Graduation Helper Documentation

Table of Contents

Description of Project 2

Software Design Process 2

Requirements & Specifications 3

Functional Requirements 3

Use Case: Register 3

Use Case: Log In 3

Use Case: Log Out 4

Use Case: Reset Password 4

Use Case: Generate Schedules 5

Use Case: Add Classes 5

Use Case: Choose Preferences 6

Use Case: Save Schedule 7

Use Case: Edit Schedule 7

Use Case: Add Sections 8

Use Case: Remove Sections 8

Use Case: Delete Schedule 9

Non-Functional Requirements 9

Frameworks and Programming Language 9

Coding Style 9

Architecture & Design 10

Front-End 10

Back-End 10

Database 11

Reflections & Lessons Learned 12

Ben 12

George 12

Mingze 12

Rachel (Eun Sun) 12

Sergey 13

Quang 13

Zuyi 13

Description of Project

Graduation Helper is a web application that generates a UIUC class schedule based on a student's desired classes and scheduling preferences. This app solves the problem of having to go through many combinations of class sections in order to find the most desirable combination while also avoiding conflicts. Instead, we do all of this automatically, making it much easier for students to obtain their preferred schedule.

Software Design Process

We chose to follow a slightly modified version of Extreme Programming (XP) while developing Graduation Helper. As required by XP, we developed our software over the course of six different iterations. After every two weeks, we received feedback on potential issues with our software from the TA, and we took this feedback into account during subsequent iterations. As a result of the feedback, we also changed which user stories we prioritized throughout the course of the project.

We also attempted to constantly maintain code clarity. This was done with refactoring, as we removed commented out code as often as possible, extracted duplicate code into new methods, and changed the functionality of certain classes in order to adapt to changing requirements. We also constantly maintained proper coding style by enforcing the JavaScript semistandard style on all commits that were to be merged into the master branch, which helped avoid some potential code readability issues.

In order to make collaboration easier, we split our seven person team split into three different groups for each iteration: front-end, back-end, and testing. We did this because it would allow each of us to focus on specific areas of the project, as well as taking full advantage of each of our individual skill sets. These groups had to communicate constantly with one another about what each group's requirements were. For example, the back-end group provided API documentation to the front-end and testing groups, which made their implementations much easier. However, if the front-end group later realized that they might need something new that the API does not provide, this would have to be communicated to the back-end group, resulting in different requirements.

Our main modification on the XP process was in the area of pair programming. Within each of the groups, we often took advantage of pair programming, but we did not rely on it for all programming work as XP would require, as this was often not feasible due to the academic nature of the project and conflicts between our schedules. However, although we wrote most of our code individually, we did extensive code review, with several different team members looking over the code in each pull request as well as manually testing the actual functionality before the code could be merged. This also resulted in very good collective code ownership, with each of us having a very solid understanding of how other areas of the web application were implemented.

Our team also took advantage of automated testing for both the front-end and the back-end of our web application. We set up our continuous integration script to run all front-end and back-end tests after every commit, requiring all tests to pass before any code could be merged into the master branch. We also chose not to follow XP's guideline on writing tests first; instead, we wrote all of our tests after writing the original code. This gave the front-end and back-end more room to experiment with what would be the best way to implement each feature. However, we still planned our tests in advance, giving the front-end and back-end groups a basic guideline on what they should be implementing.

Requirements & Specifications

Functional Requirements

Use Case: Register

Primary Actor: UIUC student with no Graduation Helper account
Goal in Context: Create an account with an email, username, and password, and send an email with a link to activate the account
Scope: User
Level: User Goal
Stakeholders and Interests: Students - want to be able to create an account in order to use the application
Preconditions: The user is not currently logged in.
Minimal Guarantees: No account will be created unless all information is correct: the user is using a UIUC email, the password is at least 10 characters, and there is no existing account associated with the user's email.
Triggers: The user clicks on the register button on either the home page or the login page
Main Success Scenario:
1. The user enters their email, username and password. The password is entered twice in order to confirm that it is correct.
2. The user clicks on the sign up button.
3. The user is redirected to the home page, which tells them to check their email in order to activate their account.
4. The user opens their email and clicks on the link in the email to activate their account.
5. The user is redirected to the home page, where they are now logged in.
Extensions:
2a. An error message is displayed if any of the provided information is incorrect.

