

**Team Members:**

Saaketh Jasti

Divya Karivaradasamy

Syed Obaid Quadri

Leonardo Acioli A. S.

**Delivered To:**

Dr. Klyne Smith

March 2, 2021



# Table of Contents

Table of Contents..................................................................................................................................................... 2

Introduction............................................................................................................................................................... 3

Requirements............................................................................................................................................................ 4

System Modeling...................................................................................................................................................... 5

Wire Frames.............................................................................................................................................................. 6

Architecture Design................................................................................................................................................. 7

Implementation........................................................................................................................................................ 8

Software Testing....................................................................................................................................................... 9

Deployment............................................................................................................................................................ 10

Glossary of Terms................................................................................................................................................... 11

# Introduction

Single Syllabus is a web application which allows users to synthesize multiple syllabi into one syllabus.

This document will present the solution engineered by The Developers for the entire software lifecycle of Single Syllabus. The audience for this document is Dr. Klyne Smith. The following presents the structure of the document:

* ***Requirements:*** High-level overview of features, requirements elicitation process, functional and non-functional requirements
* ***Design:*** Use case and sequence diagrams for major features, site map
* ***Wire frames, architecture design, and implementation***
* ***Testing:*** Researched methods and technologies, testing plan, functional and non-functional test cases, Requirements Traceability Matrix
* ***Software evolution and glossary of terms***

# Requirements

# **Requirements Overview**

The web application shall allow college students to create a single, consolidated syllabus out of their many syllabi for a semester. The user shall upload a syllabus for each of their classes in PDF format. The app will analyze each syllabus for key sections - such as "Quizzes”, “Assignments”, and “Textbooks” - and return the extracted data, aggregated across all classes, to the user. The user shall be able to edit the extracted data. Finally, the user shall be able to generate a single syllabus from the edited data for sharing and downloading. Additionally, the user will be able to view the extracted data from all semesters in a “Classes” page.

# **Requirements Gathering Process**

First, we deliberated what screens users needed. To do this, we viewed the app from the perspective of our target audience (students) and asked ourselves what the ideal experience would look like. We decided on building pages for the following high-level functionalities:

* ***log-in/sign-up***
* ***uploading syllabi***
* ***viewing and editing the extracted data***
* ***viewing/downloading/sharing the consolidated syllabus***
* ***viewing past syllabi and class data***
* ***viewing account information***

To create functional requirements, we then systematically discussed what features were needed per screen.

To create non-functional requirements, we browsed educational websites such as eLearning and took inspiration from some visual elements. For example, the ‘?’ (help) and ‘Contact Us’ buttons in our navigation bar were drawn from favorite websites.

# **Functional Requirements**

***General***

1. The user shall access Single Syllabus through a domain name. (Syed)
2. The user shall navigate through the app using a navigation bar with buttons. (Saaketh)
3. Each page shall include a navigate to next page button which directs the user through the flow.

***The user shall use the navigation bar to...***

1. Get help using our service by clicking on the ‘?’ button. (Syed)
2. Read about our project and team by clicking on the ‘About’ button. (Divya)
3. Go to the classes tab where they can view each section of the single syllabi separately. (Saaketh)
4. View their account details, including their username and past syllabi, by clicking on the user icon. (Saaketh)
5. Go to the upload syllabus page (Saaketh)
6. Learn where to submit questions, comments, and concerns by clicking on the ‘Contact’ button. (Leonardo)
7. Sign out of the app by clicking on the ‘Sign Out’ button. (Leonardo)

***Landing Page***

1. On the landing page, the user shall view a card with options to log into or sign up for an account. (Saaketh)
2. The user shall sign up for an account by creating a username and password. (Saaketh)
3. The user shall log into their account by entering their username and password. (Saaketh)
4. The user should have the option of signing up or logging in through a 3rd-party service such as Google or Apple. (Saaketh)
5. The software should validate usernames and passwords from log in. (Saaketh)
6. The software should issue an error if the username and/or password are invalid. (Divya)
7. The software should store all account information in a secure database. (Leonardo)

***Upload Syllabi Page***

1. The user shall click on an upload button to begin uploading syllabi. (Saaketh)
2. After clicking the upload button, the user shall select multiple files from their computer through a browser pop-up. (Syed)
3. If the user selects a non-PDF document, the software shall reject the upload and the website shall display an error. (Syed)
4. The user shall be able to remove certain files from the list. (Leonardo)
5. The user shall click the ‘Analyze’ button to continue to the Card View page. (Divya)

***Card View/Edit Page***

1. The user shall view cards which contain aggregated information across each class syllabus. Each card shall contain one section of information. (Divya)
2. The user shall edit the aggregated information by clicking on the ‘Edit’ button of a card. (Divya)
3. The user shall show/hide a card by clicking on the card’s ‘Show/Hide’ button to specify which sections (ex: quizzes, assignments, office hours) they want in the syllabi, so that the final syllabus does not contain extraneous information. (Divya)
4. The user shall click the ‘Generate’ button to generate a pdf from the extracted data. (Divya)

***Backend***

1. The software shall parse the PDFs for sections (ex. textbooks, quizzes, assignments, office hours, etc.) and return extracted data to the frontend. (Saaketh)
2. The software shall update the section info if the user makes edits. (Saaketh)
3. The software shall generate a PDF from the edited sections. (Saaketh)
4. The software shall store generated PDFs with the user information. (Saaketh)

***Syllabus Preview Page***

1. The user shall view the generated PDF in a scrollable document view. (Divya)
2. The user shall download the generated PDF by clicking on a ‘Download’ button. (Saaketh)
3. The user shall share the PDF via mail by clicking on the ‘Share’ button. (Leonardo)
4. The user shall save the PDF to their account clicking on the ‘Save’ button. (Leonardo)
5. The user shall discard PDF parsed by clicking on the ‘Discard’ button. (Leonardo)
6. The user shall be able to go able to go to the classes page from the preview page. (Leonardo)

***Classes Page***

1. The user shall tab through semesters to view prior syllabi. This can be achieved through a tab element with lists of document names. (Saaketh)
2. The user shall view textbooks, assignments, office hours, and other extracted data from prior classes. Each category shall have a separate tab. (Divya)

***Account Details***

1. The user shall be able to view their username. (Divya)
2. The user shall be able to change their password. (Saaketh)

# **Non-Functional Requirements**

***Development***

1. The software shall be supported on Google Chrome 87, Safari 10.11, Microsoft Edge 10, Firefox 83, Opera 60. (All)
2. The software will be initially developed for desktop. After this, we should expand to tablets and smartphones. (Leonardo)

***Appearance***

1. The landing page should display the Single Syllabus logo. (Divya)
2. Each page shall have a title describing its purpose. (Divya)
3. Each page shall have buttons in the lower right corner navigating the user to the next step. (Divya)
4. All buttons should be rounded in appearance. (Leonardo)
5. The app should have a color scheme consisting of yellows, oranges, whites, greys, and blacks. The color scheme should be consistent from page to page. (Leonardo)
6. The app should use a single font. All titles and buttons should have consistent fonts, sizes, and proportions. (Divya)
7. The website should be easily operable and intuitive to an average college student. During user testing, at least 9/10 users should complete the flow without errors and have no questions. (Saaketh)

***Privacy***

1. Log in and syllabi information shall be stored in a secure database and not shared with any third party. (Syed)
2. There shall be no tracking done on the website. Ideally, the website should be secure. (Syed)

***Maintenance***

1. The service shall be available at least 99% of the time. (For every 100 logins, 99 shall succeed). (Divya)
2. The website should undergo maintenance every two weeks, from 4:00 AM to 6:00 AM on Sundays. (Leonardo)

***Other***

1. The software will restrict the type of files uploaded to PDFs. (Syed)
2. PDF parsing shall execute within 1 minute. (Saaketh)

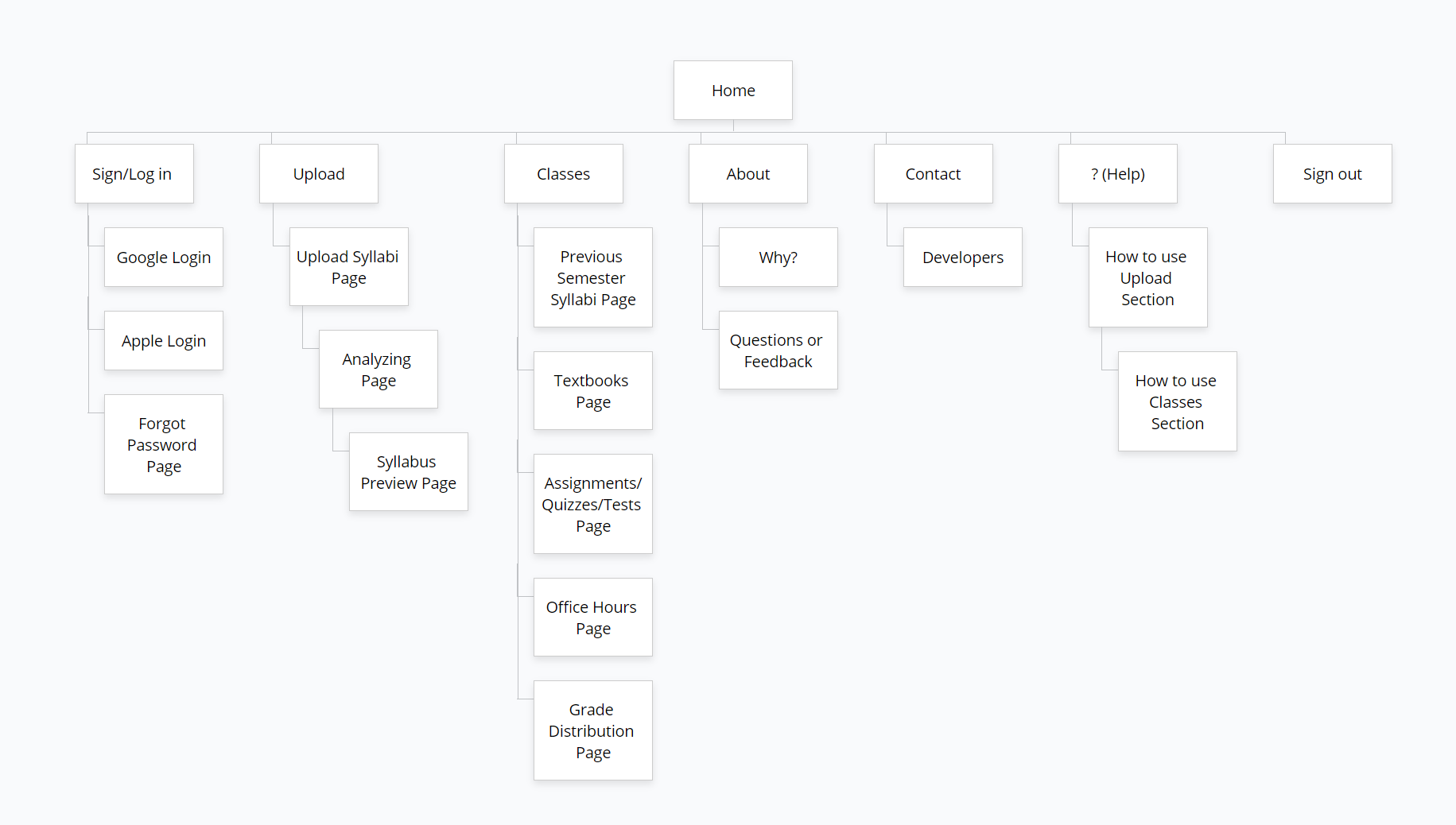
# **Out of Scope**

Any functionality not mentioned above are considered out of scope. Examples of out-of-scope activities include the following:

* Users’ ability to edit previously generated syllabi.
* Support to analyze non-PDF files.
* Users’ ability to enter custom syllabus sections.
* User’s ability to customize the style or layout of the final PDF.

