

- ~~Faculty Login.~~
- ~~Faculty Yearly, Semesterly and Individual report.~~
- ~~Readme.txt and Installation manual.~~

Feedback from PO:

- Improve the CLI

Backlog:

1. Report gathering.
2. Extra actions.

Retrospective - Sprint 5
Date: November 6th, 2022
Time: 8:00pm

Participants: Artur Vorojeykin, Gerardo Parra, Amani Hunt, Jose Milanes Urquiza, Daniel Nunez Dominguez, Mariela Torres Zuniga, Shaoting Wu, Abel Aguilera, Gabriel Campillo, Saul Castillo, Ruben Cerna Serna, Schadrack Cineas, Fernando Collante, Blake Prieto, Ryan Rodriguez

What went well:

- Great communication.
- Good progress between teammates.
- Good progress on the frontend.

Needs to change:

- Work together to gather all documents for the final deliverable.
- Contribute with each other's portions for the final deliverable.

Concerns:

- Not having enough time to finish the essential parts of the frontend.
- Not having enough time to finish the final deliverable.

Sprint 6

Review - Sprint 6
Date: November 20th, 2022
Time: 8:00pm
Participants: Artur Vorojeykin, Gerardo Parra, Amani Hunt, Jose Milanes Urquiza, Daniel Nunez Dominguez, Mariela Torres Zuniga, Shaoting Wu, Abel Aguilera, Gabriel Campillo, Saul Castillo, Schadrack Cineas, Blake Prieto, Ryan Rodriguez

Completed Goals:

- ~~Frontend (student and faculty pages).~~
- ~~Backend (actions).~~
- ~~Minimum CLI setup.~~

Feedback from PO:

- Improve the CLI
- Deploy the project to the VPS provided by Eric.

Backlog:

1. Subject Area Coordinator reports.
2. CLI commands.
3. Bug fixes in the frontend for Faculty Page.

Retrospective - Sprint 6
Date: November 20th, 2022
Time: 8:00pm
Participants: Artur Vorojeykin, Gerardo Parra, Amani Hunt, Jose Milanes Urquiza, Daniel Nunez Dominguez, Mariela Torres Zuniga, Shaoting Wu, Abel Aguilera, Gabriel Campillo, Saul Castillo, Schadrack Cineas, Blake Prieto, Ryan Rodriguez

What went well:

- Great communication.
- Were able to finish the frontend and backend with the minimum requirements from the PO.

- Were able to create some test cases for the frontend.
- Were able to start creating some of the documents for the final deliverable.

Needs to change:

- More coordination between members in order to accomplish the requirements faster.

Concerns:

- Not having enough time to finish the demo videos and posters.
- Not having enough time in general to finish everything that is required.

Sprint 7

Retrospective - Sprint 7
Date: November 29th, 2022
Time: 7:00pm
Participants: Artur Vorojeykin, Gerardo Parra, Amani Hunt, Jose Milanes Urquiza, Daniel Nunez Dominguez, Mariela Torres Zuniga, Shaoting Wu, Abel Aguilera, Gabriel Campillo, Saul Castillo, Schadrack Cineas, Blake Prieto, Ryan Rodriguez

What went well:

- Great communication.
- Were able to finish the Documentation, Posters, Videos and Presentation

Needs to change:

- Better time management in order to have time to test and proofread documents.

Concerns:

- Not having enough time to correct bugs in the code.
- Not having a full availability for the showcase.

Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

User Manual V1.0

Gerardo Parra - gparr014@fiu.edu
Fall - 2022

Background

The current implementation of the Course Evaluation System consists of three components, the backend, frontend and CLI. The frontend and CLI are very flexible because the backend is an implementation of a RESTful API that uses JSON-formatted HTTP requests (GET, PUT, PATCH, POST, OPTIONS) to access, create, and update entries in a PostgreSQL database. Therefore, this document will mainly focus on the features, input and output, implemented and required by the backend API.
Screenshots will be provided as needed.

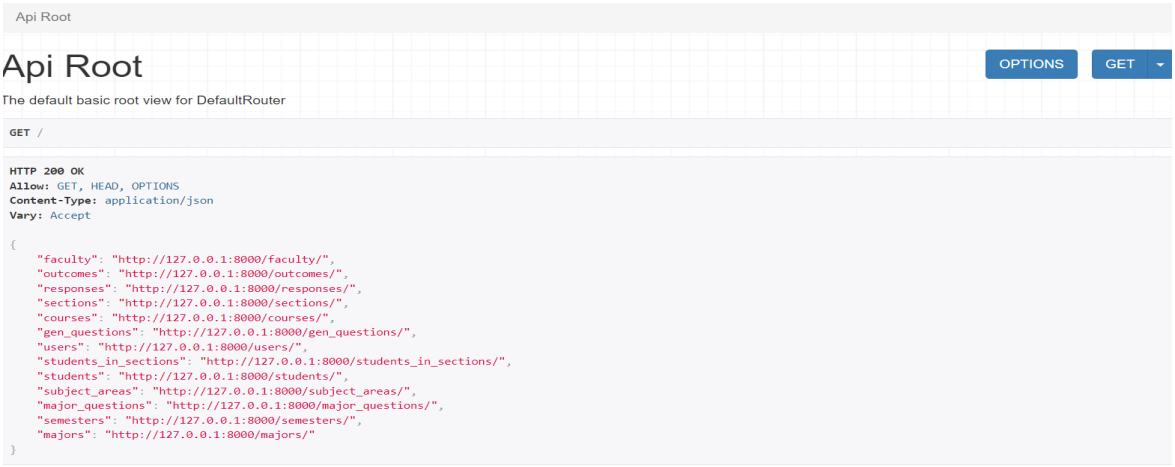
Getting Started

If the instructions of the Installation Manual were followed, the backend REST API should be ready to use. Running ***python manage.py runserver*** in the backend root folder starts the default Django web-server (caution, do not use this web-server for production, it is recommended to run

web apps such as this one behind a proxy server such as Apache or Nginx in order to have access to advanced features such as load balancing, caching, and more robust security).

API root

Navigating to ***http://localhost:8000*** provides access to the root of the REST API.



The API generates URLs for all the tables that are available in the database. Each of these endpoints can be accessed (assuming the token authentication system was disabled for testing - refer to “Token Authentication” on the Usage section to learn how to control authentication for testing).

