



# Pete's Plan

Design Document

## Team 36

- Anushka Sharma
- Chloe Wang
- Cole Doner
- Patricia Madalena Magalhaes Casaca
- Shivani Venkatraman

# Index

<b>Index</b>	<b>1</b>
<b>Purpose</b>	<b>2</b>
Functional Requirements	2
Non-Functional Requirements	4
<b>Design Outline</b>	<b>6</b>
Web Client	6
Server	6
Database	6
Component Interaction	6
<b>Design Issues</b>	<b>8</b>
Functional Issues	8
Non-Functional Issues	9
<b>Design Details</b>	<b>11</b>
Data Class Diagram	11
Class Description and Interactions	11
Sequence Diagrams	15
User visits site	15
User log-in	16
User register	17
User resets password	18
User views degree info	19
Server updates course info	20
UI Mockups	21
Landing Page (After User Login)	21
Student Requirement Page	22
Grade Calculator Page	23

## Purpose

Purdue University offers 6,700 courses in 200 specializations, making it nearly impossible for students to navigate all the requirements. Additionally, degree plan information is widely dispersed and difficult to find, not only for new students, but also upperclassmen. Pete's Plan will allow students to make degree plan choices more effectively and in a well-informed manner by pulling together resources such as plans of study, course descriptions, prerequisites, course scheduling, and more. This eliminates the need for hours of browsing through an enormous number of tabs, prevents students from asking their advisor the wrong questions, and provides guidance to students who are feeling overwhelmed.

## Functional Requirements

\* = if time allows

### 1. Users can manage their account

**As a user,**

- a. I would like to sign up for an account
- b. I would like to log in/out of my account
- c. I would like the page to remember me so I don't have sign in every time
- d. I would like to reset my password via email
- e. I would like to modify/delete my account
- f. \*I would like to use the site without being logged in
- g. \*I would like to be promoted to an admin
- h. \*I would like to sign up for Universities other than Purdue University WL

### 2. Users can enter/modify their course plan

**As a user,**

- a. I would like to enter my current plan of study
- b. I would like to modify my current plan of study
- c. I would like to enter my degrees
- d. I would like to enter transfer credits/placement tests
- e. I would like to enter AP scores
- f. I would like to create and navigate multiple degree plans
- g. I would like to be notified if I am missing a prerequisite
- h. I would like to override prerequisite warnings
- i. \*I would like to download an offline copy of my data
- j. \*I would like to upload a previously downloaded data file to my account

### 3. Users can enter specific course data

**As a user,**

- a. I would like to enter my final score for each course
- b. I would like to select a professor for each course
- c. I would like to select course times for each course

4. Users can get degree information

**As a user,**

- a. I would like to access major requirements
- b. I would like to access minor requirements
- c. I would like to see degree requirement overlap
- d. I would like to see the total requirements for multiple degrees
- e. I would like courses to be recommended to me based on selected degrees and remaining required courses
- f. \*I would like to see non-course degree requirements (e.g. Civics Literacy)

5. Users can get course information

**As a user,**

- a. I would like to access course descriptions
- b. I would like to access course prerequisites
- c. I would like to access previous course scheduling
- d. I would like to view previous course instructors
- e. I would like to see instructor ratings on Rate My Professor
- f. I would like to see classes in Boiler Grades
- g. I would like to see instructors in Boiler Grades
- h. \*I would like to see options to test out of the course (e.g. CLEP Tests)

6. Users can view a visual degree plan

**As a user,**

- a. I would like the plan to change dynamically with entered information
- b. I would like to see a block describing the course when I hover over it
- c. I would like the degree plan to be sorted by semester
- d. I would like courses to be clearly color coded by completion
- e. \*I would like to see a calendar view of my classes each semester
- f. \*I would like to export my course schedule to another calendar app

7. Users can calculate grades

**As a user,**

- a. I would like to see my cumulative GPA
- b. I would like to see my semester GPA
- c. I would like to see my major GPA
- d. I would like to approximate grades based on assignment grades and category weights

8. \*Course and instructor rating system

**As a student,**

- a. \*I would like to view reviews for instructors
- b. \*I would like to leave reviews on instructors I have had
- c. \*I would like to view reviews for courses
- d. \*I would like to leave reviews for courses I have taken

## 9. \*Admin Panel

**As an administrator,**

- a. \*I would like to manually update course data
- b. \*I would like to update other important information

## 10. \*Communicate with other users

**As a user,**

- a. \*I would like to optionally have contact information shared by default
- b. \*I would like to optionally have contact information shared by course
- c. \*I would like to view the contact information of other students enrolled in the same course as me
- d. \*I would like to communicate with classmates in the same course as me through the website

## 11. Course data automatically updates

**As an administrator,**

- a. I would like the server to automatically update course descriptions
- b. I would like the server to automatically update previous/current scheduling
- c. \*I would like the server to fetch Rate My Professor data as needed
- d. \*I would like the server to fetch Boiler Grade data as needed

## Non-Functional Requirements

## 1. Architecture

**As a developer,**

- a. I would like the front-end and back-end separated to encapsulate the application's relevant tasks.
- b. I would like to use ReactJS and TypeScript for the front-end of the application.
- c. I would like to use a NodeJS back-end in order to manage requests, house course catalog/scheduling information, and support information refresh.
- d. I would like to use a MongoDB (NoSQL) database to store account, degree-related, and course-relevant information.
- e. I would like to focus on the usability of the application.

## 2. Performance

**As a developer,**

- a. I would like our application to load/update the webpage in under 5 seconds.
- b. I would like page navigation to take less than 1 second.
- c. I would like UI updates (ex: button presses) to take less than 100ms to update (or provide a loading icon).
- d. I would like fast API responses, with <1Mb of data taking less than 1s and larger amounts taking no more than 5s.
- e. I would like our backend to have high uptime >99.9% and to be free from bugs that would cause crashing.

### 3. Security

#### **As a developer,**

- a. I would like to hash passwords using a common password hashing mechanism (ex: bcrypt).
- b. I would like to implement authentication and access control on our API.
- c. I would like to use rate limiting to prevent systematic attacks.
- d. I would like to use object-relational mapping to simplify database accesses and reduce potential attack surfaces.