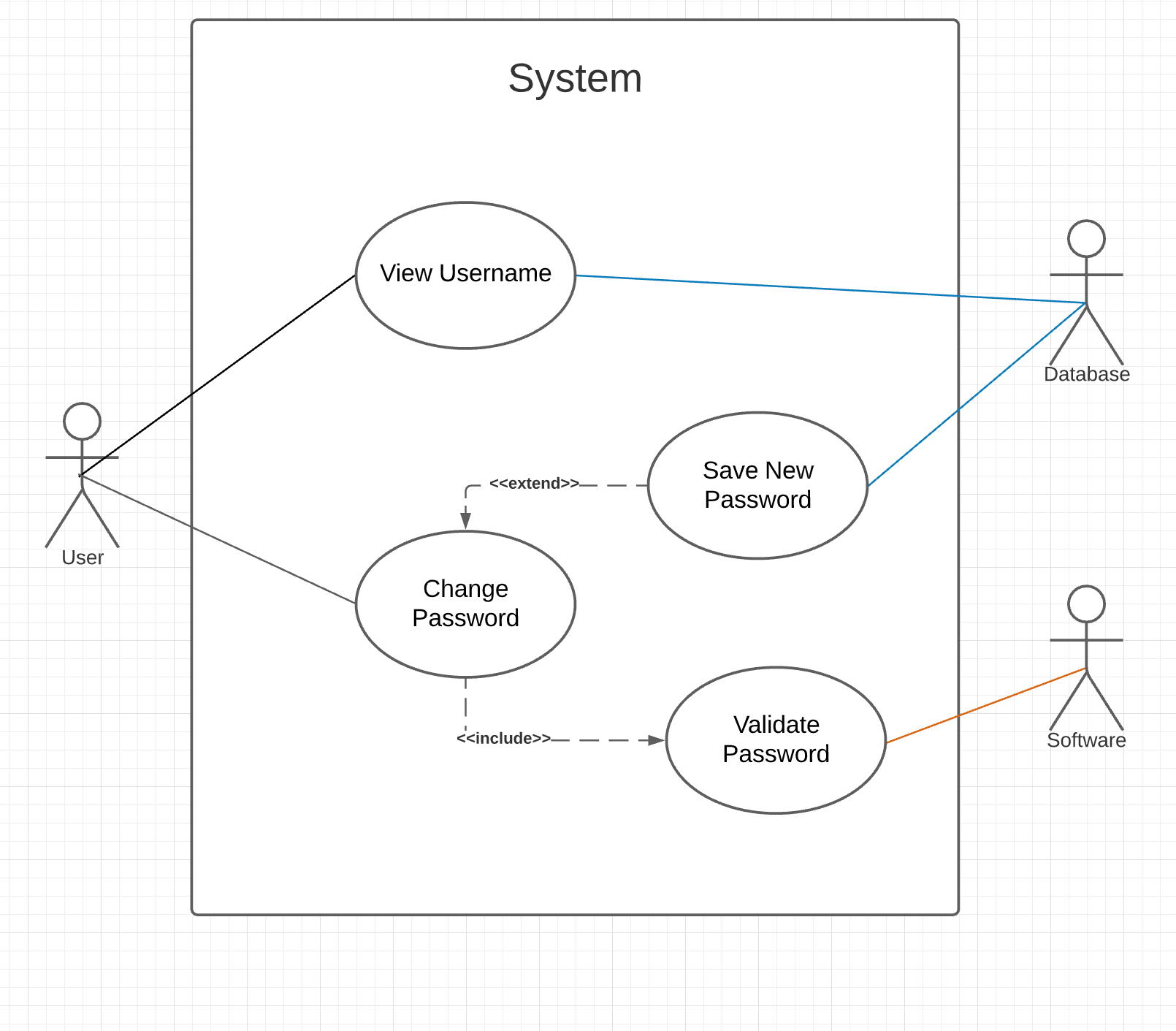
System Modeling

# **Site Map**



**Use Case Diagrams**

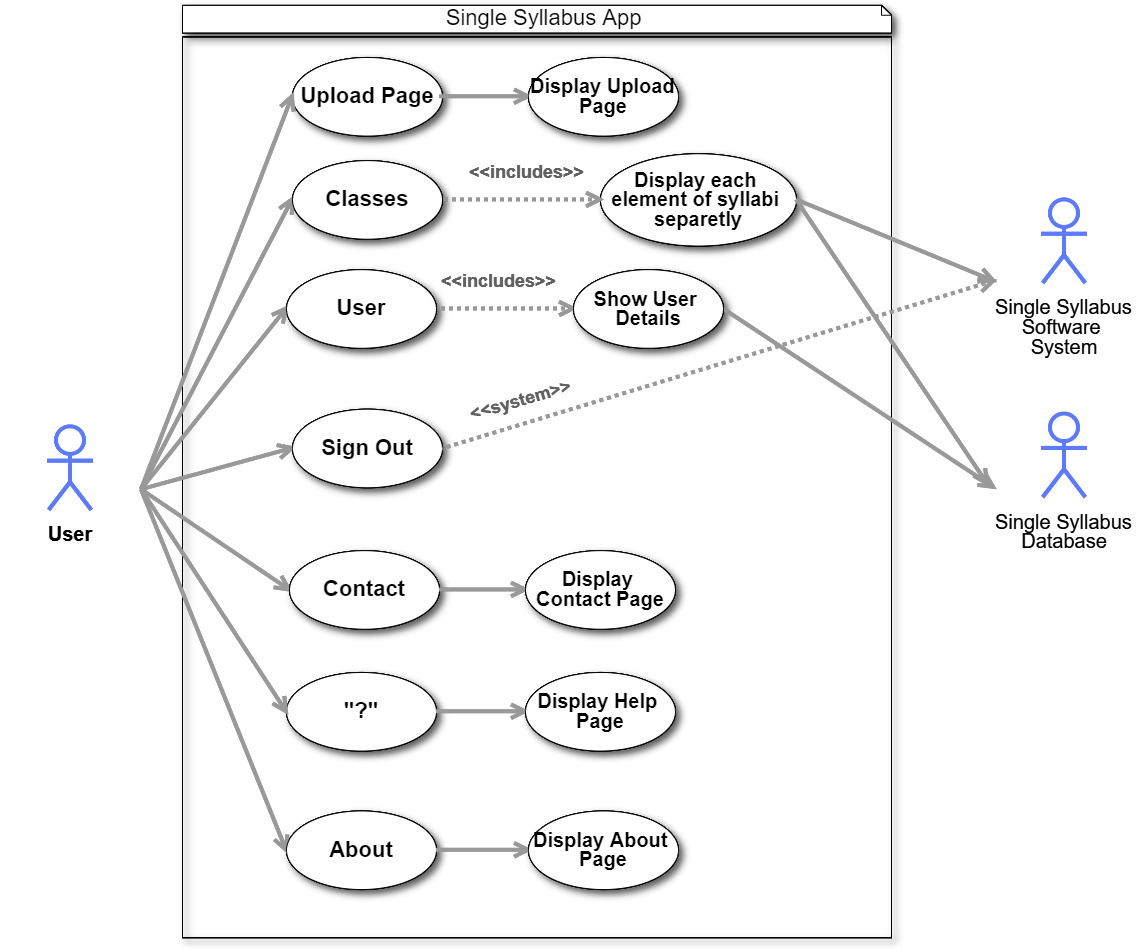
***Account Details Diagram (Saaketh)***



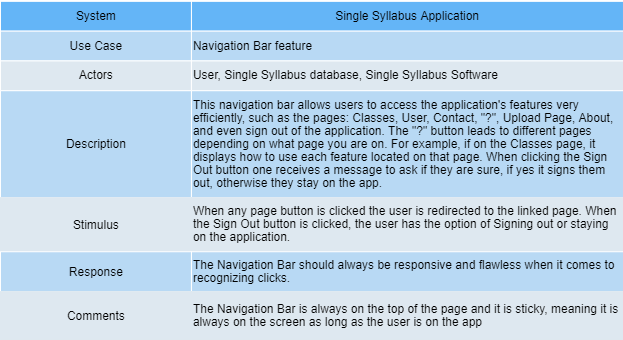
***Account Details Table (Saaketh)***

|  |  |
| --- | --- |
| **System** | **Single Syllabus Application** |
| Use case | Account Details |
| Actors | User, SS database, SS software |
| Description | The account details page allows the user to view their username and change their password in a single page. The user’s username will be the same as their email and while their password will not be displayed, for security reasons, they can change it on the screen if they want to. |
| Stimulus | The user clicks on the ‘Account’ tab in the navigation bar and this page appears. The user can then click on ‘change password’ and type in a new password. |
| Response | When the tab is clicked, the username is retrieved from the database and displayed. When the user wants to change the password and types in a new one, it is validated by the software and if it is valid, the value replaces the old password in the database. |
| Comments | The password is validated by the software on the basis that it should contain at least 1 capital letter, 1 number, and be 6 characters or longer. |