Faculty Endpoint example

If we would like to check out the entries that exist on the faculty table, simply click on the ***"faculty": "http://127.0.0.1:8000/faculty/"*** link. This will navigate to the root of the faculty endpoint which contains:

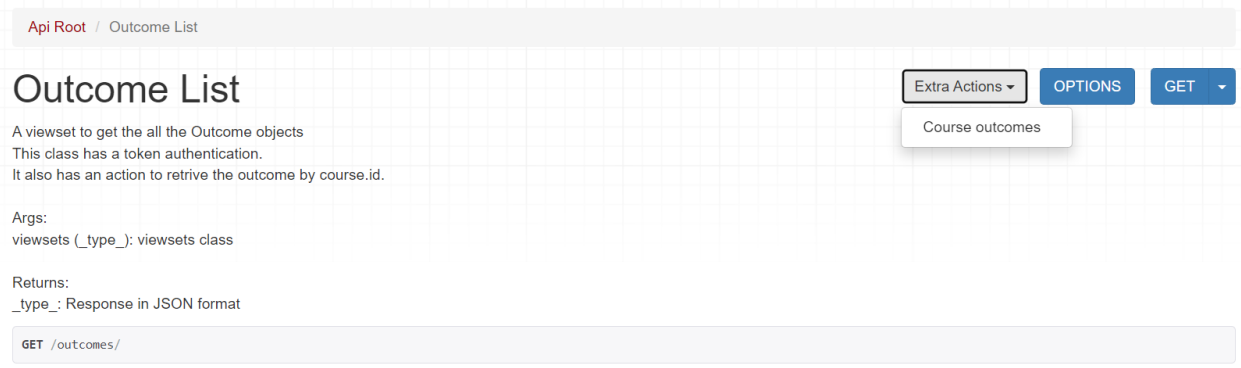
1. A JSON object which contains
    a. “count” - total count of objects present in *faculty.*
    b. “next” - a link to the next view (or null if not present)
    c. “previous” - a link to the previous view (or null if not present)
    d. “results” - an array of the objects present in *faculty*

2. An HTML Form view which allows the user to POST a new object to the *faculty* table given the required fields.
3. (optional) A Raw data button which allows the user to POST a new object using a different encoding and the ability to manually create the JSON object that will be POSTed to the table.

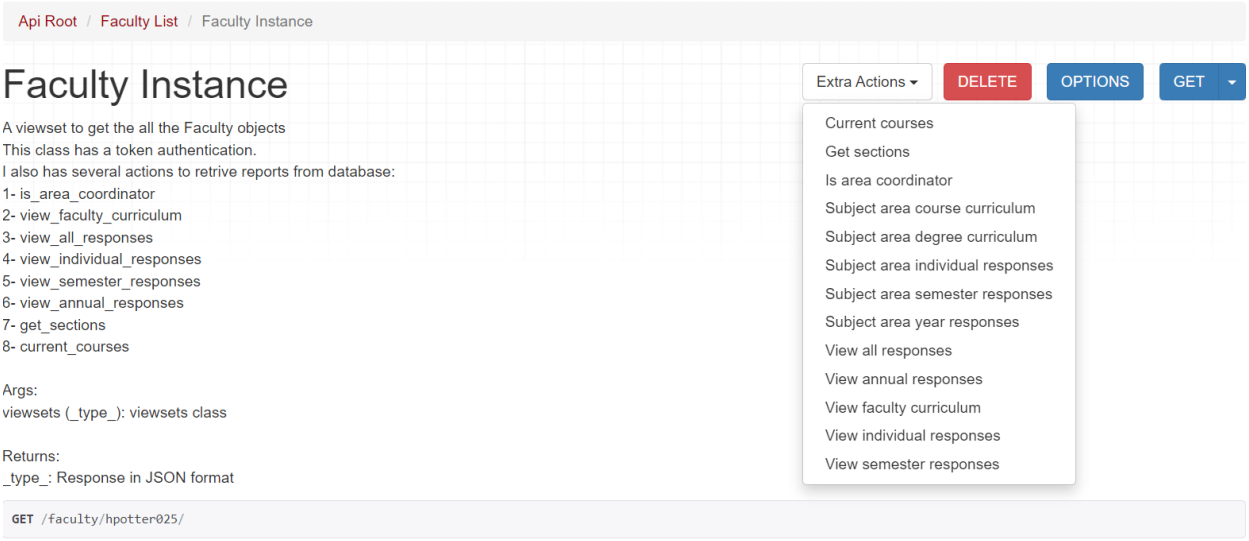
Every endpoint follows this structure when it is accessed directly.

Actions example

Some endpoints, such as /*outcomes/* and /*gen_questions/,* also have an extra dropdown button at the top-right side of the page (called a non-detailed action because a primary key is not needed to be able to execute it) which allows for extra filtering on the specified endpoint.



Moreover, some endpoints, such as */faculty/,* have actions that belong to specific primary key instances from faculty (called a detailed action). For example, say we have a faculty member with the username *hpotter025,* we have the following options if we navigate to */faculty/hpotter025/*:

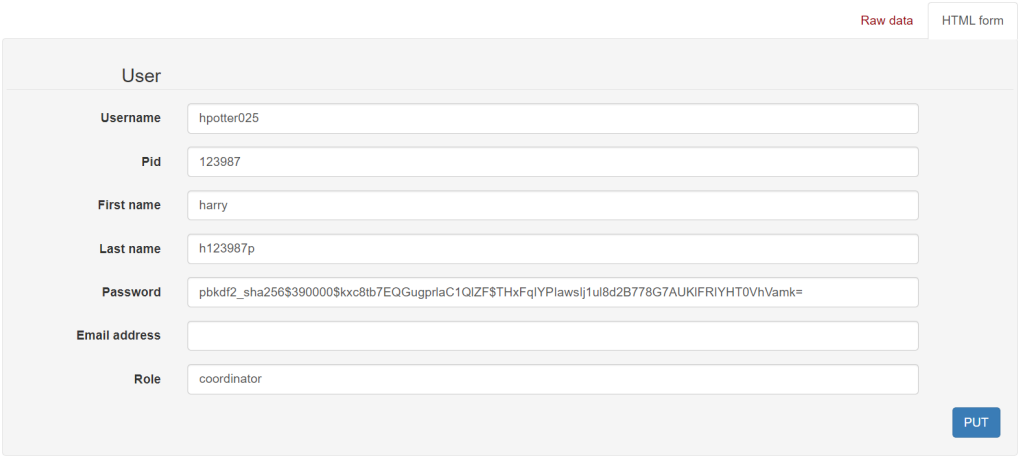


Actions are an excellent way to quickly add functionality to a table if there are not too many of them. This is an area of improvement for the system as stated in the Shortcomings video and

document. If there are too many actions, it becomes extremely difficult to maintain. Adding filtering to the views would be a much more flexible approach that can grow easily as more functionality is added to the tables.