### 4. Usability

#### **As a developer,**

- a. I would like the website interface to be intuitive and easy to understand.
- b. I would like the website's different features to be categorized by area and represented uniquely.
- c. (If time allows) I would like the website's graphical representations to also be represented in plain text for accessibility.
- d. (If time allows) I would like the lab to be accessible from mobile clients
- e. I would like specific data to be imported and exported out of the service to allow for information sharing.

### 5. Testing

#### **As a developer,**

- a. I would like to implement separate front-end tests using a front-end testing framework (ex: Jest).
- b. I would like to implement separate back-end tests using a back-end testing framework.
- c. I would like to implement end-to-end testing using an end-to-end testing tool (ex: Protractor).

### 6. Hosting

#### **As a developer,**

- a. I would like to run NodeJS and React on a local machine, utilizing Docker to host a local database.
- b. I would like to utilize a publicly accessible server to be used as a testing website for development purposes.

## Design Outline

Our project will allow users to plan courses in their project degrees. We will use a Client-Server model. The client will be a web application that interfaces with a backend server, and the client will use a Model-View-Controller pattern. The server will also access and store data in a database, and periodically fetch data into the database.

### Web Client

1. The web client will be accessed through a website and acts as the user's interface with our system.
2. The web client will fetch and send data to the server using HTTP REST requests.
3. The client will interpret these requests and display them to the user via the MVC model.

### Server

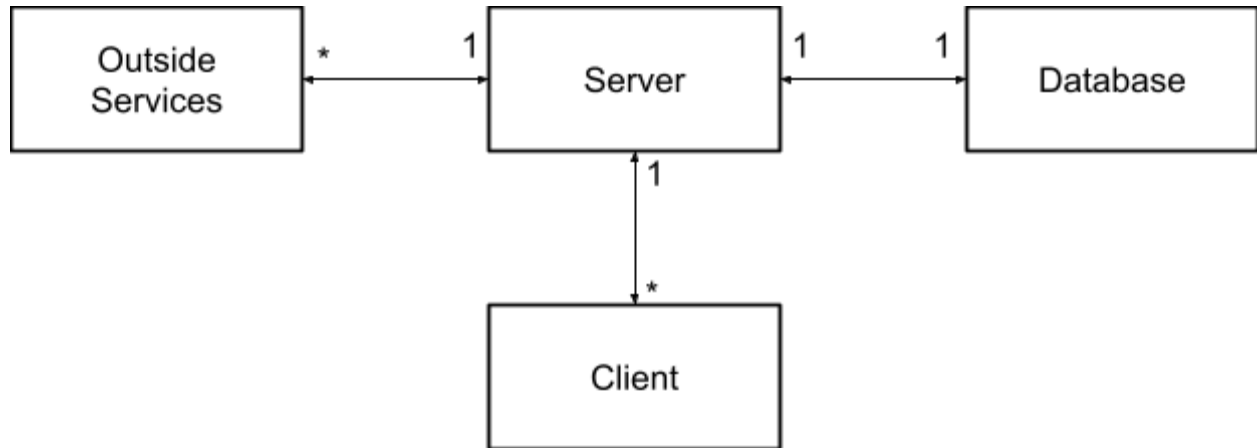
1. The server will receive and process requests from clients.
2. The server will verify client requests and respond with the corresponding data from the database.
3. The server will also periodically fetch data from outside services to keep data in the database up-to-date.

### Database

1. A Mongo Database will store data such as user plan of study, major/minor degree requirements, and course data.
2. The database will be referenced when generating degree plans and displaying course information.
3. Because the course information will likely vary in format (ex. prerequisites are only sometimes present), our database will be NoSQL and won't follow a set schema.

### Component Interaction

The client will update the UI using data provided from the backend. The clients will communicate with the backend server, and the backend server will respond to requests with data from the database. The server will also periodically update the database using data fetched from outside services.





# Design Issues

## Functional Issues

1. Should we have user accounts to store user information?
  - a. Option 1: Users are required to create an account to store information
  - b. Option 2: Users are not required to create an account, but if they do not create an account then their data will only be stored locally.
  - c. Option 3: Users cannot create an account, data is always stored locally.

Decision: Option 2; using accounts is the easiest way to store user information, especially across different sessions and browsers. However, requiring users to create accounts may be too limiting, especially for users who only want to quickly use the service. Not having user accounts would make it hard to store persistent degree information. Allowing user accounts but not requiring them is the least limiting. However, not all services may be available to users who do not create an account.

2. Should transfer credits such as AP courses, dual credit, etc. be supported?
  - a. Option 1: Don't add any support for transfer credits.
  - b. Option 2: Add support for AP courses
  - c. Option 3: Add support for all transfer credit including AP courses.

Decision: Option 2; adding support for AP courses, since AP courses are relatively simple to implement. Transfer credits from any source is very complicated and there are many different rules depending on where the credit is coming from. Therefore, we choose to only implement AP transfer credit which can be helpful for incoming freshmen.

3. How should user grades be handled?
  - a. Option 1: Don't have any class grade options for users.
  - b. Option 2: Allow users to input their class grades only.
  - c. Option 3: Allow users to input their class grades, and also approximate class grades using a grade calculator.

Decision: Option 3; grade support is important since many class prerequisites also require a specific passing grade (which can change depending on the major). Additionally, implementing a grade calculator would not be too complex and would provide more utility compared to class grades only.

4. Should course information also include semester-specific information such as the semesters a course is offered?
  - a. Option 1: Do not include any information.
  - b. Option 2: Include usual semesters in description.
  - c. Option 3: Include historical data on course scheduling

Decision: Option 3; Having historical data for courses provided, the professor that semester, and section data can give users valuable information on a course and allow them to anticipate what may happen in future semesters.

5. Should course data update automatically?
  - a. Option 1: Manually enter all course information.
  - b. Option 2: Automatically load course data from the Course Catalog
  - c. Option 3: Automatically load data on individual sections

Decision: Option 3; If the data is accessible, it would be very useful to students to have access to course history when selecting their courses per issue 4, and adding this data manually would be unsustainable for a University-wide scale.