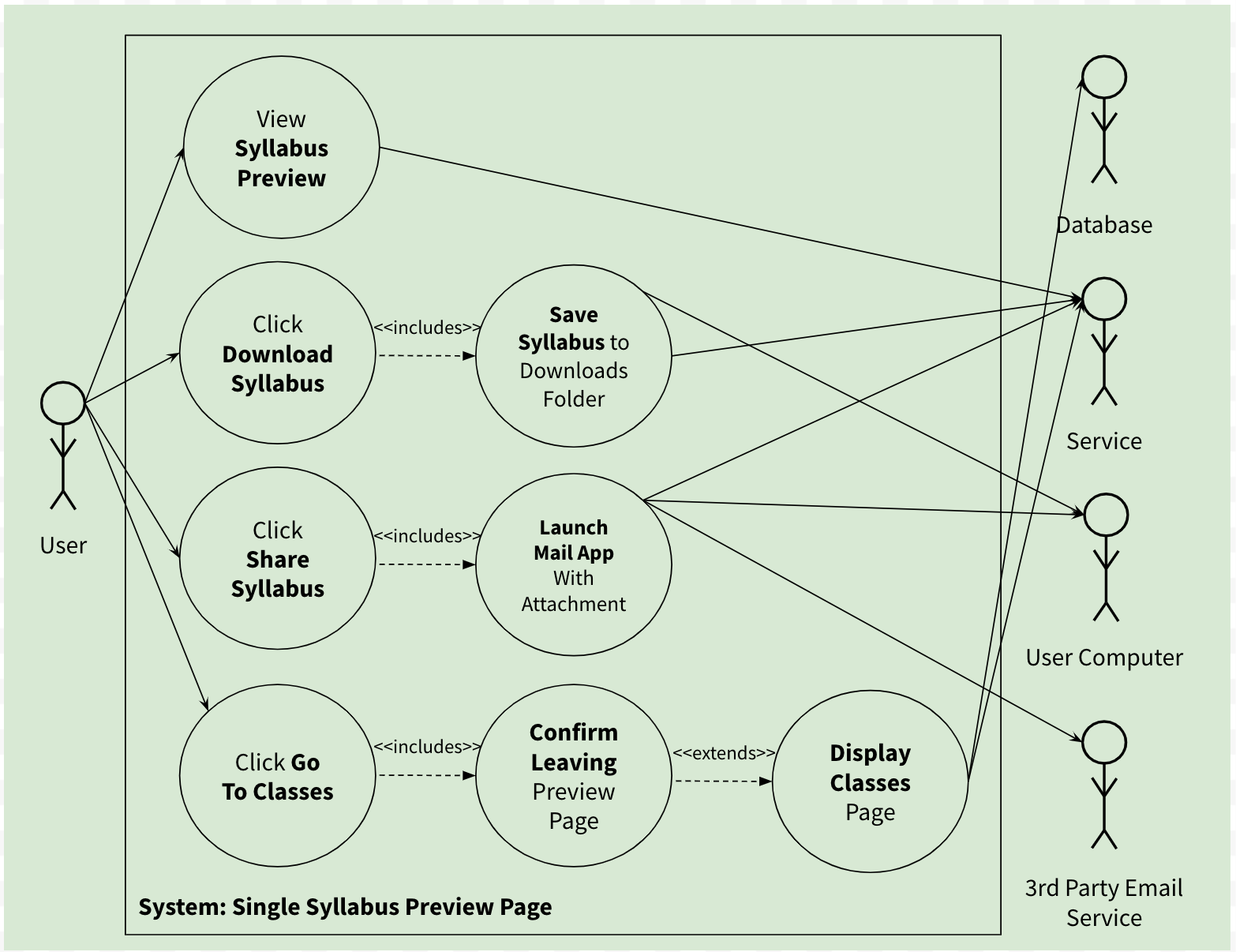
***Navigation Bar Diagram (Leonardo)***



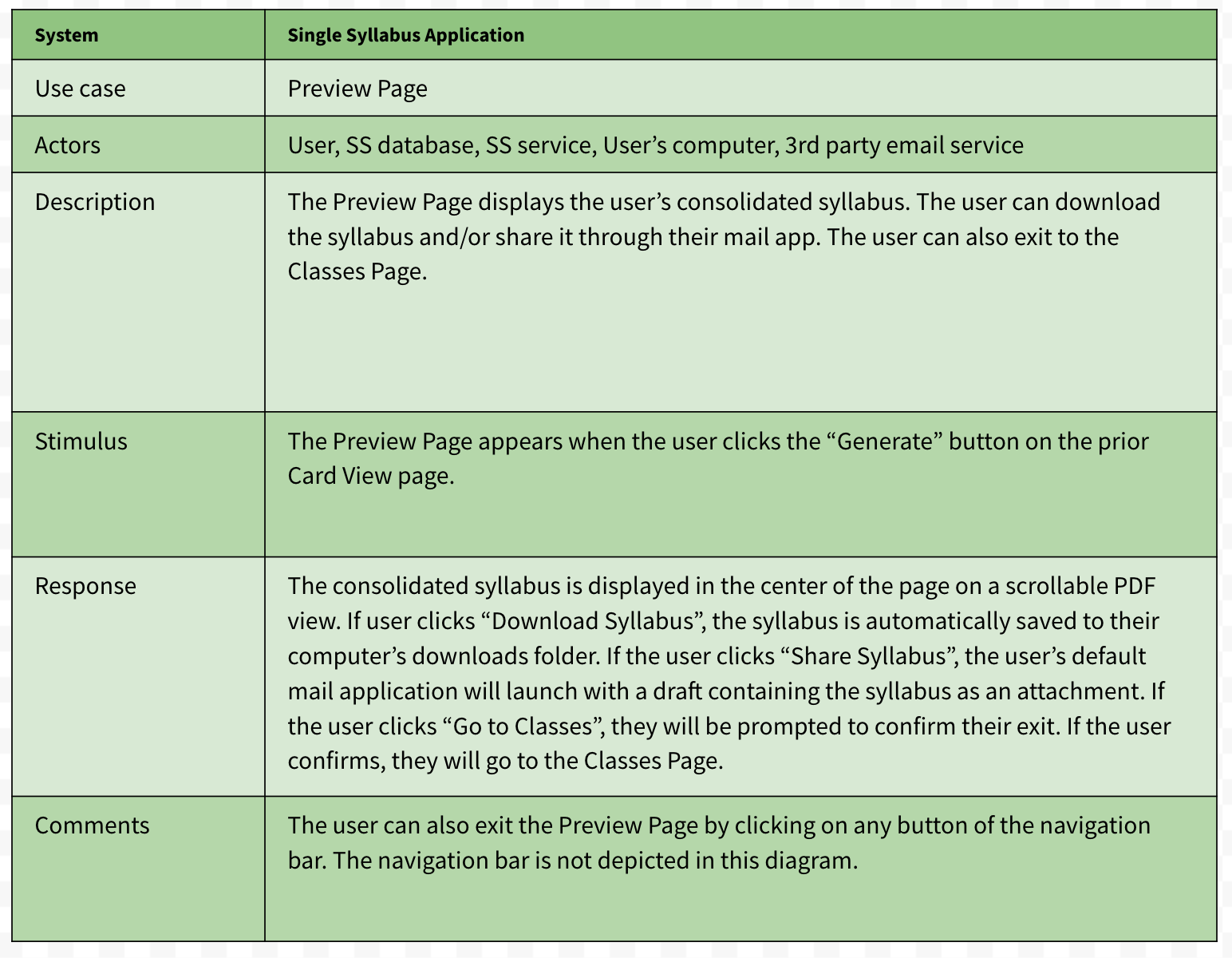
***Navigation Bar Table (Leonardo)***



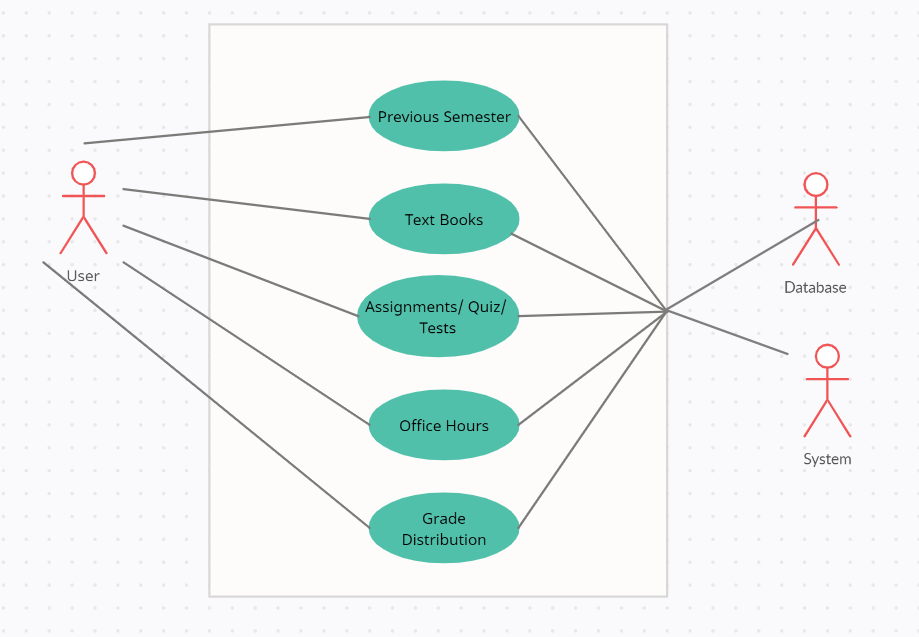
***Preview Page Diagram (Divya)***



***Preview Page Table (Divya)***



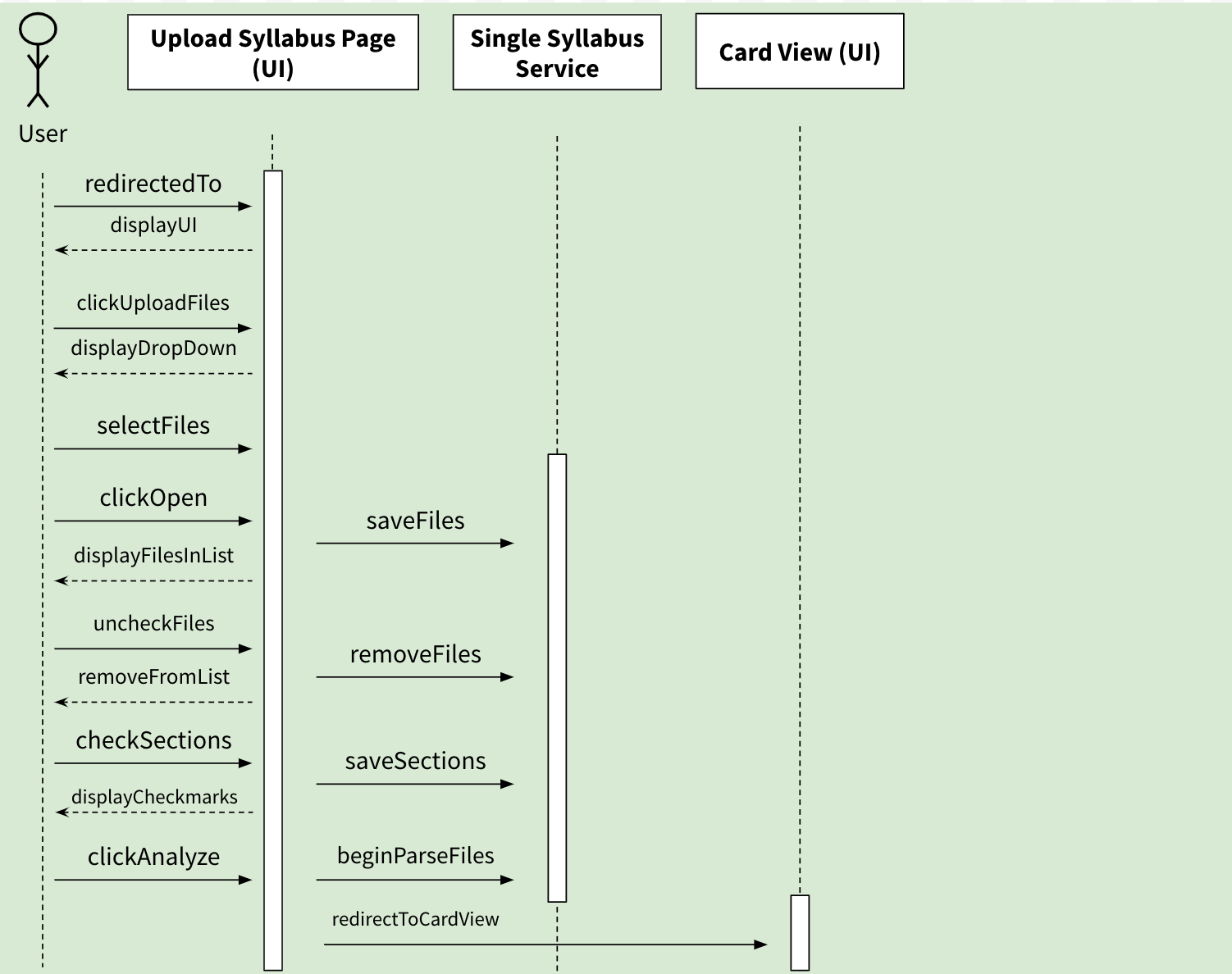
***Class Page Diagram (Syed)***



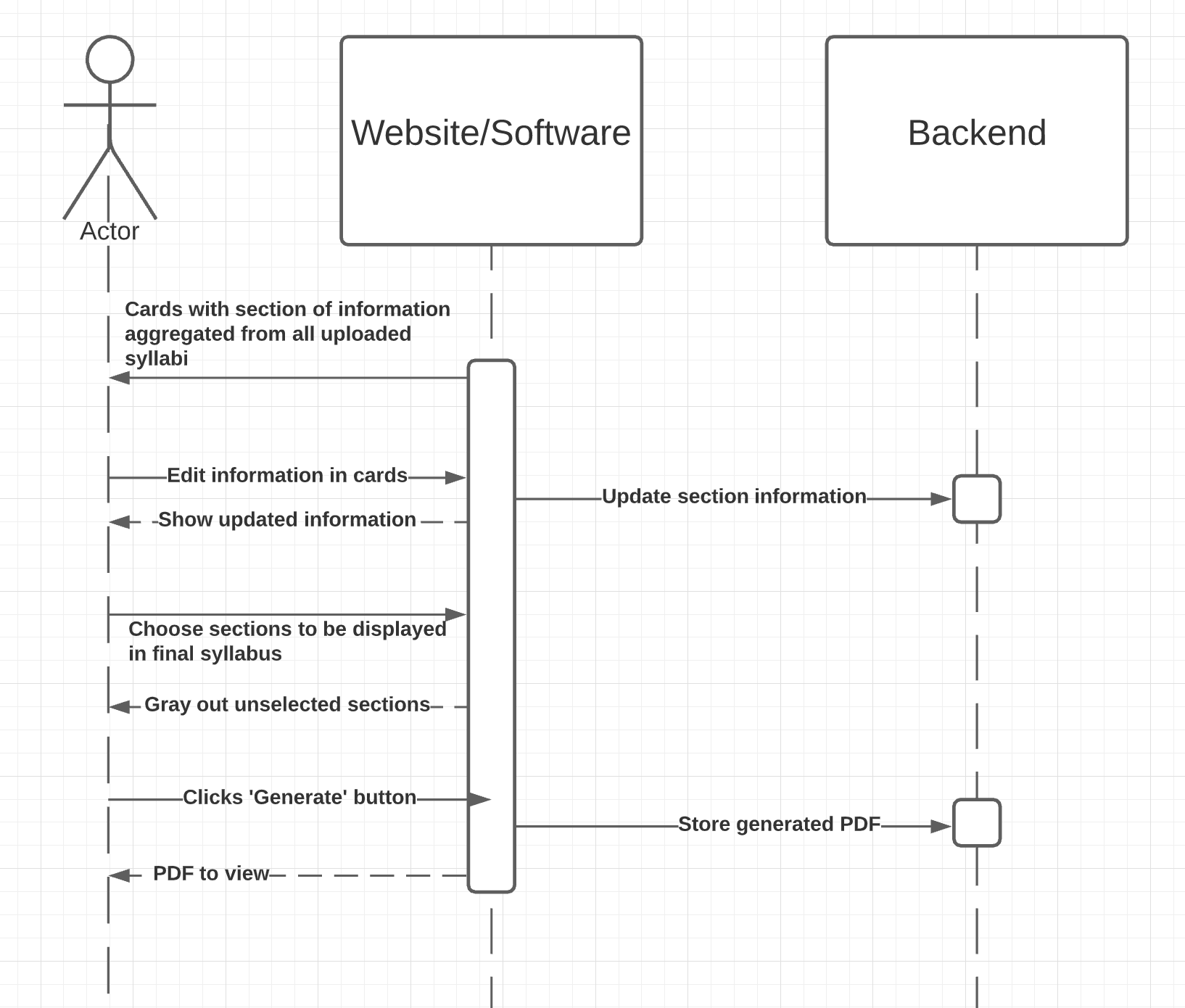
***Class Page Table (Syed)***

|  |  |
| --- | --- |
| **System** | **Single Syllabus Application** |
| Use Case | Class Page |
| Actors | User, Single Syllabus Database |
| Description | The Classes page will allow the user to view all the different sections of the uploaded syllabus/syllabi such as Office hours, textbooks, etc. This page also allows the user to show/hide certain sections. A user can view previous semesters from this page as well. |
| Stimulus | The user will click on the ‘Classes’ tab to be led to this page which will show all the different sections and their options. |
| Response | When the page is loaded, the list of sections including the unique sections in each syllabus is shown with all the necessary information. Users can choose to edit the text or show/hide a section. |
| Comments | The page will have an easy-to-use navigation tool to scroll through the page. This allows the user to quickly go to a section without needing to scroll through the whole page. |