Use Case: Log In

Primary Actor: UIUC student with a Graduation Helper account
Goal in Context: Log in to an existing Graduation Helper account

Scope: User
Level: User Goal
Stakeholders and Interests: Students - want to be able to log in in order to use the application
Preconditions: The user is not currently logged in.
Minimal Guarantees: The user will not be logged in unless the email and password are correct and the account has already been activated.
Triggers: The user clicks on the Login button on the home page
Main Success Scenario:
1. The user enters their email and password.
2. The user clicks on the Login button.
3. The user is redirected to the home page, where they are now logged in.
Extensions:
2a. An error message is displayed if any of the provided information is incorrect.

Use Case: Log Out

Primary Actor: UIUC student with a Graduation Helper account
Goal in Context: Log out of the Graduation Helper application
Scope: User
Level: User Goal
Stakeholders and Interests: Students - want to be able to log out when they are done using the application
Preconditions: The user is currently logged in.
Minimal Guarantees: The user will be redirected back to the home page and will no longer be able to access the page they were on unless they log back in again.
Triggers: None
Main Success Scenario:
1. The user hovers over their username in the corner of the page in order to bring up a drop-down menu.
2. The user clicks on the Logout button.
3. The user is redirected to the home page, where they are now logged out.
Extensions:
None

Use Case: Reset Password

Primary Actor: UIUC student with a Graduation Helper account
Goal in Context: Let the user to change their password to a new one when they have forgotten their old password
Scope: User
Level: User Goal
Stakeholders and Interests: Students - want to be able to access their account even if they have forgotten their password
Preconditions: The user is not currently logged in.

Minimal Guarantees: The password will not be changed unless the authentication code is correct.
Triggers: The user clicks on the Forgot Password button on the login page
Main Success Scenario:
1. The user enters their email.
2. The user clicks on the Send Email button.
3. The user checks their email, finds the email from Graduation Helper, and enters the authentication code.
4. The user enters their new password. The new password is entered twice in order to confirm that it is correct.
5. The user clicks on the Reset Password button.
6. The password is changed. The user is redirected to the home page, where they are now logged in.
Extensions:
2a. An error message is displayed if the email is not associated with an existing account.
5a. An error message is displayed if the authentication code is incorrect or the passwords do not match.

Use Case: Generate Schedules

Primary Actor: UIUC student with a Graduation Helper account
Goal in Context: Given a set of classes and preferences provided, generate several different potential schedules and display them to the user
Scope: User
Level: User Goal
Stakeholders and Interests: Students - want to be able to generate schedules to make the class registration process easier
Preconditions: The user is currently logged in.
Minimal Guarantees: The generated schedules will not have any conflicts between sections. The generated schedule will contain all of the classes the user has selected.
Triggers: The user clicks on the Generate Schedule button on the home page
Main Success Scenario:
1. The user uses a drop-down menu to select a year and semester.
2. The user clicks the Next button and gets redirected to the Generate Schedule page.
3. The user chooses which classes to add. (Use Case: Add Classes)
4. The user chooses which preferences they want. (Use Case: Choose Preferences)
5. The user clicks on the Generate button.
6. A list of preview images for generated schedules is displayed to the user.
Extensions:
5a. The Generate button is disabled until the user has added at least one class.

Use Case: Add Classes

Primary Actor: UIUC student with a Graduation Helper account

Goal in Context: Before generating schedules, let the user to add the classes that they want to be on the schedule
Scope: User
Level: User Goal
Stakeholders and Interests: Students - want to be able to choose what sections they want on their schedule
Preconditions: The user is currently logged in. The user has navigated to the Generate Schedules page.
Minimal Guarantees: The list of displayed classes is up to date based on the UIUC Course Explorer.
Triggers: None
Main Success Scenario:
1. The user clicks on the Select Class dropdown, which displays a list of subjects.
2. The user chooses a subject.
3. The user chooses a class from the displayed list of classes for the subject.
4. The user clicks on the Add button in order to add the class.
Extensions:
4a. The user may repeat this step for each of the classes they want to add.