## Non-Functional Issues

1. What language should we use for our backend server?
  - a. Option 1: Python
  - b. Option 2: JavaScript
  - c. Option 3: Java

Decision: Option 2; Python and JavaScript are more flexible since they are script-based programming languages and many packages already exist to do specific tasks. Since the frontend will be in TypeScript/JavaScript, a JavaScript-based backend will be potentially more interoperable.

2. What database should we use to store our data?
  - a. Option 1: A relational database such as PostgreSQL, MariaDB, etc.
  - b. Option 2: A NoSQL document database such as MongoDB
  - c. Option 3: A cloud hosted database such as Firebase

Decision: Option 2; non-relational databases are more flexible in their schema options, allowing us to update and define schemas on-the-fly. Relational databases are very strict in schemas, which can be a potential roadblock when developing and needing to change schemas. Cloud-based databases are also an option, but we have the ability to host our own database and not incur the latency penalty of contacting a database in the cloud.

3. What frontend framework should we use?
  - a. Option 1: React
  - b. Option 2: Angular
  - c. Option 3: Vue
  - d. Option 4: Some other framework (Svelte, etc.)

Decision: We should use the ReactJS framework, since it is very well-documented and relatively easy to learn. Considering that our front-end developers all have varying levels of experience with frameworks (and they have knowledge relevant to niche frameworks), we prioritized usability when selecting this framework. React is a very modular framework, so it would be easier for multiple front-end developers to work with simultaneously. The framework is particularly good for website development, which is our focus in this project.

4. What language should we use for our frontend?
  - a. Option 1: JavaScript
  - b. Option 2: TypeScript

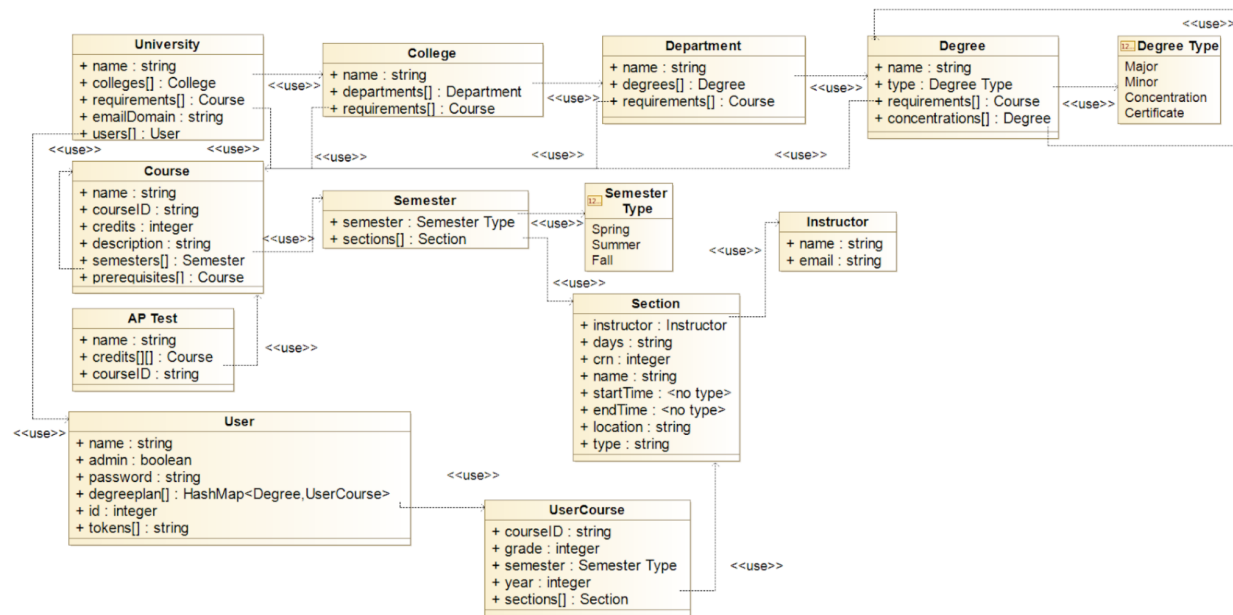
Decision: Option 2; TypeScript provides advantages to JavaScript such as type-checking, which can prevent bugs during development. TypeScript is also easily-interoperable with JavaScript.

5. How should the frontend interact with the backend?
  - a. Option 1: REST API
  - b. Option 2: GraphQL API

Decision: Option 1; REST APIs are relatively simple to implement, yet flexible in what they can do. They can also be interacted with relatively simply, while GraphQL requires more complex queries with a more complex backend.

# Design Details

## Data Class Diagram



## Class Description and Interactions

The classes are based on the organization of their counterparts within the university and keeping things modular with dependence on parents, and represents how they will be stored within our database.

1. University
  - a. The University class represents the entire university (e.g. Purdue University)
  - b. Each University references multiple Colleges (e.g. Purdue has College of Science, College of Engineering, etc.)
  - c. Universities also can reference requirements (e.g. the university core course requirements)
  - d. Universities reference Users objects, who are the students that attend that university.
2. College
  - a. The College class represents a specific college inside a university.
  - b. Each College references multiple Departments (e.g. the College of Science has the Computer Science and Math departments)
  - c. Colleges can also reference specific requirements for students inside their College.

### 3. Department

- a. The Department class represents a specific department in the College class
- b. Each Department instance references multiple Degree objects (e.g. the Computer Science Department offers three degrees: Computer Science, Data Science, and Artificial Intelligence)
- c. The Department class will reference an array of Course objects, which represent the course requirements associated with that specific department. (e.g. the Computer Science department requires that all its students take CS 180)

### 4. Degree

- a. The Degree class represents a specific degree within the Department Class
- b. The Degree class has an important enum field 'type' that specifies what type of degree it represents: major, minor, concentration, or certification
- c. Each Degree instance references multiple Course objects, which represent the course requirements needed for that specific degree (e.g. a Computer Science (major) degree requires students to take: CS 180, CS 182, CS 240, CS 250, CS 251 and CS 252)
- d. If a degree offers concentrations/tracks, then its corresponding Degree object will contain an array of more Degree objects called 'concentrations'. The Degree objects contained within 'concentrations' will have 'type'='concentration'. (e.g. A Degree object for Computer Science of 'type'='major' would contain an array called 'concentrations' that would contain 9 Degree objects: Computational Science and Engineering, Computer Graphics and Visualization, Database and Information Systems, (Algorithmic) Foundations, Machine Intelligence, Programming Language, Security, Software Engineering, and Systems Software. Each of these 9 objects would be Degree objects with 'type'='concentration')
- e. The above 'concentrations' field is nullable, for degrees that have no concentrations.