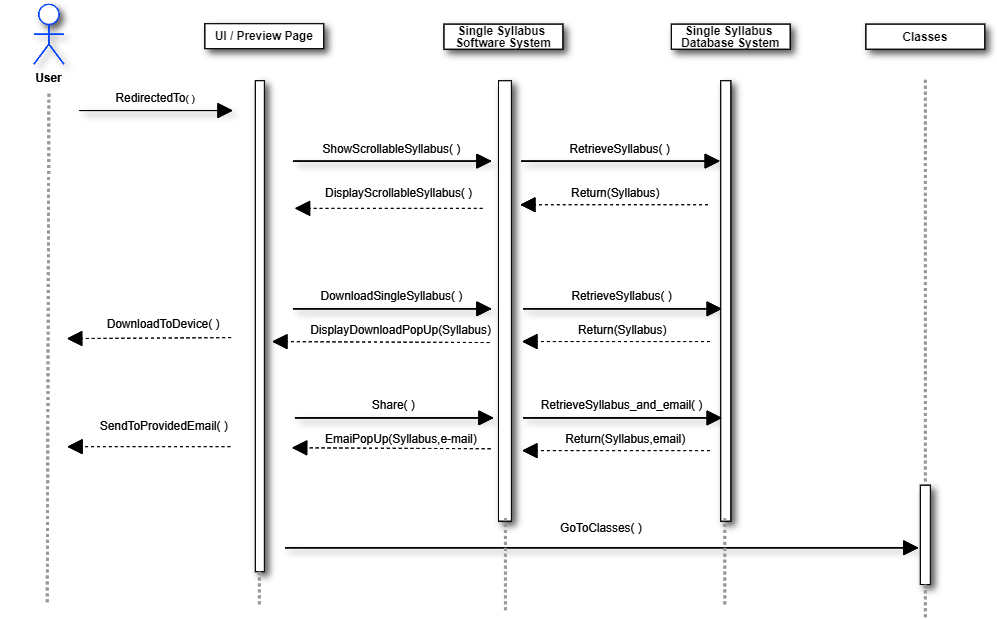
**Sequence Diagrams**

***Upload Syllabi Page (Divya Karivaradasamy)***

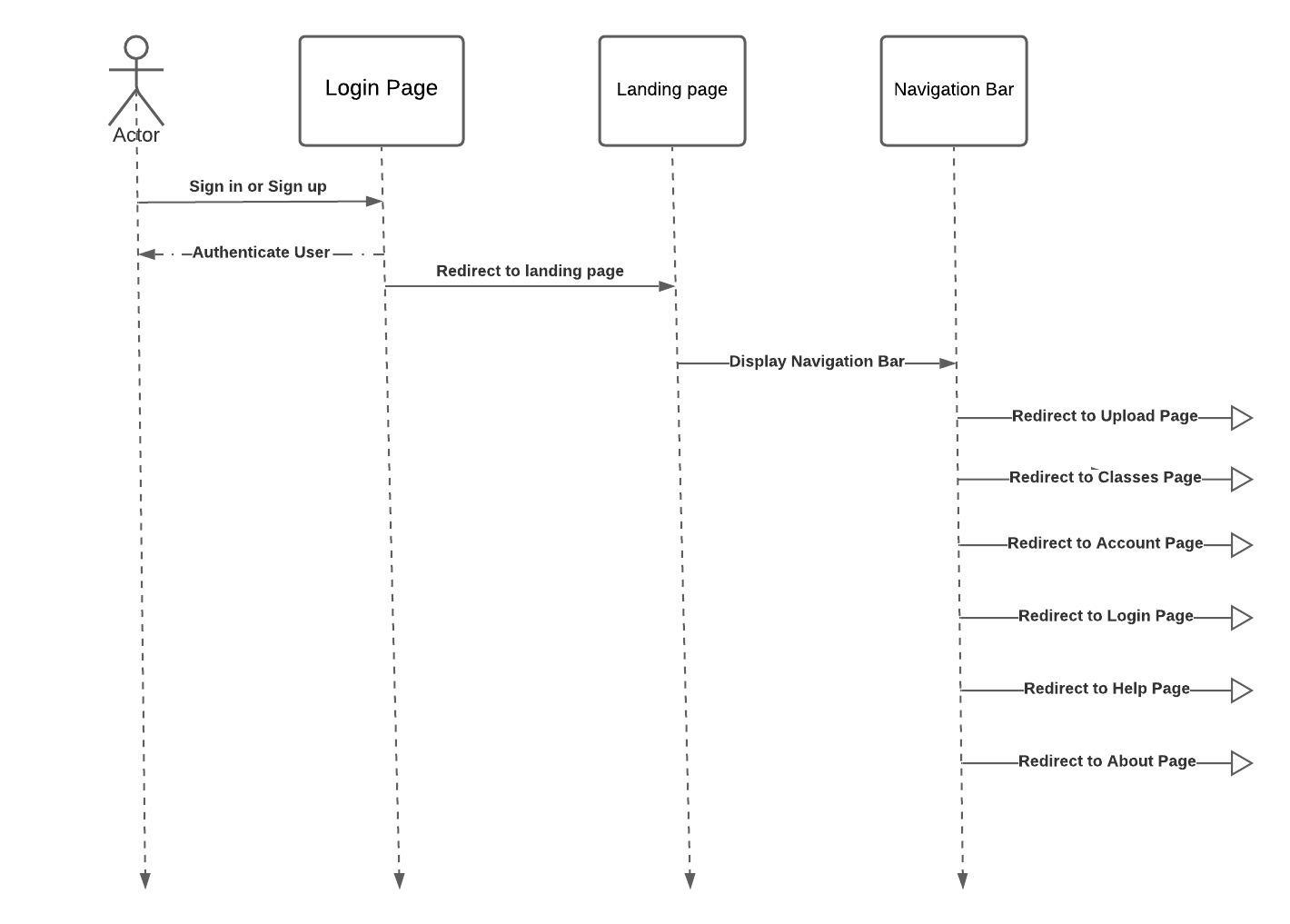
***Card view/edit page (Saaketh)***



***Preview Page (Leonardo Acioli)***



***Navigation Bar (Syed)***



Wire Frames

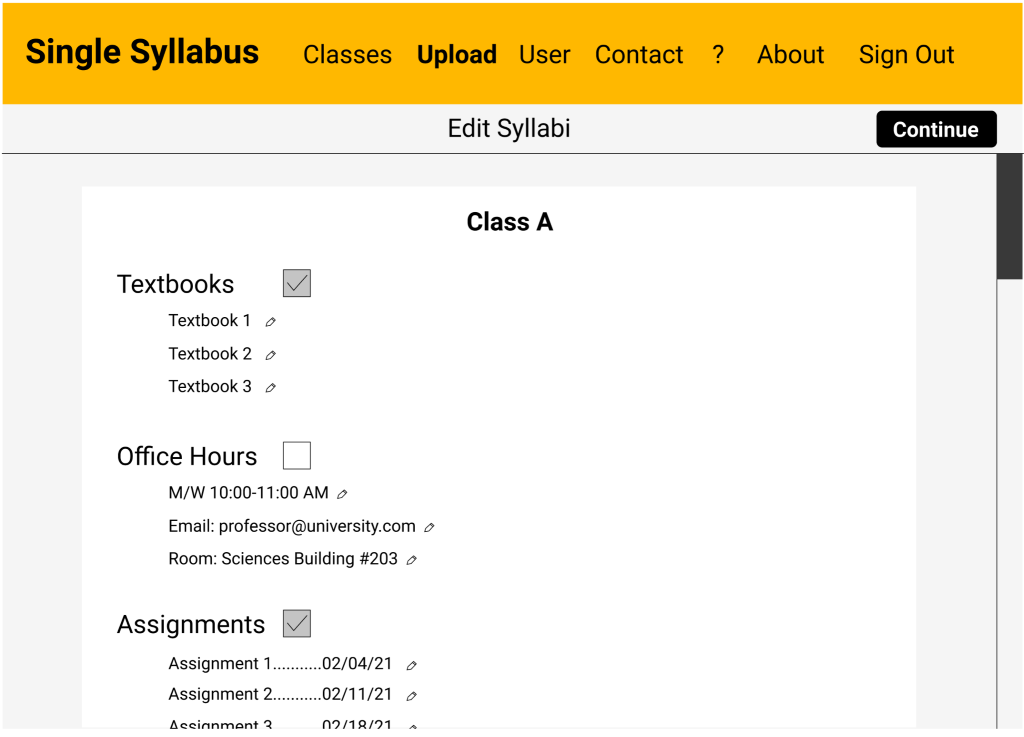
***Log-on Page***



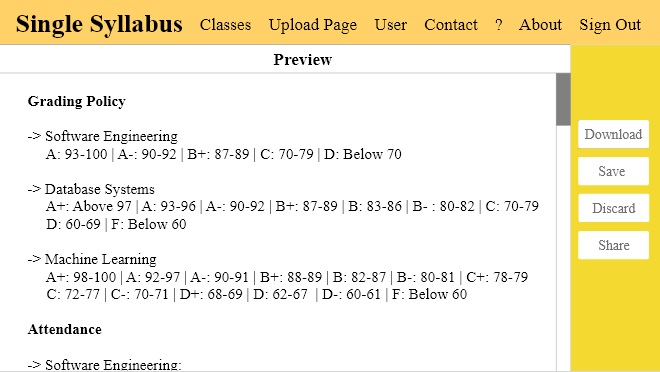
***Upload Page (Leonardo Acioli)***



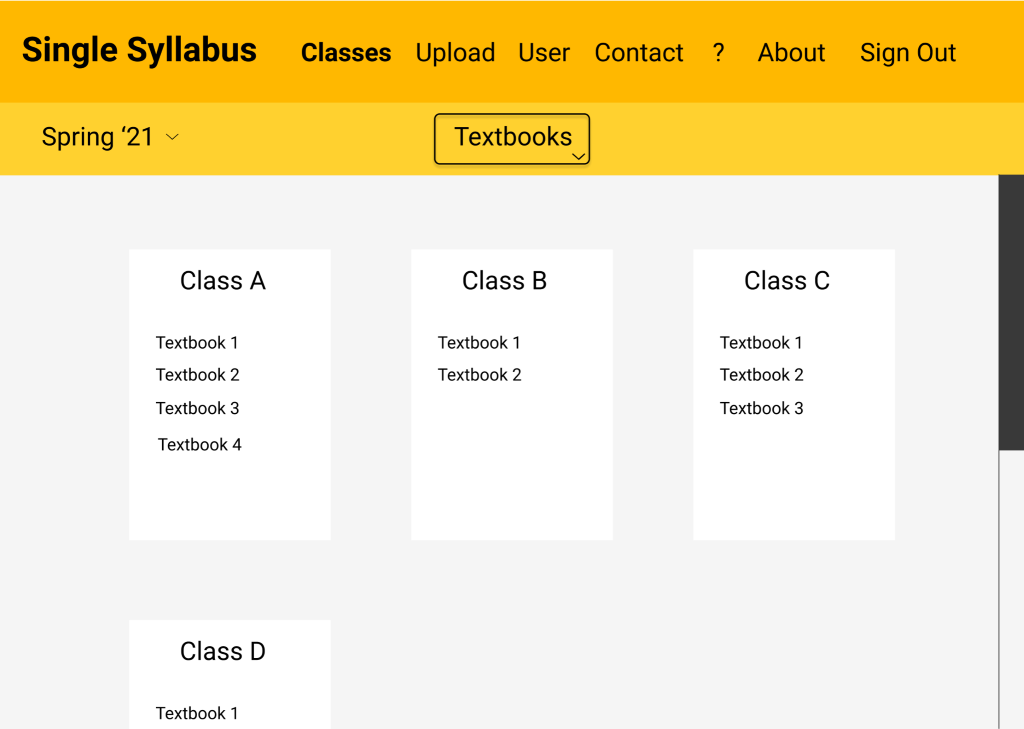
***Syllabus Edit Page (Saaketh Jasti)***