Editing an instance

Accessing an instance directly allows the user to edit it. This function is performed as a PUT request.



Here, one can modify every field (if the primary key is modified, then a new entry will be created using everything that is autofilled).

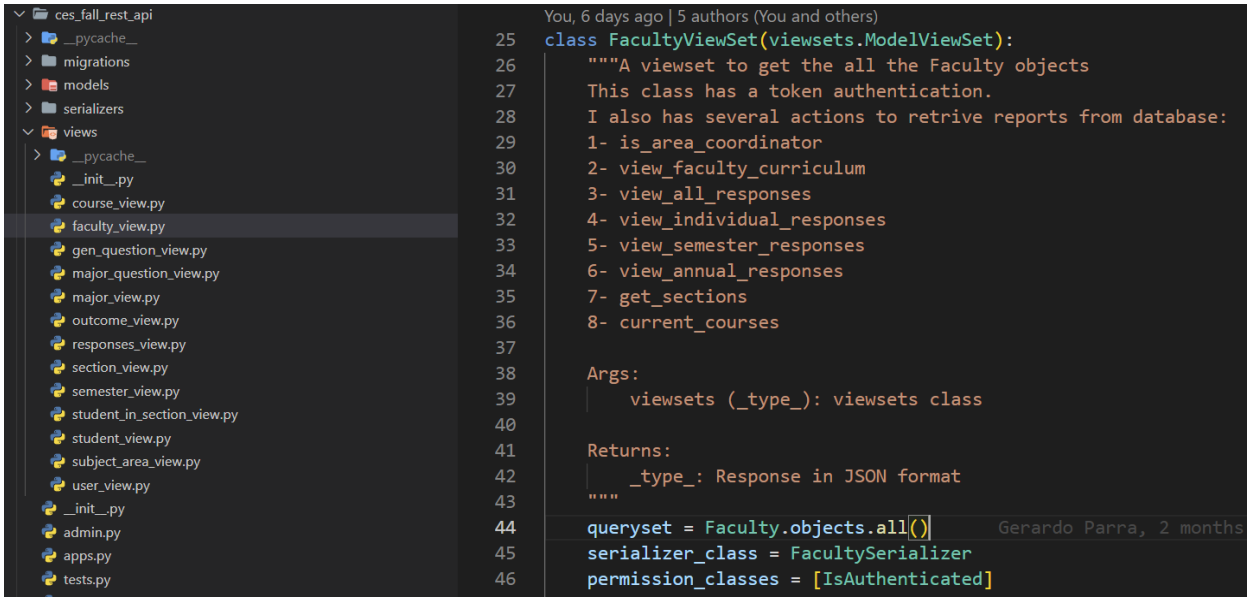
Usage

Any software that can talk HTTP can use the API (curl, python-requests, fetch API, etc). Our implementation of the frontend solely used the fetch API to make requests.

```
const api_endpoint = "http://localhost:8000";       You, 2 weeks ago • Merged faculty login and cleaned u

const fetchFromEndpoint = async (endpoint, token) => {
    return await fetch(`${api_endpoint}${endpoint}`, { headers: { Authorization: `Token ${token}` } });
}

const getToken = async (username, password) => {
    const body = new FormData();
    body.append("username", username);
    body.append("password", password);

    const response = await fetch(`${api_endpoint}/get-token/`, {
        method: "POST",
        body: body,
    });

    const status = response.status;
    const data = await response.json();
    const statusMessage = response.statusText;

    return { status, statusMessage, data };
};

export { fetchFromEndpoint, getToken };
```

Token Authentication

Token authentication is the auth system we used. Because of this, every request has to include an '**Authorization: Token *<token>***' header with the token generated by POSTing to the */get-token/* endpoint and providing a username and password that is present on the database. For testing and developing, this could make the process slow and tedious so it is recommended to comment-out the line: ***permission_classes = [IsAuthenticated]*** on whichever view is being tested or developed.



If testing with the auth system enabled, it is recommended to use Postman to interact with the API.

Admin

Every semester, the admin is in charge of loading all respective semester information. Because some tables contain foreign keys, there is an implied order to this operation. For instance, the */students/* endpoint, requires a major object and a user object. Currently, the recommended order to load semester data is:

1. **Faculty** (if different from last semester)
2. **Subject Area** (if different from last semester)
3. **Majors** (if different from last semester)
4. **MajorQuestions** (if different from last semester)
5. **Semester**
6. **Outcomes** (if different from last semester)
7. **Course**

8. **GenQuestions** (if different from last semester)
9. **Section**
10. **Users** (if different from last semester)
11. **Students** (if different from last semester)
12. **Students_in_section**

Because the current implementation of the CLI is unfinished, in terms of what it can add to the API, further development is highly encouraged. The CLI can combine multiple steps from this process. For instance, the creation of a Student, requires a User object, so the CLI can combine this process by testing the response received from the API when POSTing a user:

- Response: "username": ["user with this username already exists."]
    - Skip student creation
- Else, create student

In fact, a more summarized way could be to try and load students in sections in one command by taking advantage of the responses received from the API. So a Students In Section CSV file could have the following format:

username, pid, fname, lname, email, major, section

By reading this line, the CLI can do the following:

1. Test the response from POSTing username, pid, fname, lname, email to */users/*
    a. If the object was created (received an object with the specified fields)
        i. POST username, major to */students/*
        ii. POST username, section to */students_in_sections/*
    b. If the response is **"username": ["user with this username already exists."]** we know we can assume the user already exists AND it is a student, so we can safely:
        i. POST username, section to */students_in_sections/*

The following is an incomplete list of commands that could be combined:

- Course and Outcomes
- Sections and Semester
- Majors and MajorQuestions

Again, these would follow a similar process of first checking the response received from the API, then making a decision if to insert a new entry or use the existing entry.