### 5. Course

- a. The Course class represents a specific course offered at a university, and has fields for the unique identifiers of that course.
- b. The Course class has many associations: Course objects are contained in all University, College, Department, and Degree objects in an array labeled 'requirements'.
- c. Course objects contain an array called 'semesters' that contain Semester objects. These Semester objects represent the semesters during which the course is offered. (e.g. a course that is offered Fall, Spring, Summer would contain a 'semesters' array with three Semester objects)
- d. The APTest Class also contains Course objects, since each AP Test has equivalent courses that it counts towards

## 6. Semester

- a. The Semester class represents a single term (summer, spring, fall)
- b. Each Semester object holds an array of Section objects. These Section objects represent each section of that class that is offered. (e.g. CS 307 would be represented by a Course instance that has a Semester object representing our current spring semester. Within that Semester object, there would be a Section object for each occurrence of that class, such as the MWF 1:30-2:20 PM section that we are enrolled in)

## 7. Section

- a. The Section class represents specific occurrence of a class (e.g. the CS 307 section that happens on MWF 1:30-2:20 PM and is taught by Professor Turkstra)
- b. Each Section object will contain unique identifiers such as instructors, start/end time, etc.
- c. A Section object is contained within each UserCourse class to capture the specific section that the user is enrolled in.
- d. An array of Section objects is contained within each Semester object to represent all of the different sections of a course available for that particular semester
- e. The Section class also contains an Instructor object, which represents the professor for that class

## 8. Instructor

- a. The Instructor class represents a specific professor and contains identifiers for them
- b. Instances of Instructor are located in Section objects

## 9. APTest

- a. The APTest class represents a specific AP Exam offered by College Board (e.g. an APTest object for 'AP Biology' would
- b. Each APTest object contains a 2D array called 'credits'. The 0th indices of 'credits' contain an int value between 1 and 5, which represents an AP score. The 1st indices of 'credits' contain Course objects. (e.g. for an APTest object representing AP Biology, there would be a 5 x 2 array. Because receiving a 5 in AP Bio excuses you from BIO 110 and BIO 111, the 'credits' array would have credits[5] = [<Course object of BIO 110>, <Course object of BIO 111>]

#### 10. User

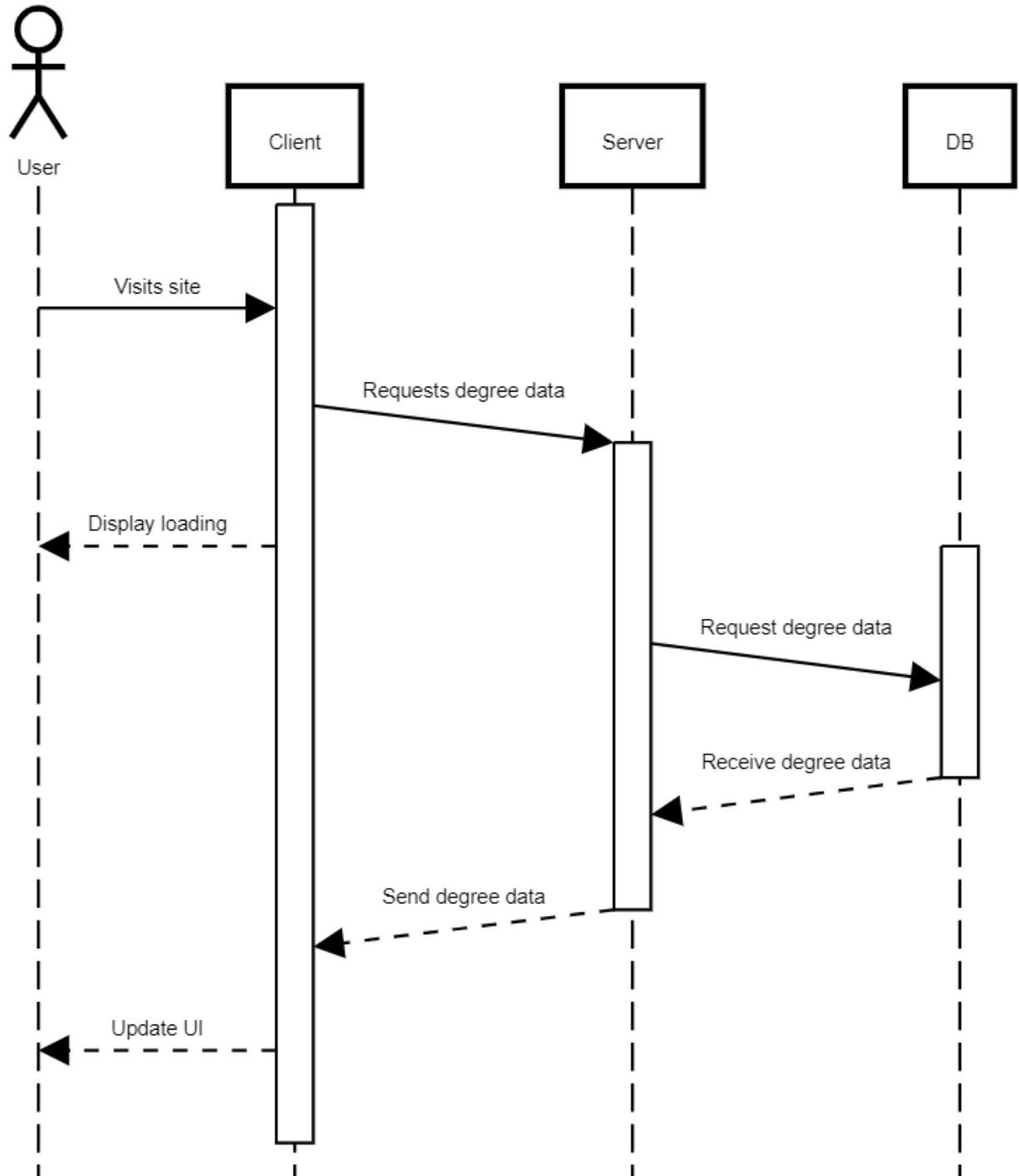
- a. The User class represents a specific student/admin who is using our application.
- b. Many User objects are contained within each University class object.
- c. The User class contains a field called 'degrees' with Degree objects. This represents any majors, minors, or certifications that a user is enrolled in. (e.g. a User who is studying CS will contain a Degree object for CS in their 'degrees' array.
- d. User class contains an array of UserCourse objects called 'courses', which represent the courses that a student is/was/will be enrolled in.