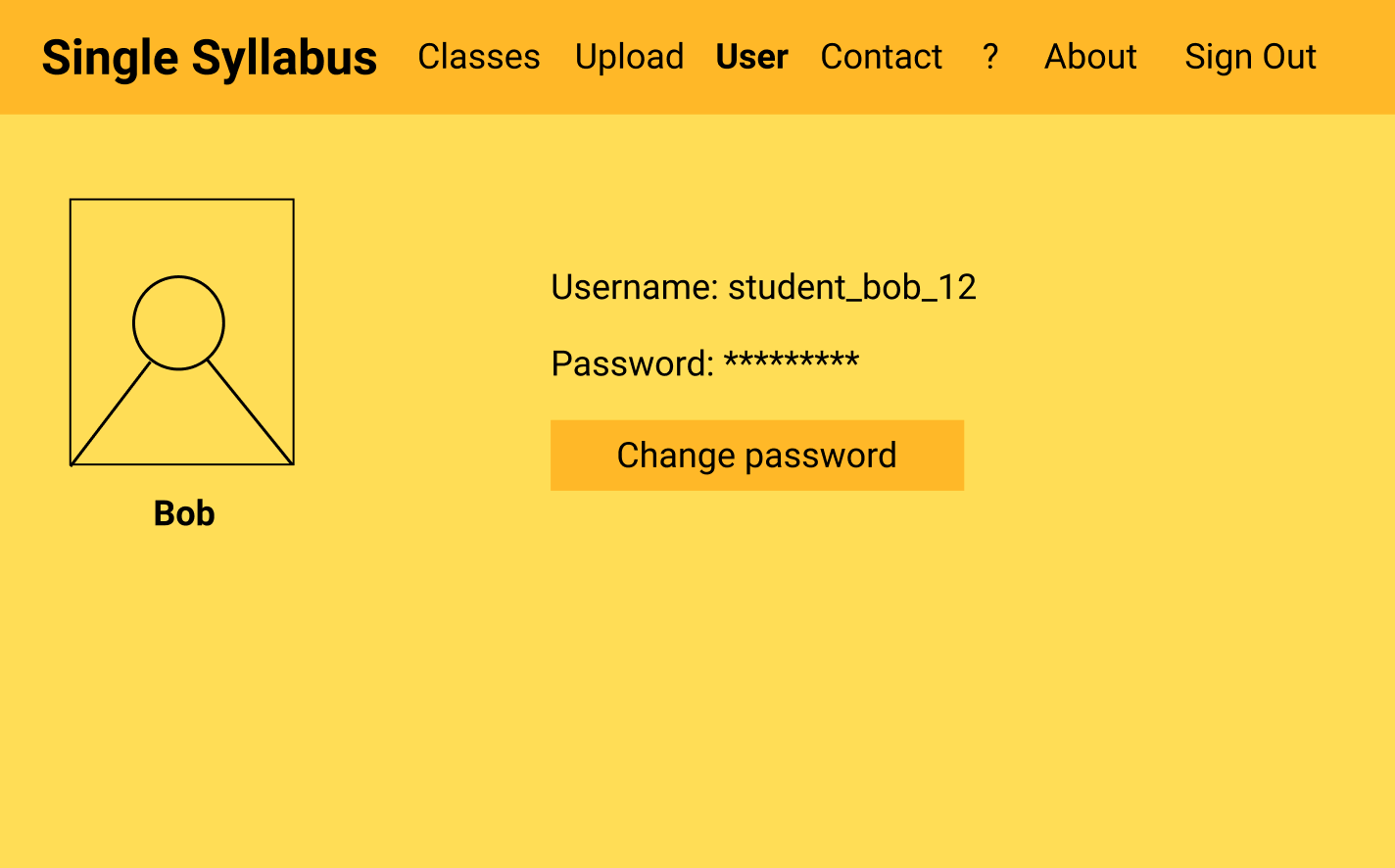
***Preview Page (Leonardo Acioli)***



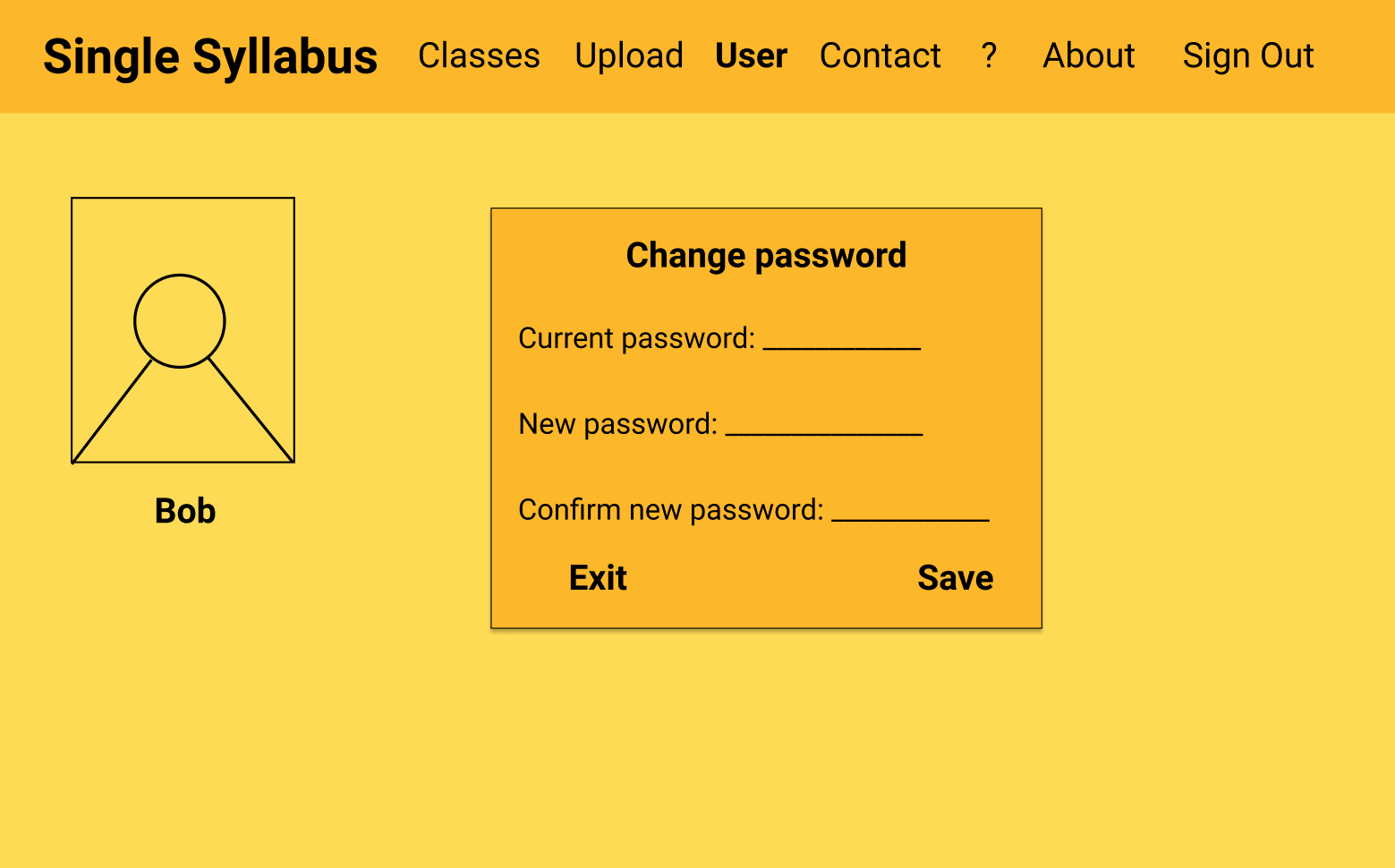
***Classes (Saaketh Jasti)***



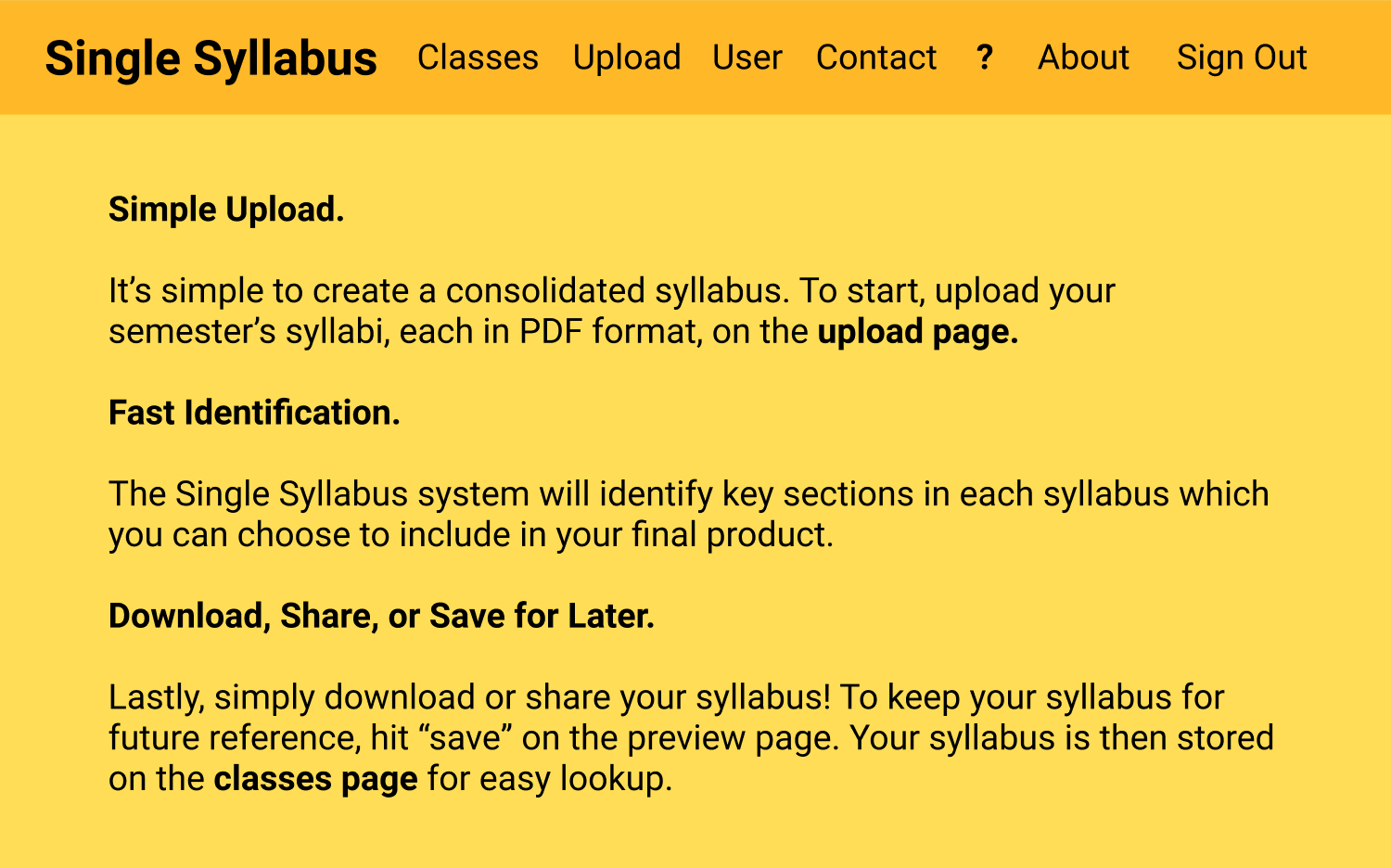
***User Page (Divya Karivaradasamy)***



***User Page – Change Password (Divya Karivaradasamy)***



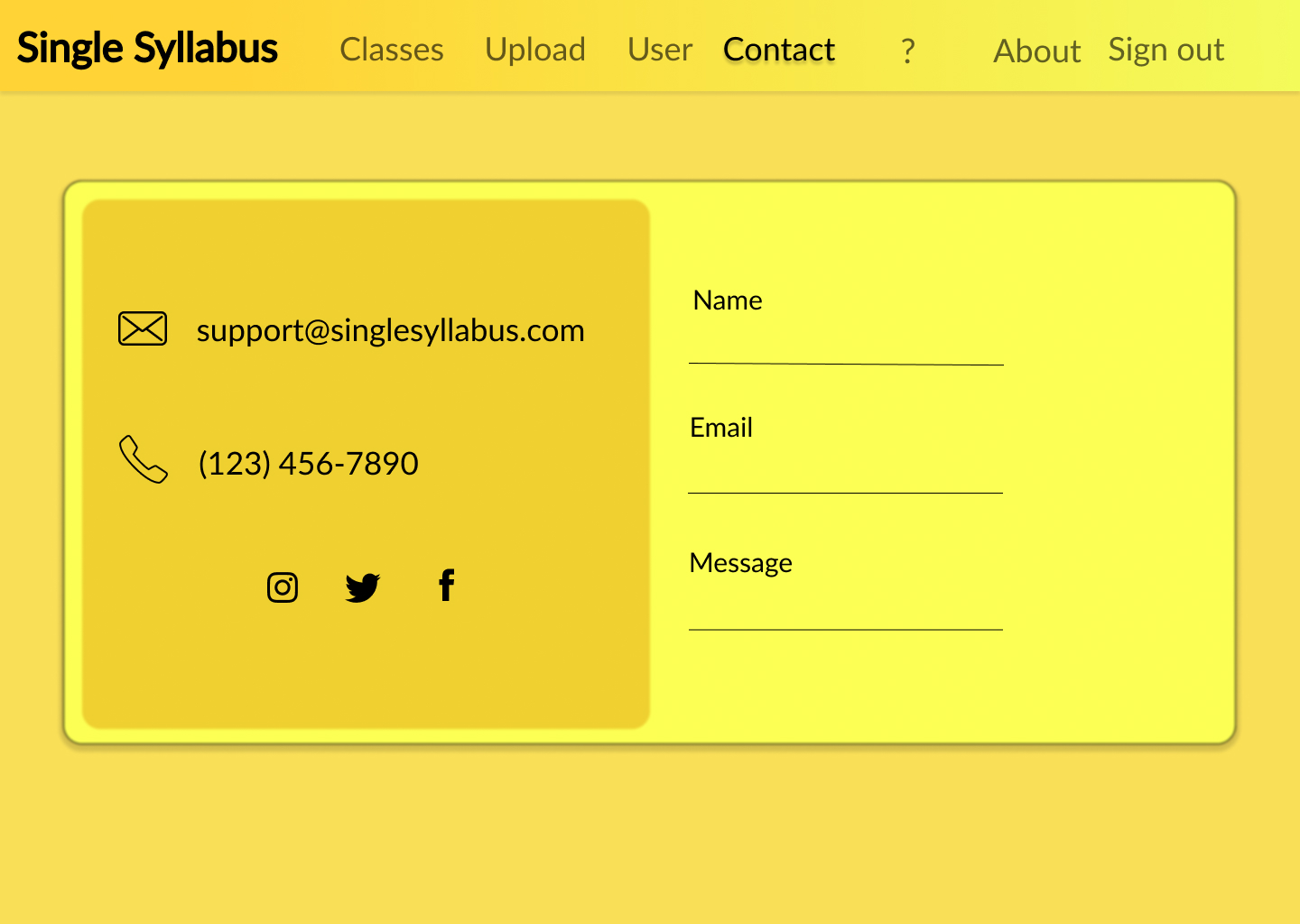
***Help Page (Divya Karivaradasamy)***



***About Page (Syed Quadri)***

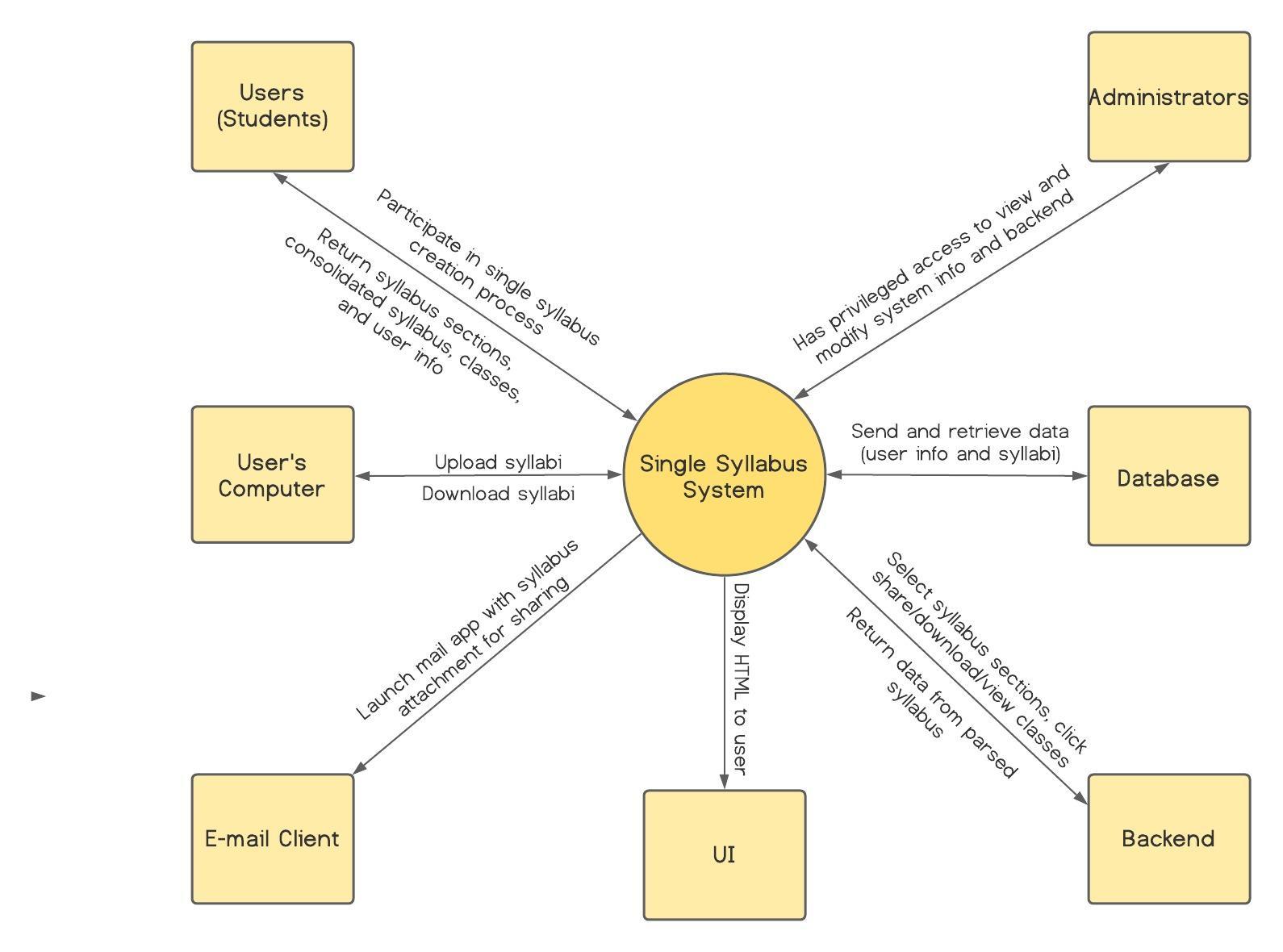


***Contact Page (Syed Quadri)***

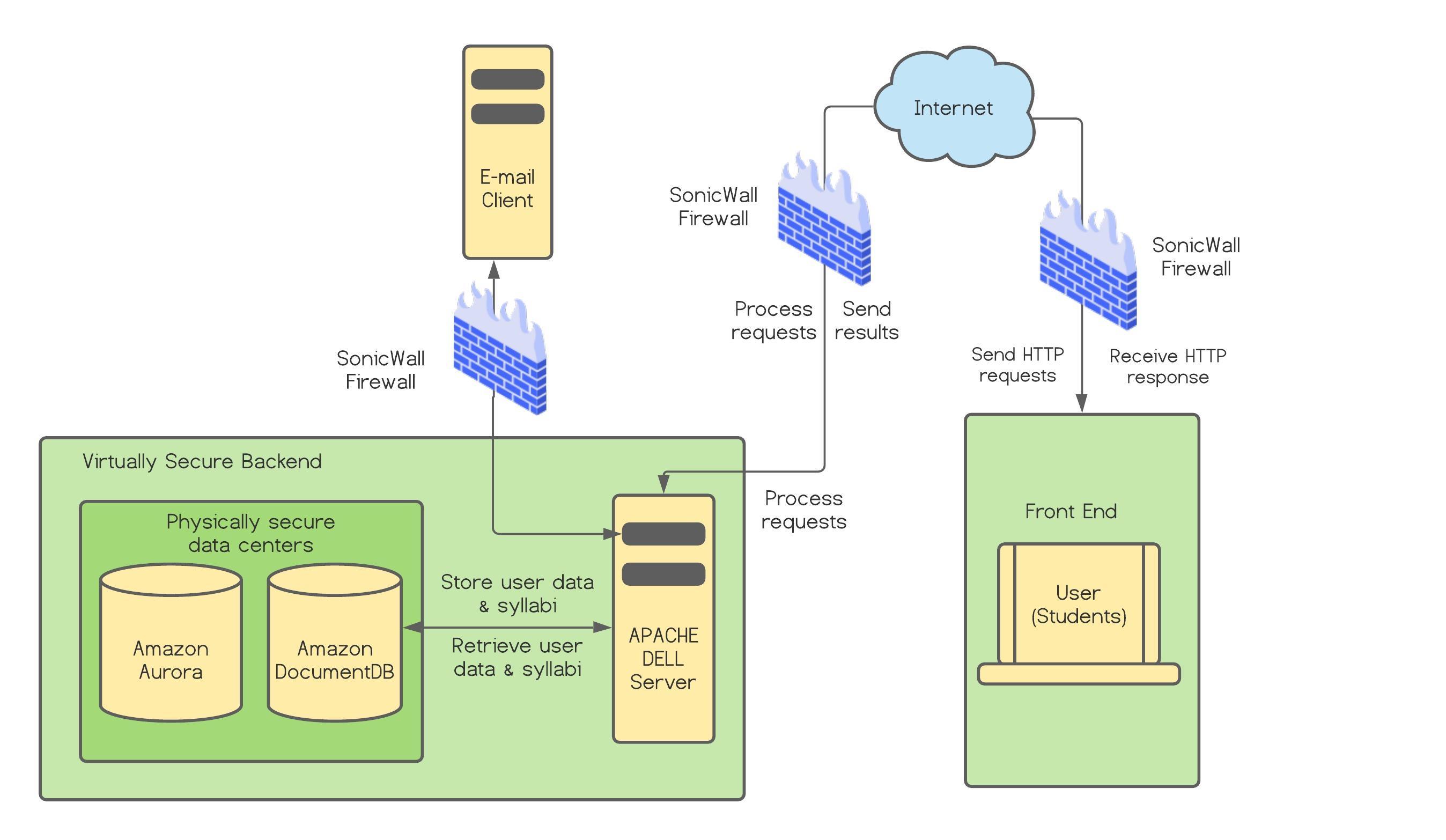


Architecture Design

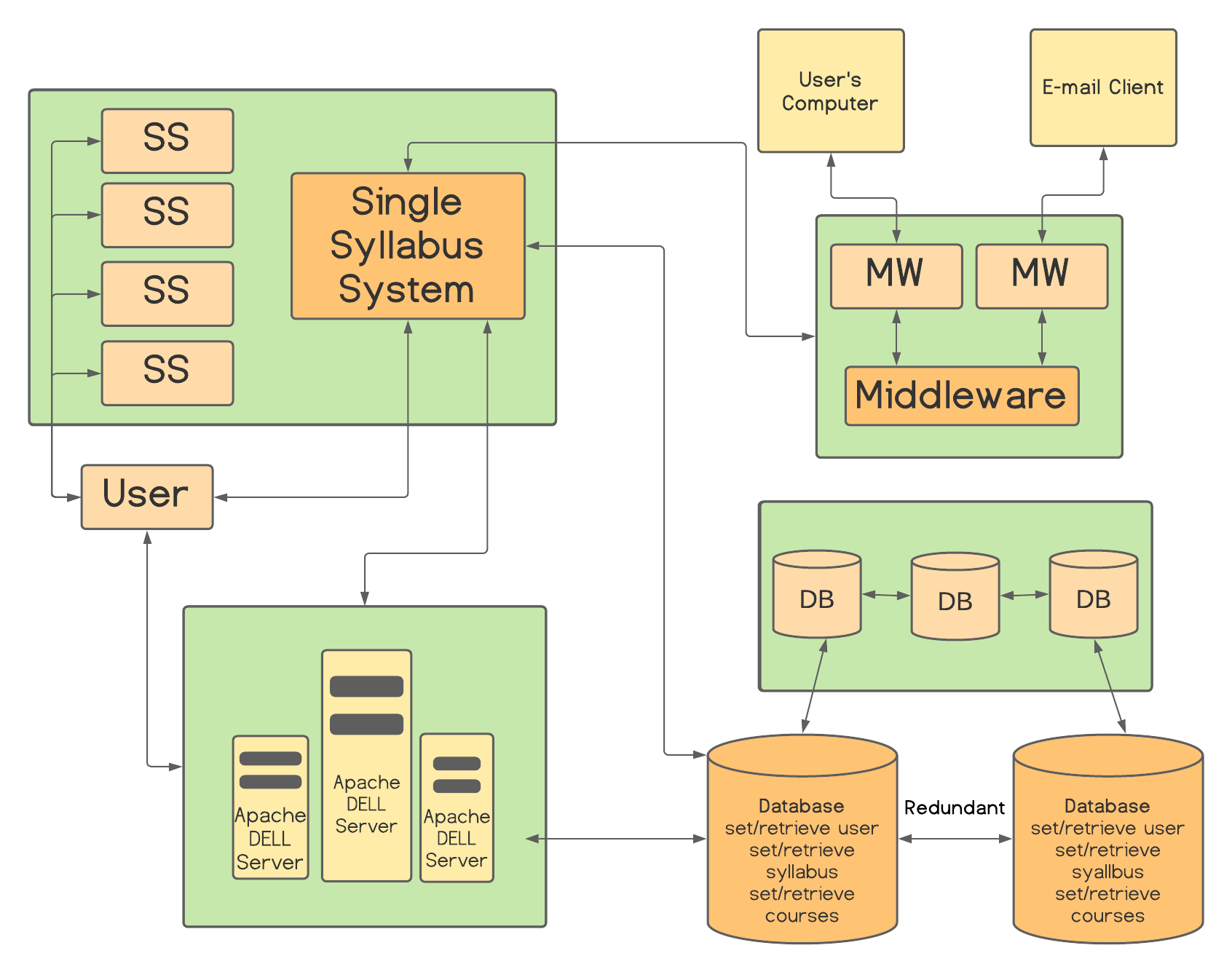
**Context Diagram (Whole Team)**



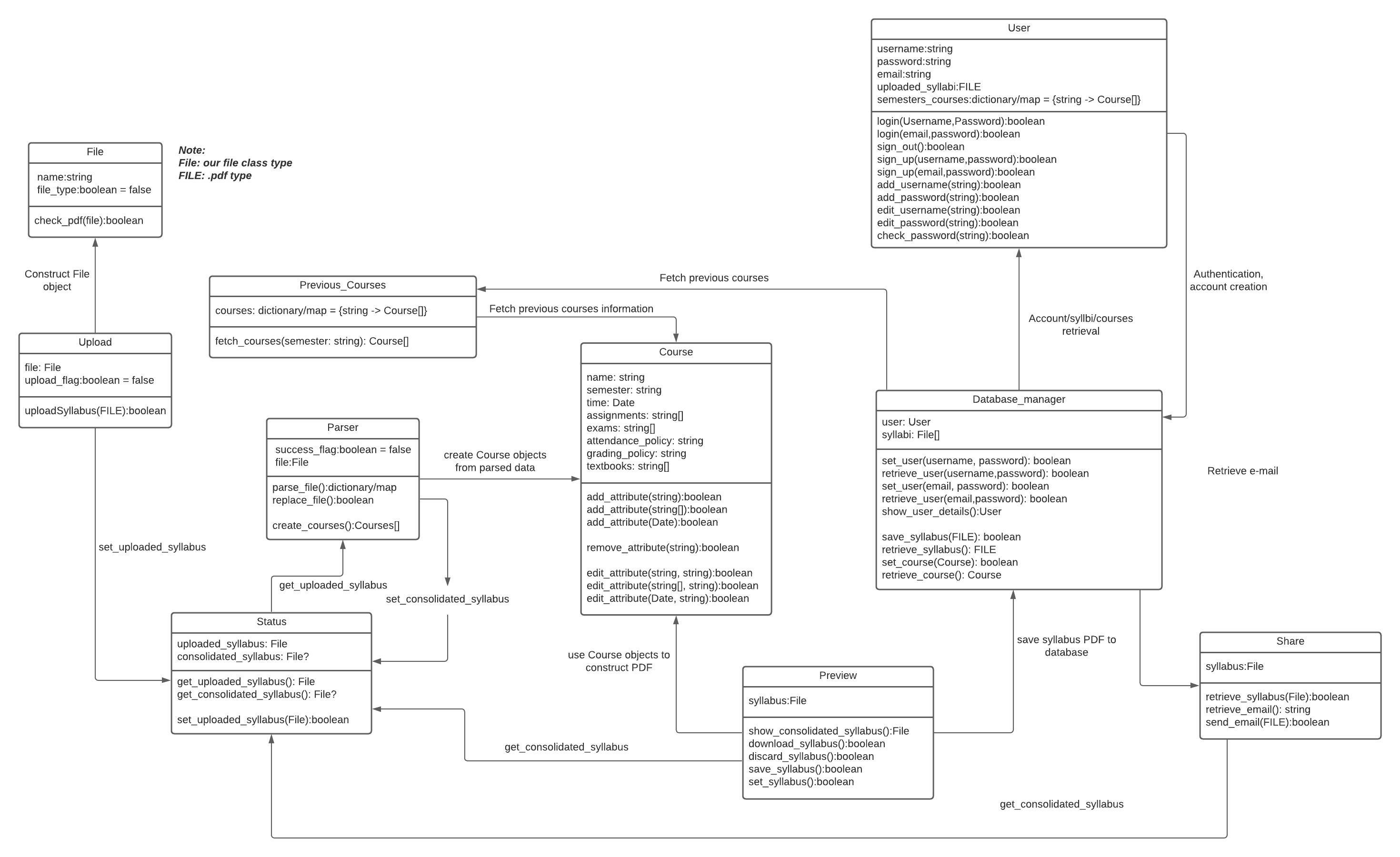
**Physical View (Whole Team)**



**Logical View (Whole Team)**



**UML Classes and Association (Whole Team)**



Implementation

**Research Findings**

* When looking for an appropriate programming language to create the UI and integrate easily with the backend that can efficiently process PDFs, we found a lot of support for the React JS + Python combo and so that is what we went forward with. We found a couple of websites made with these two technologies that seemed to be in a similar vein to ours which also increased our confidence in them.
* When researching PDF parsing tools in particular, we found Python libraries with which developers can examine and label features of a document for later analysis.
* When looking at other websites we discovered that they were all very sleek and simple. Our interface needed to be simple and intuitive for the users.
* When looking on how to integrate HTML and CSS, Bootstrap + React JS and Bootstrap + CSS were a great option for pleasing aesthetics and high functionality. Bootstrap allowed for advanced graphics and design within the application. The Bootstrap website had great resources when it comes to things as to how to implement the components within our code.
* Websites such as W3 schools and Stack overflow were able to provide insightful information and tools that helped the development of the application.