Use Case: Choose Preferences

Primary Actor: UIUC student with a Graduation Helper account
Goal in Context: Before generating schedules, let the user to choose their preferences for class days/times
Scope: User
Level: User Goal
Stakeholders and Interests: Students - want to see schedules with more desirable class times first
Preconditions: The user is currently logged in. The user has navigated to the Generate Schedules page.
Minimal Guarantees: None
Triggers: None
Main Success Scenario:
1. The user chooses which day they don't want class with the No Class Days dropdown, and uses the corresponding Add button to add the preference.
2. The user chooses which time they don't want class (Morning, Lunch, or Evening) with the No Class Times dropdown, and uses the corresponding Add button to add the preference.
3. The user clicks on the Select No Class Time button.
4. The user drags to select a specific time on a specific day when they do not want class.
Extensions:
1a. The user may repeat this step for each of the days they want to add.
2a. The user may repeat this step for each of the times they want to add.
3a. The user may repeat this step for each of the specific times they want to add.

Use Case: Save Schedule

Primary Actor: UIUC student with a Graduation Helper account
Goal in Context: Save a generated schedule
Scope: User
Level: User Goal
Stakeholders and Interests: Students - want to be able to save a generated schedule in order to view or edit it later
Preconditions: The user has chosen to generate schedules, and there is at least one generated schedule.
Minimal Guarantees: Any schedule that the user chooses to save gets stored in the database.
Triggers: The user clicks on the preview image for a generated schedule
Main Success Scenario:
1. The user looks at the details of the generated schedule in order to make sure that it matches what they want.
2. The user clicks on the Save button.
3. The generated schedule is closed, and the user can see the list of preview images of generated schedules again.

Extensions:
None

Use Case: Edit Schedule

Primary Actor: UIUC student with a Graduation Helper account
Goal in Context: Edit a previously saved schedule by adding or removing sections, and then save it again after making changes
Scope: User
Level: User Goal
Stakeholders and Interests: Students - want to be able to edit a generated schedule in case their plans change or it does not fully match their needs
Preconditions: There is at least one saved schedule.
Minimal Guarantees: The schedule will not be modified until the user chooses to save it. Any valid schedule that the user chooses to save gets stored in the database. A schedule will only be saved if it has at least one section.
Triggers: The user clicks on the preview image for a saved schedule on the My Schedules page
Main Success Scenario:
1. The user looks at the details of the saved schedule in order to make sure that they want to edit it.
2. The user clicks on the Edit button.
3. The user is redirected to the Edit Schedule page. The user views the sections currently in their schedule.
4. The users chooses the sections they want to add to their schedule. (Use Case: Add Sections)

5. The user chooses the sections they want to remove from their schedule. (Use Case: Remove Sections)
6. The user clicks on the Save button.
7. A message appears to confirm that the schedule has been successfully saved in the database.

Extensions:
6a. An error message appears if the user tries to save an empty schedule.

Use Case: Add Sections

Primary Actor: UIUC student with a Graduation Helper account
Goal in Context: While editing a schedule, the user can look up a class, see its sections, and choose which sections to add to their schedule
Scope: User
Level: User Goal
Stakeholders and Interests: Students - want to be able to add new sections if they want to take a new class
Preconditions: The user is editing a schedule on the Edit Schedule page.
Minimal Guarantees: The list of displayed classes and sections is up to date based on the UIUC Course Explorer.
Triggers: None
Main Success Scenario:
1. The user clicks on the Select Class dropdown, which displays a list of subjects.
2. The user chooses a subject.
3. The user chooses a class from the displayed list of classes for the subject.
4. The user clicks the Choose button, which results in the sections for that class being displayed.
5. The user clicks on the checkbox next to any sections that they want to add to their schedule.
6. The newly added section is now displayed in the schedule view.

Extensions:
5a. The user can click on an already checked checkbox in order to remove a section from their schedule.

Use Case: Remove Sections

Primary Actor: UIUC student with a Graduation Helper account
Goal in Context: While editing a schedule, the user can see the sections in their schedule and remove any of these sections
Scope: User
Level: User Goal
Stakeholders and Interests: Students - want to be able to remove sections if they no longer want to take a class
Preconditions: The user is editing a schedule on the Edit Schedule page.
Minimal Guarantees: None

Triggers: None
Main Success Scenario:
1. The user clicks on the X button next to the section they want to delete.
2. The deleted section is removed from the schedule view.
Extensions:
None