#### 11. UserCourse

- a. The UserCourse class represents a specific course that the user is enrolled in.
- b. Each UserCourse will have a Semester object called 'semester' which will represent when this course is being taken.
- c. Each UserCourse object will have a Section object called 'section' which will represent the specific section of the course (ex. MWF 1:30-2:30 PM section of CS 307)

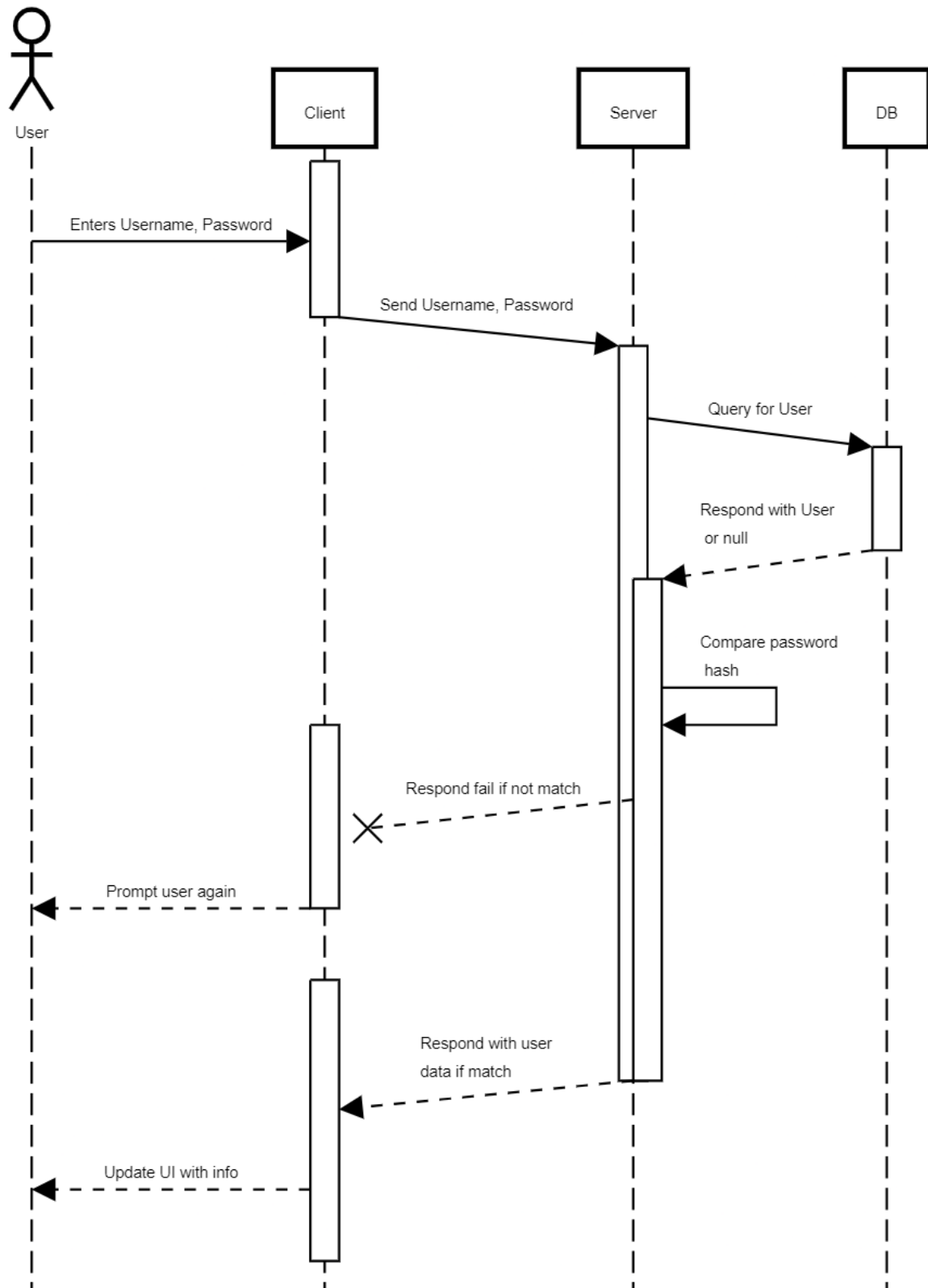
## Sequence Diagrams

User visits site

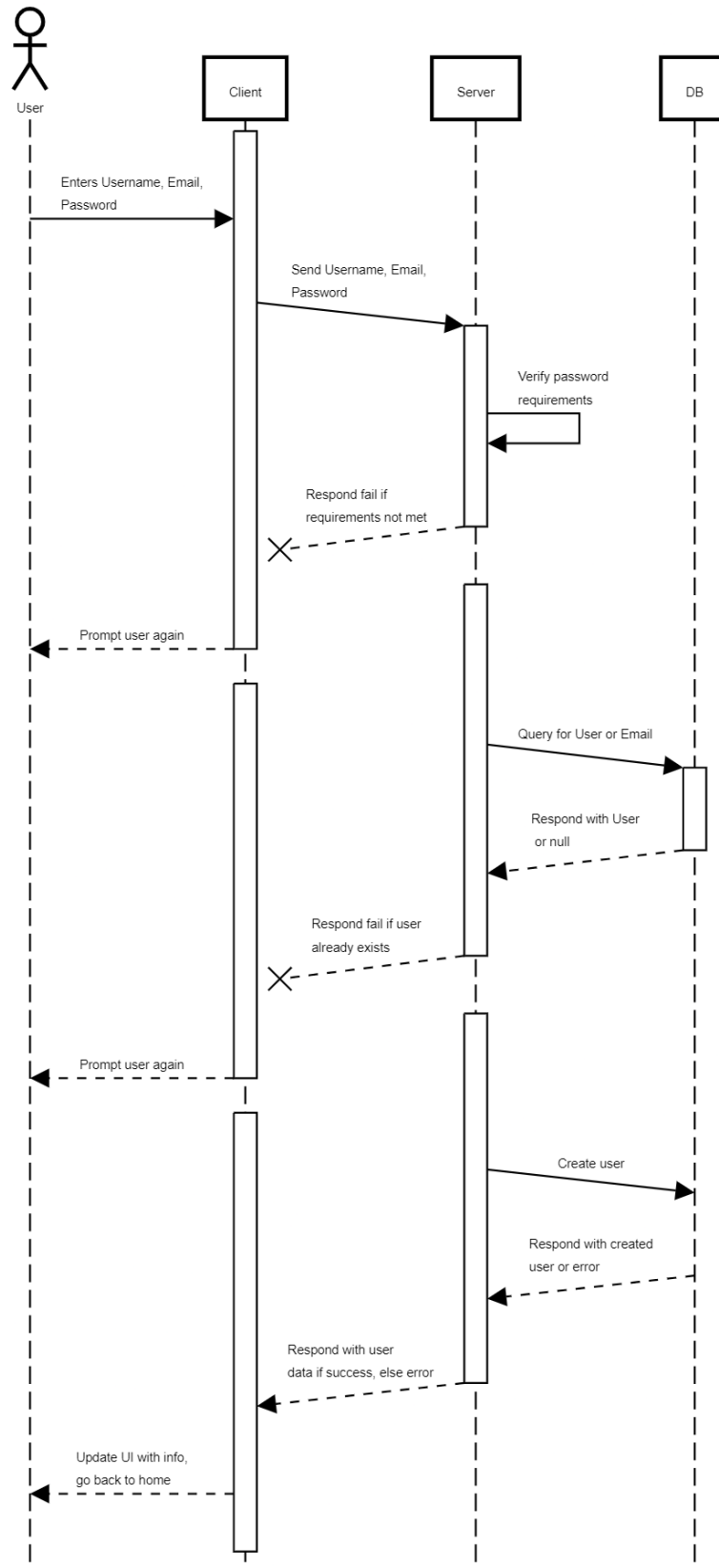




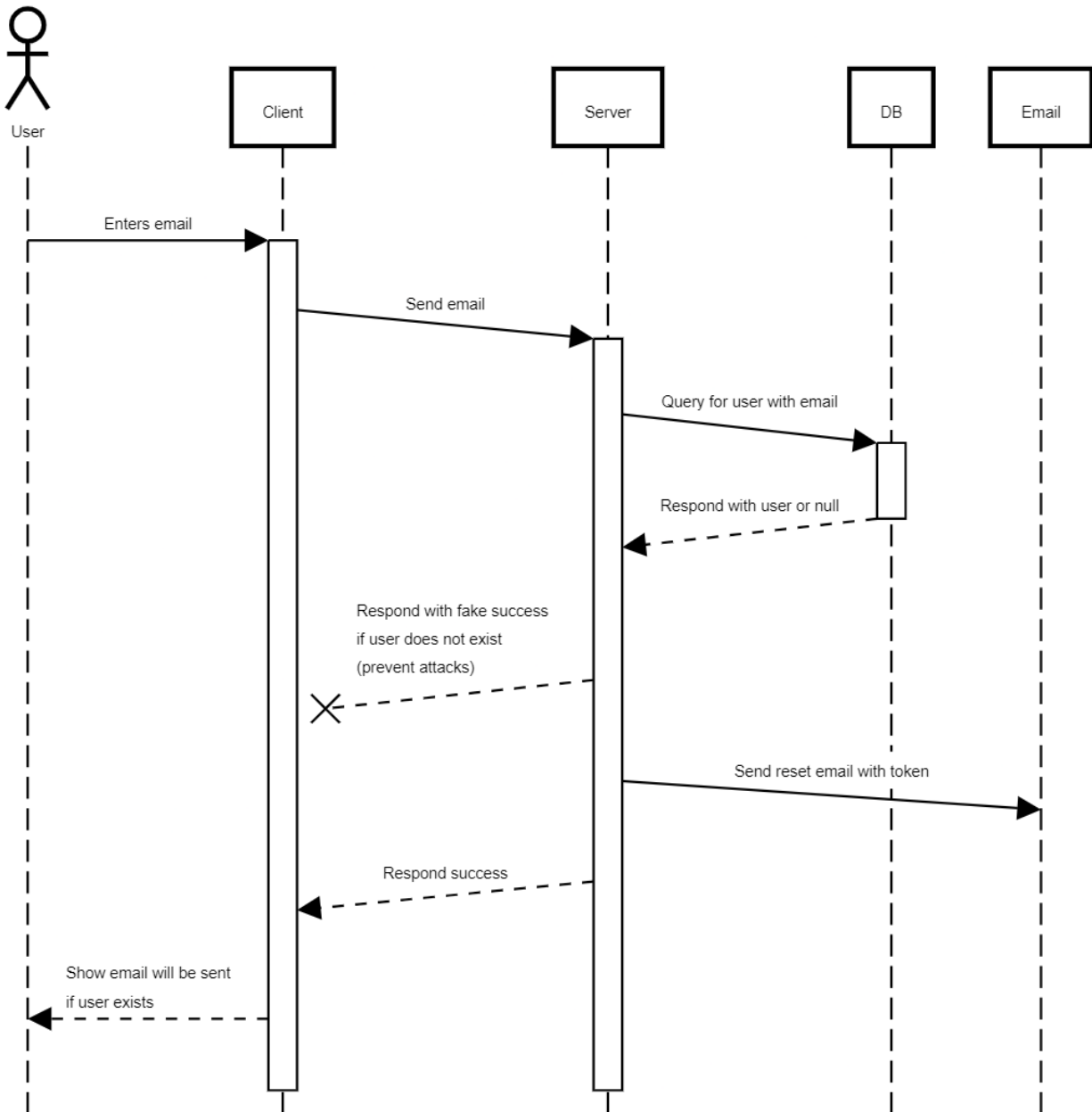