**Implementation Plan**

The Developers built Single Syllabus by dividing up features and components. Each team member built their chosen features by referencing the requirements and wireframes, especially for the frontend portion of development. The team will coordinate their coding efforts by routinely pushing their code changes to GitHub, informing the others of when a new change has occurred, and meeting during unofficial times to discuss problems when they arise.

The team will set up the database together so every team member can experience working with AWS. Once the database is ready, the team will connect the backend to read and store user data from the database. Lastly, the team will create test cases directly based on the functional requirements found in the Software Testing section. If any test case fails, the appropriate component will be updated to resolve the issue and/or better meet the requirements. Throughout this process, the Developers will check whether the requirements are up-to-date with technical needs, and if not will update the requirements.

**Tools Needed**

To build Single Syllabus, we will use the **React JS** framework along with **HTML** and **CSS** for the front end. For our backend, we will use the **Flask** framework and the **Python** language. To store our data and users, we will use **AWS Aurora**. To store documents, we will use **AWS Document DB**. We will host Single Syllabus using **Award Space**. We will use **VSCode** as our editor and **GitHub** as our configuration manager.

**Link for the Code**

GitHub Repository

<https://github.com/divstar1/single-syllabus>

Website URL

https://main.d3mmoqhwxcpncq.amplifyapp.com/

Software Testing

**Research Testing Technologies and Methodologies**

**Requirements Testing**

Requirement Testing: Demonstrating that requirements are feasible and will satisfy user needs (verification and validation).

**Development Testing**

Unit Testing: Testing individual components at the coding level to make sure they are functional work as expected.

Integration Testing: Testing whether groups of individually tested components interact and work well together, without breaking anything.

System Testing: A black box testing method used to test the complete and integrated system to ensure it meets the requirements.

**Release Testing**

Scenario Based Testing: Testing the entire application flow with different typical user scenarios. Ensuring no hiccups and expected end results are achieved.