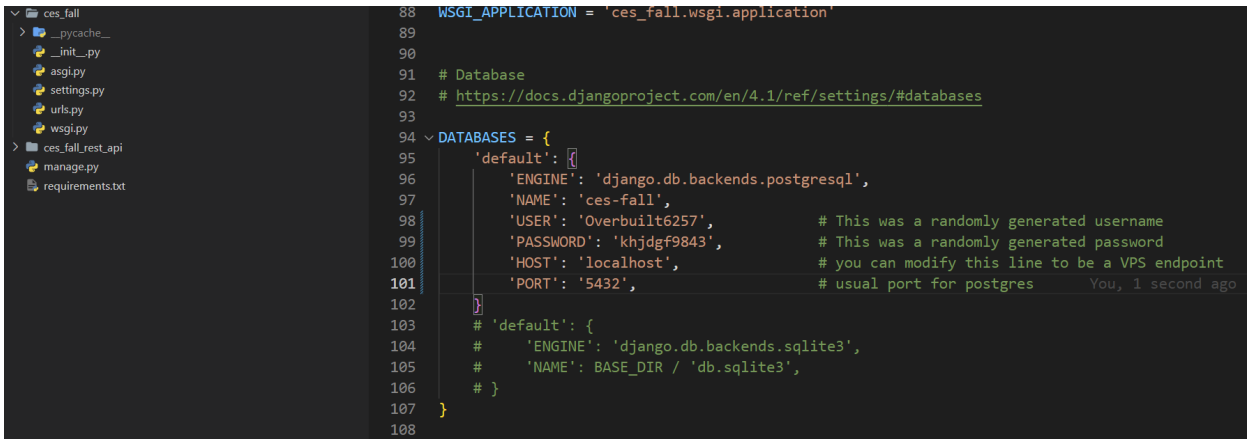
Implementation Details

This project was created with the following commands:

```
$ django-admin startproject ces_fall
$ cd ces_fall/
/ces_fall$ python manage.py startapp ces_fall_rest_api
/ces_fall$
```

Settings.py

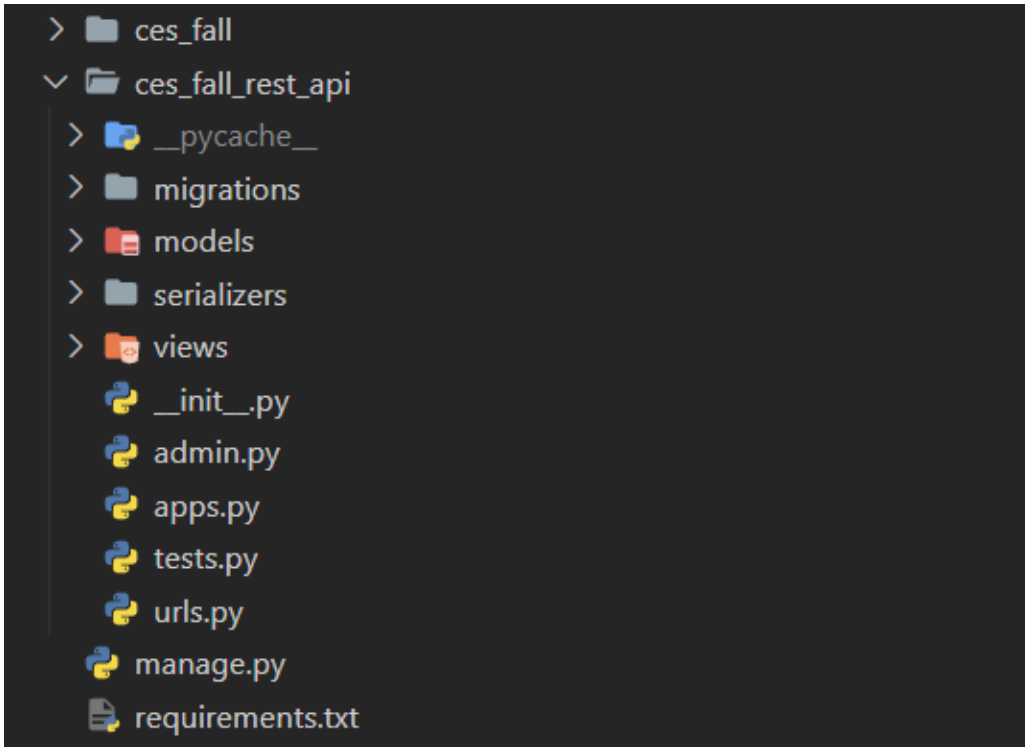
ces_fall_rest_api is the app where we implemented all the models, views and serializers. All django projects contain a configuration file, *settings.py,* which is usually located inside the folder that was created after the *startproject* command. This file holds all settings related to the django and all further packages that can be added to django projects. In our case, *settings.py* contains allowed hosts, origins, restframework authentication classes, and the most important setting: the database connection.



Throughout the development of the backend, we used a postgres instance hosted on a VPS because it was easier to share all development, rather than using a local postgres instance. This created a problem when we made substantial changes to the schema, because we needed to migrate entirely new tables, fields, modify current data types, etc., team members that did not have the latest migration changes were faced with errors from the compilation process. For a better development experience, it is recommended to use a local postgres instance to prevent the problem stated before. The reason why we did not use sqlite3 is because postgres supports array types inside fields (gen_questions, major_questions, responses, outcomes, all use this data type).

Folder Structure

The folder structure of the *ces_fall_rest_api* app has migrations, models, serializers, and views, each containing the respective python classes/functions.



Models

The **models** folder includes all the tables that will be present in the database. Django uses models as representations of tables and automatically converts these to python objects for a much easier approach to SQL (refer to the Django docs on Models for more information).

Serializers

The serializers folder contains all serializers required by the djangorestframework package. These are a way to serialize/deserialize requests sent from the views. These also take care of sanitizing data to make sure everything conforms to the models (refer to the REST Framework docs on Serializers to learn more).

Views

The views folder contains the views/endpoints that are used to create the API. Views are extremely flexible and customizable in the REST Framework. We used ModelViewSets because we wanted all endpoints to behave correctly to different types of HTTP requests. Some views such as *faculty_view.py* use the decorator *@action* which is used to create an action on the root of the view or as part of an instance: detail=False or detail=True (refer to the REST Framework docs on Views and Viewsets to learn more and how to customize the behavior).

```
48      # An action to check if the faculty is an area coordinator
49      @action(detail=True, url_path='is_area_coordinator')
50      def is_area_coordinator(self, request, pk=None):
51          try:
52              SubjectArea.objects.get(coordinator=pk)
53              return Response(True)
54          except SubjectArea.DoesNotExist:
55              return Response(False)
```

Each of these folders is also a package (contains __init__.py) to make importing easier and prevent cluttering of the namespace.