Use Case: Delete Schedule

Primary Actor: UIUC student with a Graduation Helper account
Goal in Context: Delete a previously saved schedule
Scope: User
Level: User Goal
Stakeholders and Interests: Students - want to be able to delete a schedule if they no longer want it, in order to reduce clutter
Preconditions: There is at least one saved schedule.
Minimal Guarantees: Any schedule that the user chooses to delete will be removed from the database.
Triggers: The user clicks on the preview image for a saved schedule on the My Schedules page
Main Success Scenario:
1. The user looks at the details of the saved schedule in order to make sure that they want to delete it.
2. The user clicks on the Delete button.
3. The schedule details get closed, and the preview image for the deleted schedule disappears from the list.
Extensions:
None

Non-Functional Requirements

Frameworks and Programming Language

React and Node.js must be used for the front-end. Express.js must be used for the back-end. Mocha must be used for both front-end and back-end tests. Selenium WebDriver must be used for front-end tests. All code must be written in JavaScript.

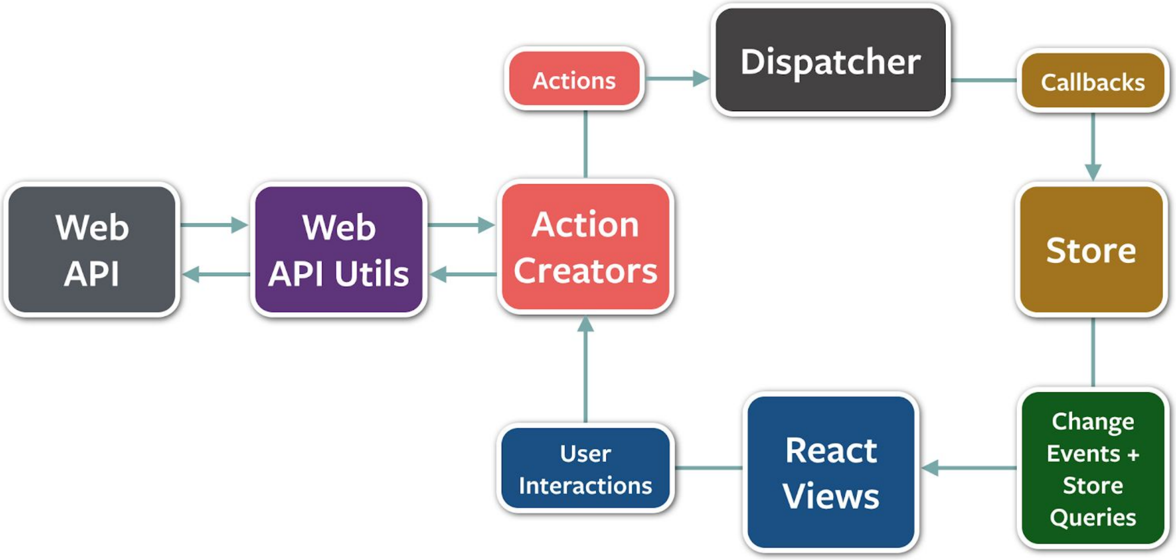
Coding Style

The JavaScript Standard Style (with semicolons) must be used.

Architecture & Design

Front-End

Our front-end uses React and its corresponding flux architecture.



Frontend UML Class Diagram

Back-End

Our back-end uses a popular Node.js framework called Express.js for the server side communication with the React front-end. This framework gives us a simple way to implement a rest api for our front-end to use. Our back-end system can be, in a way, classify into 3 general categories for our rest api, general purpose, user data, and core features.

In our most basic category, we have a general purpose api for directly getting information from the university xml which we would then parse and send back the info to the front-end. Information such as getting a list of years, semesters, courses, sections, and etc. can all be accessed from these api calls. Since they are all dynamically retrieved from the university data, any update on their end will automatically update ours as well.

Another category is our api calls for interacting with user data. These api, in essence, deals with the data from our users and, as such, mainly interacts with our database hosted on Microsoft Azure. Information such as user registration, authentication, and login which affects storing data are part of this category. Included are features for user interaction with our generated schedules. Whenever a user saves, deletes or edits the schedules they send back the request for the actions to these api along with the required data. Information such as user id or courses could be sent depending on the specific action. Once the request is received, the back-end makes the database connection and updates the information accordingly or inserts new entries.