## User log-in



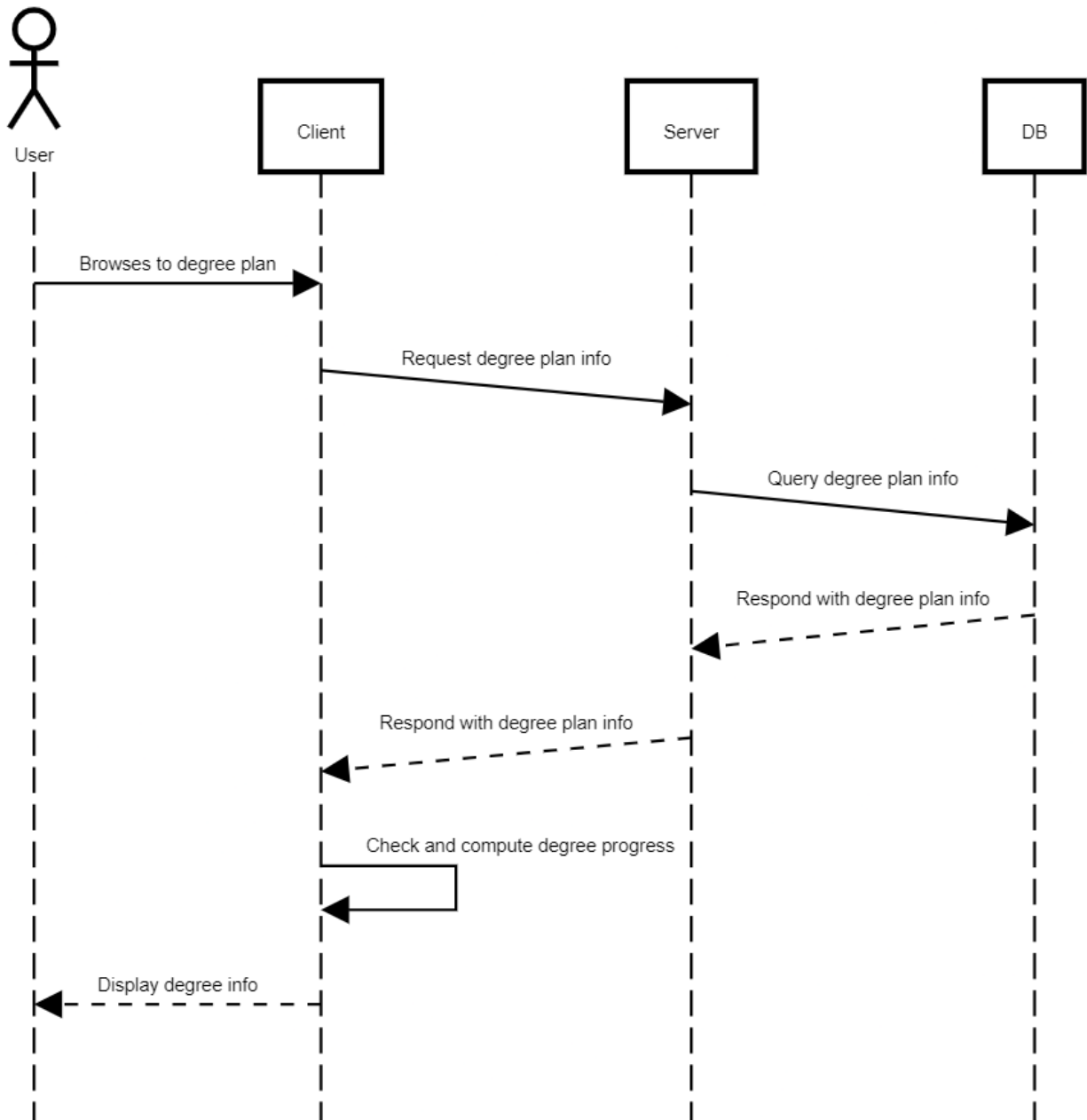
## User register



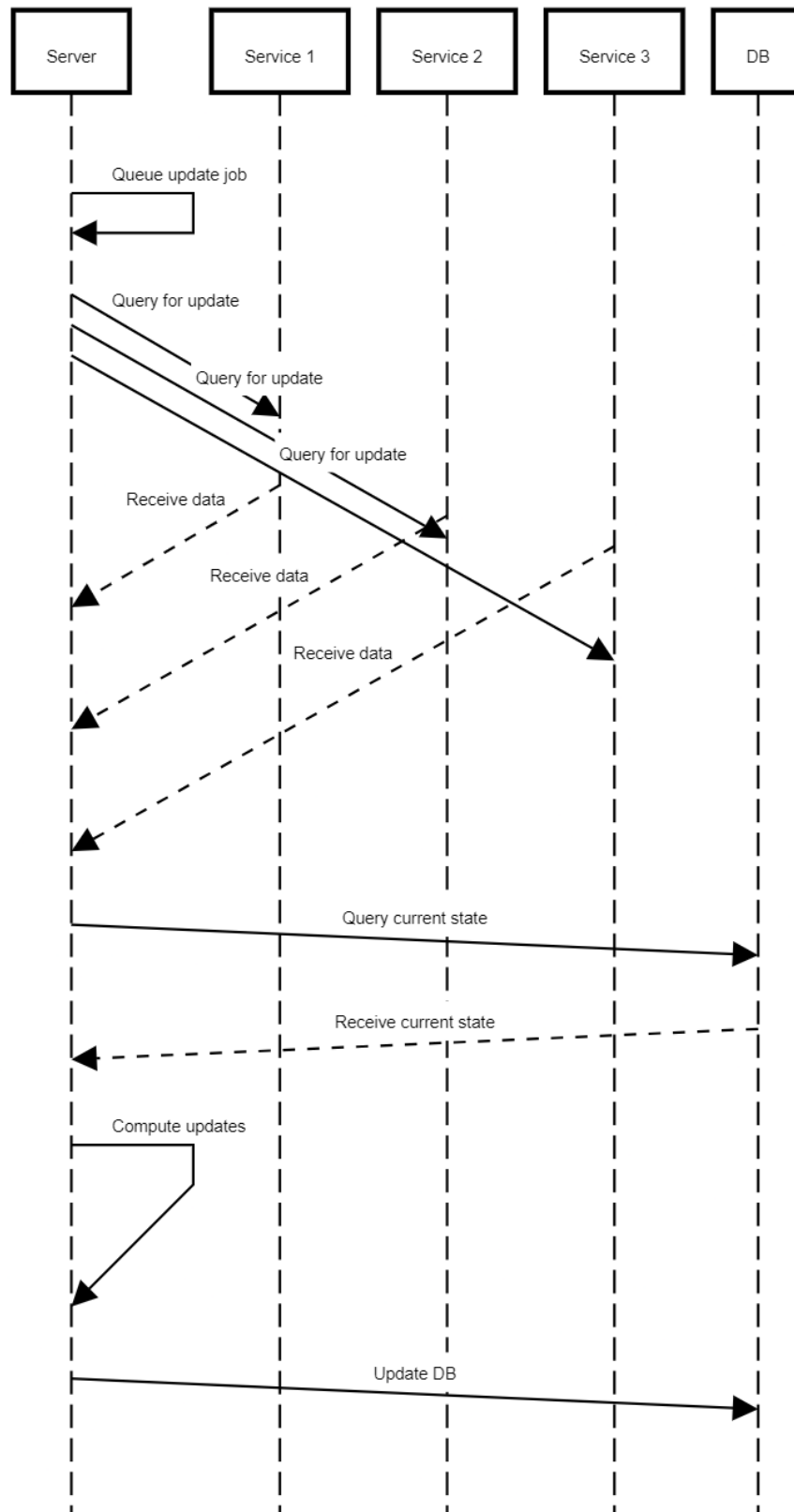
## User resets password



## User views degree info

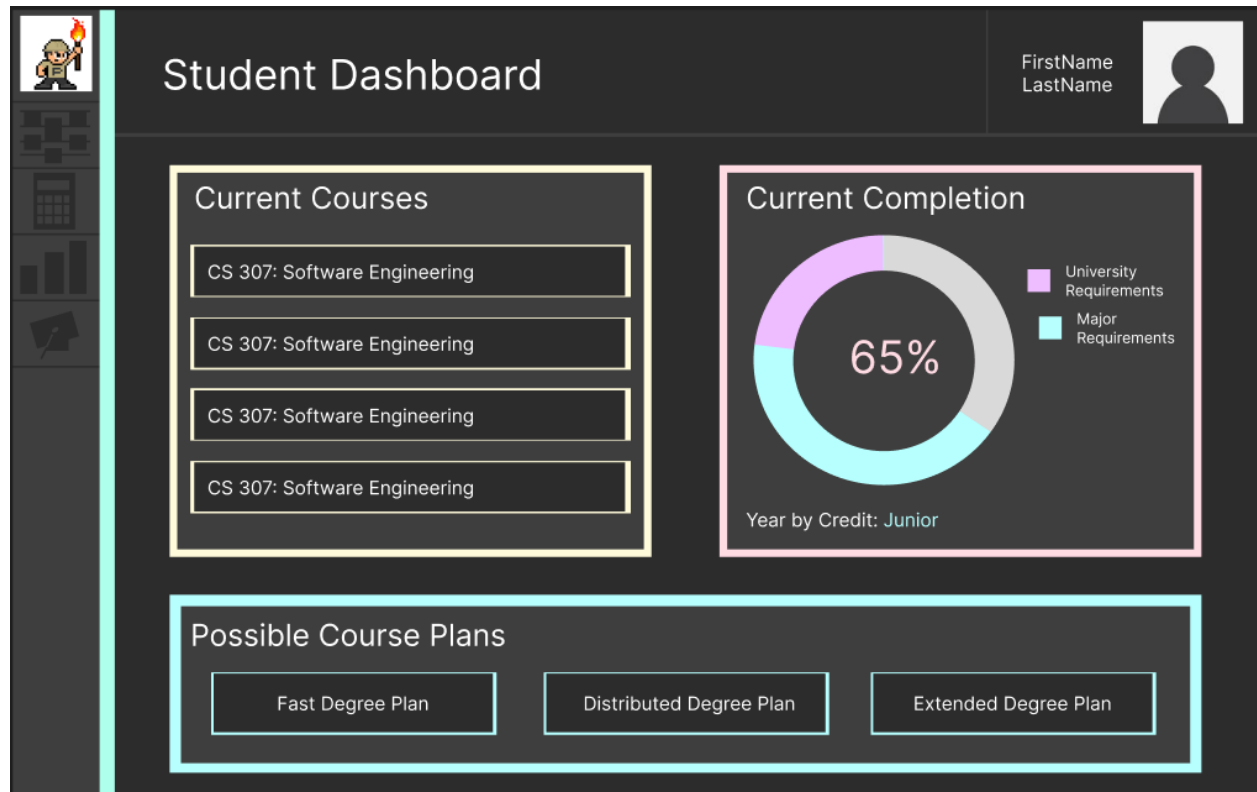


## Server updates course info




## UI Mockups

### Landing Page (After User Login)








## Student Requirement Page




## Student Requirements

FirstName  
 LastName













University Requirements ▼  
 College Requirements ▼  
 Major Requirements ▼  
 Minor Requirements ▼



## Student Requirements

FirstName  
 LastName




Major Requirements ▼

Laboratory Sciences  
 Information Literacy  
 Oral Communication  
 Written Communication  
 Quantitative Reasoning  
 Human Cultures: Humanities

**BIOL 110000: Fundamentals of Biology I**  
 Credit Hours: 4.00. This course is designed primarily to provide an introduction to the principles of biology for students in agriculture and health sciences. Principles of biology, focusing on diversity, ecology, evolution, and the development, structure, and function of organisms.


BIOL 111000: Fundamentals of Biology II  
 BIOL 112000: Fundamentals of Biology I  
 BIOL 113000: Fundamentals of Biology II  
 BIOL 12100 Biology I: Ecology, Diversity, & Behavior  
 BIOL 14600 Introduction to Biology




## Grade Calculator Page



## Student Grade Calculator

FirstName  
LastName






### Category Breakdown

Add Category


### Assignment Grades




Add Grade



## Student Grade Calculator

FirstName  
LastName





### Category Breakdown

Homework	10%
----------	-----

Add Category

Class Name: ENGL 207

### Assignment Grades

Essay 1	92%
---------	-----

Add Grade