Config Files for app

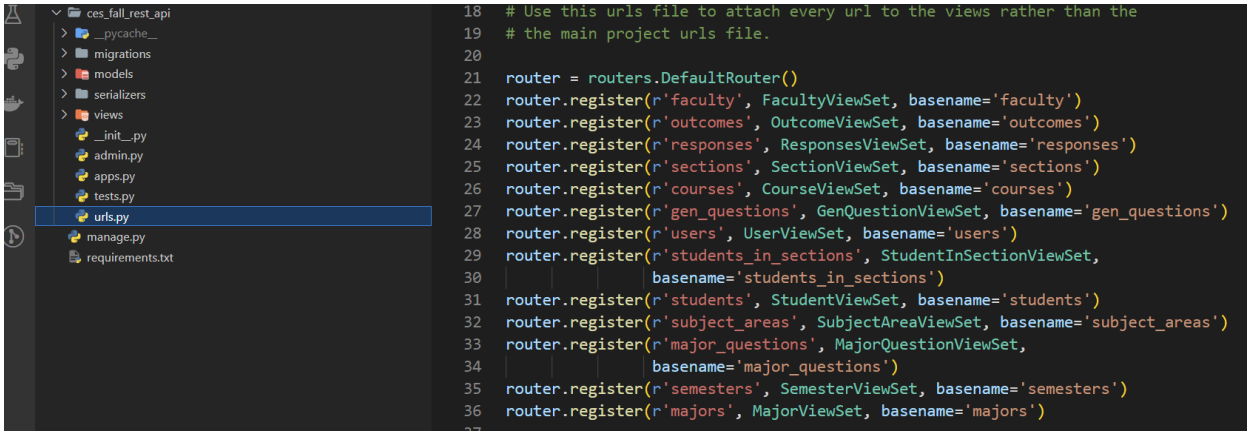
The admin.py and urls.py files are also very important.

Admin.py

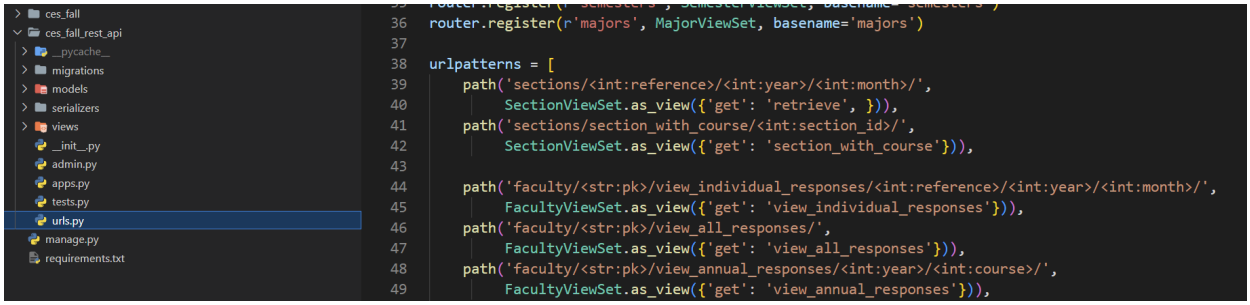
Admin.py is used to register models on the admin interface provided by Django. This can be extremely helpful when developing.

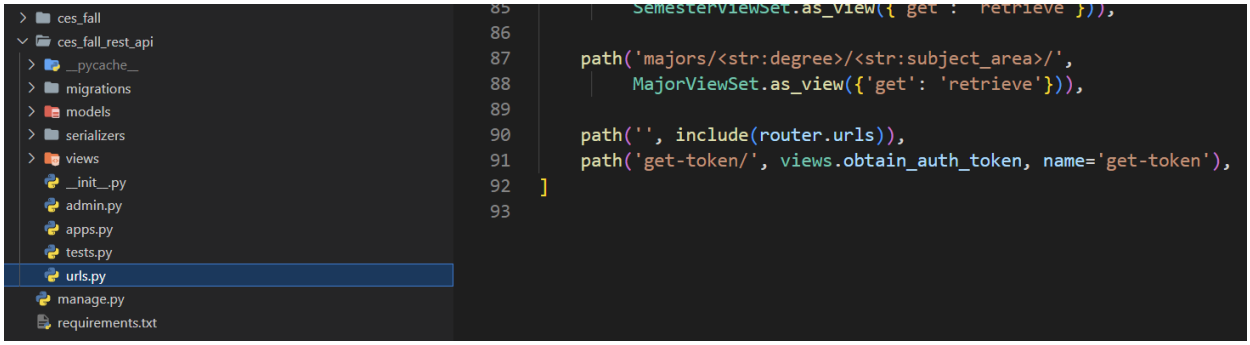
Urls.py

Urls.py is used to map all URLs available to the REST API. We used DefaultRouter because it automatically implements urls for GET, POST, PUT, etc..

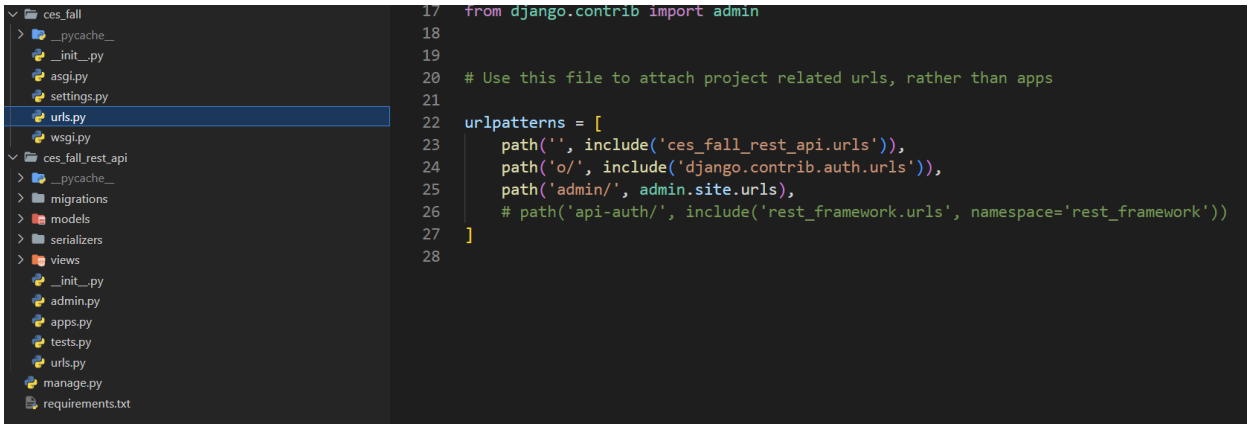


The urls.py file also contains *urlpatterns* which can be used to manually attach different URLs that will be used for individual actions, filtering, or anything related to a view's URL.