Our final category is what we would consider as the main feature(s) of our website, the course generation. After the user selects all the information they would like on the front-end, such as the semester, year, and list of classes they want to take, the front-end would make a request to this api and the generation process will begin. The course generation works by first using some general utility functions which we are used in the other parts of the code and retrieves all the necessary information about these picked classes. Information about all the specific sections are collected during this stage. After the collection, all of the data is pre-process to correlate which class sections must go together for that particular course. This stage makes it easier for the next stage which combines all the course sections together. The final stage of this generation process calculates all the possible permutations of the pre-processed course sections and builds the non-conflicting schedules. If user preferences are set, then the score will be calculated for each schedule during this process as well. Finally, the generated schedules are then sorted by their score and the data sent back to the front-end.

Api documentation can be seen by running the following command with apidoc installed. This will generate a apidoc folder with an html file that can be open to visually see the docs.

```
  $ apidoc -i routes/ -o apidoc/
```

Backend UML Class Diagram
Generate Schedules Sequence Diagram
Choose Classes Sequence Diagram
Save Schedule Sequence Diagram

Database

To store data, our website uses a mysql database hosted on Microsoft Azure. Our Graduation Helper database consists of 4 tables which stores users, authentication, courses, and schedules. The users table, as its name implies, stores user info such as user id, username, email, and etc. The authentication table is connected to users for use in registration authentication and as well as reset. The schedules table contains all the saved user schedules serving as a connection between users and the schedules courses in the courses table. For each schedule saved the courses are stored in the courses table and a new schedule entry is created to identify the newly stored courses for that particular schedule and user.

Reflections & Lessons Learned

Ben

As our project was divided into three divisions, I worked as a part of the backend team. Throughout this project I was exposed to Node.js, Javascript, and HTTP requests for the first time and learned a great deal from this project in all the aforementioned areas. As I struggled my way through, my team members were very supportive and of great help. I wrote API calls using the course explorer API as well as schedule based API calls in addition to DB schema design and interactions. Additionally, I learned the importance of communication between the frontend and backend teams and how each worked together to implement features. Overall this project and team were both an extremely positive experience for me, despite some rather bad experiences with team projects in the past.

George

This is the first time I got involved in a start-from-scratch project, and I really learned a lot from it. We divided into different groups and I was mainly in charge of user systems part in the backend. In coding skills aspects, I learned node.js while I never used javascript before, as well as how to build a robust user system (User database, login, register, etc). What I found also precious is the experience of the group communications. Working in such a nice group made me realize the importance of teamwork!

Mingze

We divided the group into three small groups – backend, frontend, and testing group. I mainly focus on the frontend. I learned the importance of communication between different groups. For example, frontend guys need to discuss with backend guys the specifications of the API calls, such as the number of parameters and the format of each parameter. It is also necessary for frontend group to communicate with testing group. For example, the frontend needs to tell the testing group whether a result is a feature for now or a bug.

Rachel (Eun Sun)

This project was a good opportunity for me to learn javascript and testing skills because I had not used javascript before. Also, I learned a good team work from group members. I appreciate them for their hard work and kindness. I used to have a negative thought about group project since I had bad experience from other CS classes, but my thought has changed through this group project. I really enjoyed working with this team although it was hard to learn and understand javascript.

Sergey

Our project was split up into three different groups. As a part of the testing group, I set up continuous integration with Travis and wrote the front-end tests. I had to learn how to use the Selenium JavaScript API and troubleshoot some compatibility issues in order to get these tests to work in Travis. The biggest thing that I learned from this project, though, was the value in communication between different teams. Since we were all doing very different tasks, it was really important for each team to tell the other teams when they needed something done, and for each team to understand the changes that other teams have recently made.

Quang

For the project, most of our development was split between different groups. During the project, I worked mainly on the backend with implementing various api calls for the front-end and generating schedules. I've learned that it is good to plan out things beforehand as to not have to refactor too much. Keeping in contact with others is also very important since when working on a team project. If you don't let the front-end and testing people know what or how something is implemented then they can't use your work for the rest of the project.

Zuyi

I was part of the testing group and was in charge of backend testing. I had to learn how to use the node testing environment of mocha and chai and several advanced testing libraries. From the project I've learnt that the most important thing is to be up to date and keep contact with other group members on their progresses. Thus you can get everything done on time and contribute to the project in a speedy manner