The ces_fall folder also contains a *urls.py* file which is used by django's URL dispatcher to generate all URLs (this is the reason why the app's urls.py is included here).



(Refer to the REST Framework docs on Routers to learn more).

Manage.py

Manage.py is the script used to create and migrate changes to the database, create super users, attach shells to the project for testing, running the django web server, etc.. This file is automatically generated by django-admin on project creation, and it is one of the most important files from django.

Important Commands

One of the commands we used constantly was ***python manage.py makemigrations && python manage.py migrate***. These commands modify and commit changes to the database as new modifications are available. Sometimes, django might not detect new changes made to the PostgreSQL 14.6 Documentation (www.postgresql.org) app so instead do ***python manage.py makemigrations ces_fall_rest_api && python manage.py migrate*** to apply the new changes. As mentioned before, an initial migration is included inside the ces_fall_rest_api/migrations, which can be used to deploy an empty database that conforms with the last changes made to the schema.
If a local postgres instance is running, simply do ***python manage.py migrate***, to create a new public schema under the username provided inside the ces_fall/settings.py file.
(Refer to the Django-Admin doc to learn more)

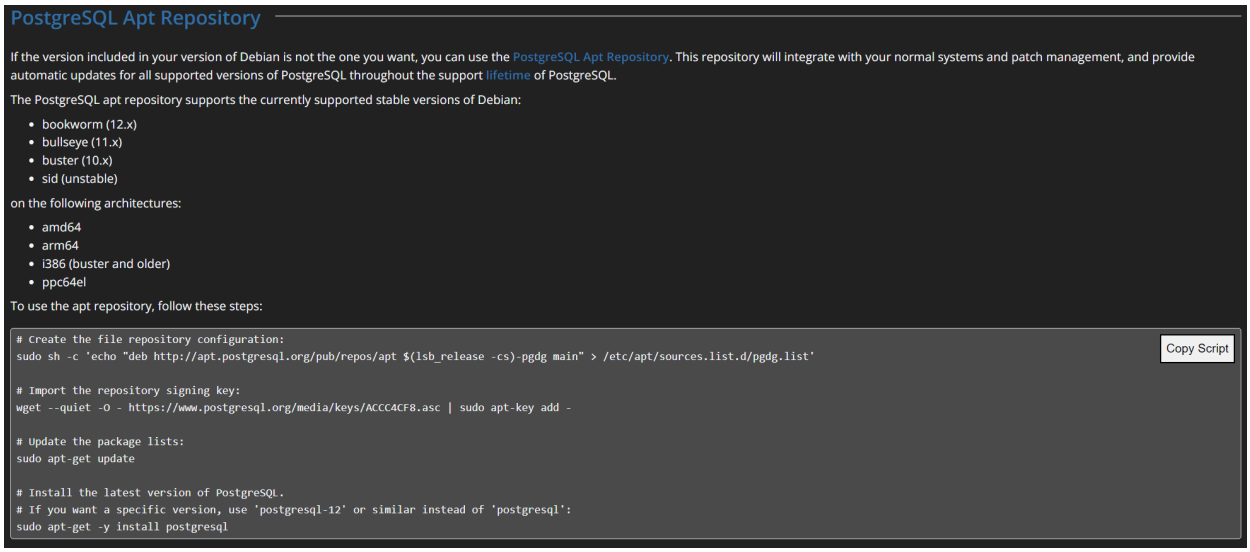
Installation Guide

Notes

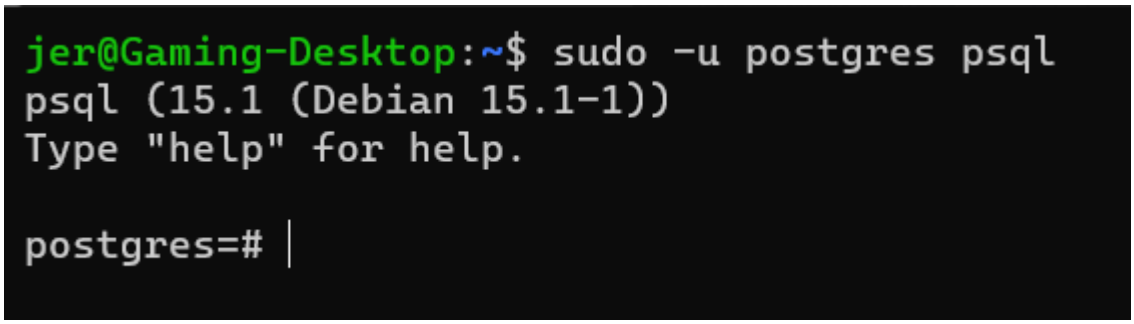
Included in the root of the backend folder is a pgdump_all backup (**postgres_backup.sql**) of the current database with some testing data (there is no private student or faculty info, only test students and faculty. The test credentials are: faculty: username - testfaculty1, password -

password1 student: username - teststudent1, password - password1) which can be restored locally. Tested on PostgreSQL 14 and 15.
The following are the steps required to setup postgres and restore the backup:

1. Install postgres on your platform from: PostgreSQL: Downloads (our platform of choice was Ubuntu/Debian because support is much more widely available).



2. After install run ***sudo -u postgres psql*** and this should take you to the psql shell.
    a. The installation created a default user '**postgres**' and a '**public**' schema
    b. If you get a socket error, check if the postgres service is running on systemd
        i. sudo systemctl status postgresql should tell you this



3. If the installation was successful, simply run:
    a. sudo -u postgres psql -f postgres_backup.sql postgres

```
jer@Gaming-Desktop:~/projects/ces-fall2022/backend$ sudo -u postgres psql -f postgres_backup.sql postgres
SET
SET
SET
DROP DATABASE
psql:postgres_backup.sql:23: ERROR:  role "Overbuilt6257" cannot be dropped because some objects depend on it
DETAIL:  owner of database template1
owner of database postgres
psql:postgres_backup.sql:30: ERROR:  role "Overbuilt6257" already exists
ALTER ROLE
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
UPDATE 1
DROP DATABASE
CREATE DATABASE
ALTER DATABASE
You are now connected to database "template1" as user "postgres".
SET
```

4. The restore created a superuser '**Overbuilt6257**' with password '**khjdgf9843**' and a database name '**ces-fall**'.

Refer to the User Manual: Settings.py for more information on setting up the database connection on the backend.

Backend Installation

For debian/ubuntu

1. Install python, pip, venv, python3-dev, libpq-dev
   sudo apt update && sudo apt install python3 python3-pip python3-venv python3-dev libpq-dev

```
jer@Gaming-Desktop:~$ sudo apt update && sudo apt install python3 python3-pip python3-venv python3-dev libpq-dev
Hit:1 https://deb.debian.org/debian testing InRelease
Hit:2 https://packages.cloud.google.com/apt cloud-sdk InRelease
Hit:3 https://deb.nodesource.com/node_19.x bookworm InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
1 package can be upgraded. Run 'apt list --upgradable' to see it.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3 is already the newest version (3.10.6-1).
python3-pip is already the newest version (22.3+dfsg-1).
python3-venv is already the newest version (3.10.6-1).
python3-dev is already the newest version (3.10.6-1).
libpq-dev is already the newest version (15.1-1).
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
```

2. Navigate to the project root folder, and create the virtual environment
   python3 -m venv env

```
jer@Gaming-Desktop:~/projects/ces-fall2022$ python3 -m venv env
jer@Gaming-Desktop:~/projects/ces-fall2022$ ls
backend  cli  env  frontend  LICENSE  README.md
jer@Gaming-Desktop:~/projects/ces-fall2022$
```

3. Activate the virtual environment created (should have **(env)** at the beginning of the line)
   source env/bin/activate

```
jer@Gaming-Desktop:~/projects/ces-fall2022$ source env/bin/activate
(env) jer@Gaming-Desktop:~/projects/ces-fall2022$
```

4. Update pip, setuptools and wheel
   pip install pip setuptools wheel -U

```
(env) jer@Gaming-Desktop:~/projects/ces-fall2022$ pip install pip setuptools wheel -U
Requirement already satisfied: pip in ./env/lib/python3.10/site-packages (22.3)
Collecting pip
  Using cached pip-22.3.1-py3-none-any.whl (2.1 MB)
Requirement already satisfied: setuptools in ./env/lib/python3.10/site-packages (65.5.0)
Collecting setuptools
  Using cached setuptools-65.6.3-py3-none-any.whl (1.2 MB)
Collecting wheel
  Using cached wheel-0.38.4-py3-none-any.whl (36 kB)
Installing collected packages: wheel, setuptools, pip
  Attempting uninstall: setuptools
    Found existing installation: setuptools 65.5.0
    Uninstalling setuptools-65.5.0:
      Successfully uninstalled setuptools-65.5.0
  Attempting uninstall: pip
    Found existing installation: pip 22.3
    Uninstalling pip-22.3:
      Successfully uninstalled pip-22.3
Successfully installed pip-22.3.1 setuptools-65.6.3 wheel-0.38.4
(env) jer@Gaming-Desktop:~/projects/ces-fall2022$
```

5. Install the dependencies
   pip install -r backend/requirements.txt

```
(env) jer@Gaming-Desktop:~/projects/ces-fall2022$ pip install -r backend/requirements.txt
Collecting asgiref==3.5.2
  Using cached asgiref-3.5.2-py3-none-any.whl (22 kB)
Collecting certifi==2022.9.24
  Using cached certifi-2022.9.24-py3-none-any.whl (161 kB)
Collecting cffi==1.15.1
  Using cached cffi-1.15.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (441 kB)
Collecting charset-normalizer==2.1.1
  Using cached charset_normalizer-2.1.1-py3-none-any.whl (39 kB)
Collecting cryptography==38.0.4
  Using cached cryptography-38.0.4-cp36-abi3-manylinux_2_28_x86_64.whl (4.2 MB)
Collecting Deprecated==1.2.13
  Using cached Deprecated-1.2.13-py2.py3-none-any.whl (9.6 kB)
Collecting Django==4.1.3
  Using cached Django-4.1.3-py3-none-any.whl (8.1 MB)
Collecting django-cors-headers==3.13.0
  Using cached django_cors_headers-3.13.0-py3-none-any.whl (13 kB)
Collecting django-oauth-toolkit==2.2.0
```

6. Now the backend is ready to start.
   python backend/manage.py runserver

```
(env) jer@Gaming-Desktop:~/projects/ces-fall2022$ python backend/manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
November 29, 2022 - 15:50:14
Django version 4.1.3, using settings 'ces_fall.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

For Windows systems

1. Install python 3.10 (or latest) binary from [Python Releases for Windows | Python.org]
2. When the installer opens, make sure to select “Install launcher for all users” and select “Add Python 3.10 to PATH”.
3. Install pip and virtual environment, run the following command
   python -m pip install — upgrade pip
   pip install virtualenv
4. Create a virtual environment using venv:
   virtualenv env

```
                          ~/Desktop/ces-fall2022/ces_fall
$ virtualenv env
created virtual environment CPython3.9.7.final.0-64 in 6500ms
```

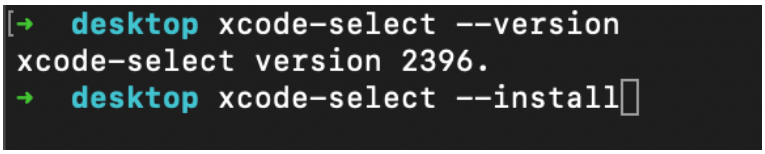
5. Activate the environment:
   source env\Scripts\activate
6. Install psycopg2-binary
   pip install psycopg2-binary

```
                              ~/Desktop
$ pip install psycopg2-binary

Collecting psycopg2-binary
  Downloading psycopg2_binary-2.9.5-cp39-cp39-win_amd64.whl (1.2 MB)
     ---------------------------------------- 1.2/1.2 MB 3.0 MB/s eta 0:00:00
Installing collected packages: psycopg2-binary
Successfully installed psycopg2-binary-2.9.5
```

7. Install the dependencies
   pip install -r backend/requirements.txt
8. Now the backend is ready to start.
   python manage.py runserver

For Mac

1. Install XCode and run the following command to enable emulation for x86
   xcode-select --install



2. Install Homebrew with the following command
   /bin/bash -c "$(curl -fsSL
   https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
3. Install python, pip, venv, run the following command
   brew install python
   python3 -m pip install --upgrade pip
   pip install virtualenv

```
[➜  desktop python3 --version
Python 3.10.8
[➜  desktop brew install python
Running `brew update --auto-update`...
==> Auto-updated Homebrew!
Updated 2 taps (homebrew/core and homebrew/cask).

You have 4 outdated formulae and 1 outdated cask installed.
You can upgrade them with brew upgrade
or list them with brew outdated.

Warning: python@3.10 3.10.8 is already installed and up-to-date.
To reinstall 3.10.8, run:
  brew reinstall python@3.10
```

```
[➜  desktop python3 -m pip install --upgrade pip
Requirement already satisfied: pip in /opt/homebrew/lib/python3.10/site-packages
 (22.3.1)
[➜  desktop pip install virtualenv
Requirement already satisfied: virtualenv in /opt/homebrew/lib/python3.10/site-p
ackages (20.16.3)
```

4. Navigate to the project root folder, and create the virtual environment
   python3 -m venv env
5. Activate the virtual environment created
   source env/bin/activate

```
[➜  desktop python3 -m venv env
[➜  desktop source env/bin/activate
(env) ➜  desktop
```

6. Install the dependencies
   pip install -r backend/requirements.txt
7. Now the backend is ready to start.
   python3 manage.py runserver

Frontend Installation

For debian-based linux systems

1. Check if you have Curl.
   sudo apt update && sudo apt install curl

```
jer@Gaming-Desktop:~/projects/ces-fall2022/frontend$ sudo apt update && sudo apt install curl
Hit:1 https://packages.cloud.google.com/apt cloud-sdk InRelease
Hit:2 https://deb.debian.org/debian testing InRelease
Hit:3 https://deb.nodesource.com/node_19.x bookworm InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
1 package can be upgraded. Run 'apt list --upgradable' to see it.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
curl is already the newest version (7.86.0-2).
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
jer@Gaming-Desktop:~/projects/ces-fall2022/frontend$
```

2. Add node repository.
   curl -fsSL https://deb.nodesource.com/setup_19.x | sudo -E bash - &&
   sudo apt-get install -y nodejs

```
jer@Gaming-Desktop:~/projects/ces-fall2022/frontend$ curl -fsSL https://deb.nodesource.com/setup_19.x | sudo -E bash - &&\
sudo apt-get install -y nodejs

## Installing the NodeSource Node.js 19.x repo...


## Populating apt-get cache...

+ apt-get update
Hit:1 https://packages.cloud.google.com/apt cloud-sdk InRelease
Get:2 https://deb.debian.org/debian testing InRelease [164 kB]
Hit:3 https://deb.nodesource.com/node_19.x bookworm InRelease
Get:4 https://deb.debian.org/debian testing/main amd64 Packages.diff/Index [63.6 kB]
Get:5 https://deb.debian.org/debian testing/main Translation-en.diff/Index [63.6 kB]
Get:6 https://deb.debian.org/debian testing/main amd64 Packages T-2022-11-29-2016.35-F-2022-11-29-2016.35.pdiff [1,639 B]
Get:6 https://deb.debian.org/debian testing/main amd64 Packages T-2022-11-29-2016.35-F-2022-11-29-2016.35.pdiff [1,639 B]
Get:7 https://deb.debian.org/debian testing/main Translation-en T-2022-11-29-2016.35-F-2022-11-29-2016.35.pdiff [943 B]
Get:7 https://deb.debian.org/debian testing/main Translation-en T-2022-11-29-2016.35-F-2022-11-29-2016.35.pdiff [943 B]
Fetched 293 kB in 1s (311 kB/s)
Reading package lists... Done

## Confirming "bookworm" is supported...

+ curl -sLf -o /dev/null 'https://deb.nodesource.com/node_19.x/dists/bookworm/Release'

## Adding the NodeSource signing key to your keyring...

+ curl -s https://deb.nodesource.com/gpgkey/nodesource.gpg.key | gpg --dearmor | tee /usr/share/keyrings/nodesource.gpg >/dev/null

## Creating apt sources list file for the NodeSource Node.js 19.x repo...
```

3. Navigate to the Frontend sub-folder.
   npm install --force

```
jer@Gaming-Desktop:~/projects/ces-fall2022/frontend$ npm install --force
npm WARN using --force Recommended protections disabled.
npm WARN ERESOLVE overriding peer dependency
npm WARN While resolving: react-textarea-counter@1.0.0
npm WARN Found: react@18.2.0
npm WARN node_modules/react
npm WARN   react@"^18.2.0" from the root project
npm WARN   15 more (@react-aria/ssr, @restart/hooks, @restart/ui, ...)
npm WARN
```

```
added 1481 packages, and audited 1482 packages in 8s

213 packages are looking for funding
  run `npm fund` for details

9 vulnerabilities (8 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
jer@Gaming-Desktop:~/projects/ces-fall2022/frontend$
```

4. Ready to run frontend.
   npm start

```
Compiled with warnings.

[eslint]
src/App.js
  Line 4:3:  'Switch' is defined but never used  no-unused-vars
  Line 5:3:  'Link' is defined but never used    no-unused-vars

src/components/FacultyReport/FacultyReportMenu.js
  Line 6:7:     'SubjectAreaYearDropdown' is assigned a value but never used
        no-unused-vars
  Line 13:25:   'semester' is assigned a value but never used
```

For Windows systems

1. Download .msi installer from nodejs.org
2. Navigate to the Frontend sub-folder.
   npm install --force
3. Ready to run frontend.
   npm start

For Mac

1. Install Node
   brew install node
2. Navigate to the Frontend sub-folder.
   npm install --force
3. Ready to run frontend.
   npm start

Wish List

Backend

- The way that we query the database through the REST API is by making manual actions for each of the fields we need from the frontend. We should have instead opted for filtering from the REST Framework instead, that way the API is more generalized, responsive and requires less modification.

- Implementing unit testing since the beginning of development. We realized too late that unt testing would be very useful in order to prevent some of the errors that were not as easy to catch.

- Implement OAuth in order to make use of the Panthersoft login (to prevent the users from having to remember multiple logins).
- In the CLI we need a reliable way to delete and modify data, including new commands including: bulk adding students in sections for a CSV file, etc., and having a UX.

Frontend

- Use Redux in combination with React. This would have made the frontend more flexible, responsive and efficient because of the use of caching and global state.
- Using a more modern framework such as Material UX in order to provide a more pleasing interface.
- Use RTK Query in order to cache requests to the backend.
- Remake the logic of the Evaluation Page (retrieve questions, outcomes, etc.) in order to reduce redundancy in the code.
- Redesign the Faculty Page to be more compact and straight forward. Include the Subject Area controls in a way that it does not take away from the regular faculty controls.