Performance Testing: Testing done to check how the software will respond under various conditions. The goal is to check responsiveness and real user situations.

**Acceptance Testing**

Alpha Testing: Testing is carried out by the developers themselves to identify all possible issues and bugs. A select group of users work closely with the development team to identify tasks that a real user would perform.

Beta Testing: Where a release of the software is made available to a larger group of users to allow them to experiment and to raise problems that they discover with the system developers.

**Testing Technologies**

Unit Testing: PyTest (Python), Jest (JS, automated)

Integration/System Testing: Robot (Python), Cypress (JS)

Requirement, Scenario, and Performance testing can be done with the previously mentioned frameworks.

Alpha, Beta, and Acceptance testing can be done by trial users.

**Testing Plan**

We will start by testing our code through *unit testing, component/integration testing, and system testing.* Unit testing will involve testing individual components (such as the log-in button or checkboxes) once they are created. We will then test how components work together to ensure every feature (ex: log-in) provides a smooth experience. After component testing, the entire system will be tested against our functional and non-functional requirements.

We will then ensure the application can be released through *scenario and performance testing*. We will test numerous potential user scenarios (ones that should work and those that would break the application) and check for expected results. Finally, performance testing will ensure that all our non-functional requirements are met by testing things like speed of retrievals, etc.

Lastly, we will execute acceptance testing through *alpha testing and beta testing.* During alpha testing, users will guide developers to test common scenarios. In beta testing, the development team will employ a small group of users to act as students using Single Syllabus. The developers will then identify and fix the issues the users find. Finally, the application will be deemed worthy of release with a final test by the stakeholders and consumers.

**Create Functional and Non-Functional Test Cases**

*Functional*

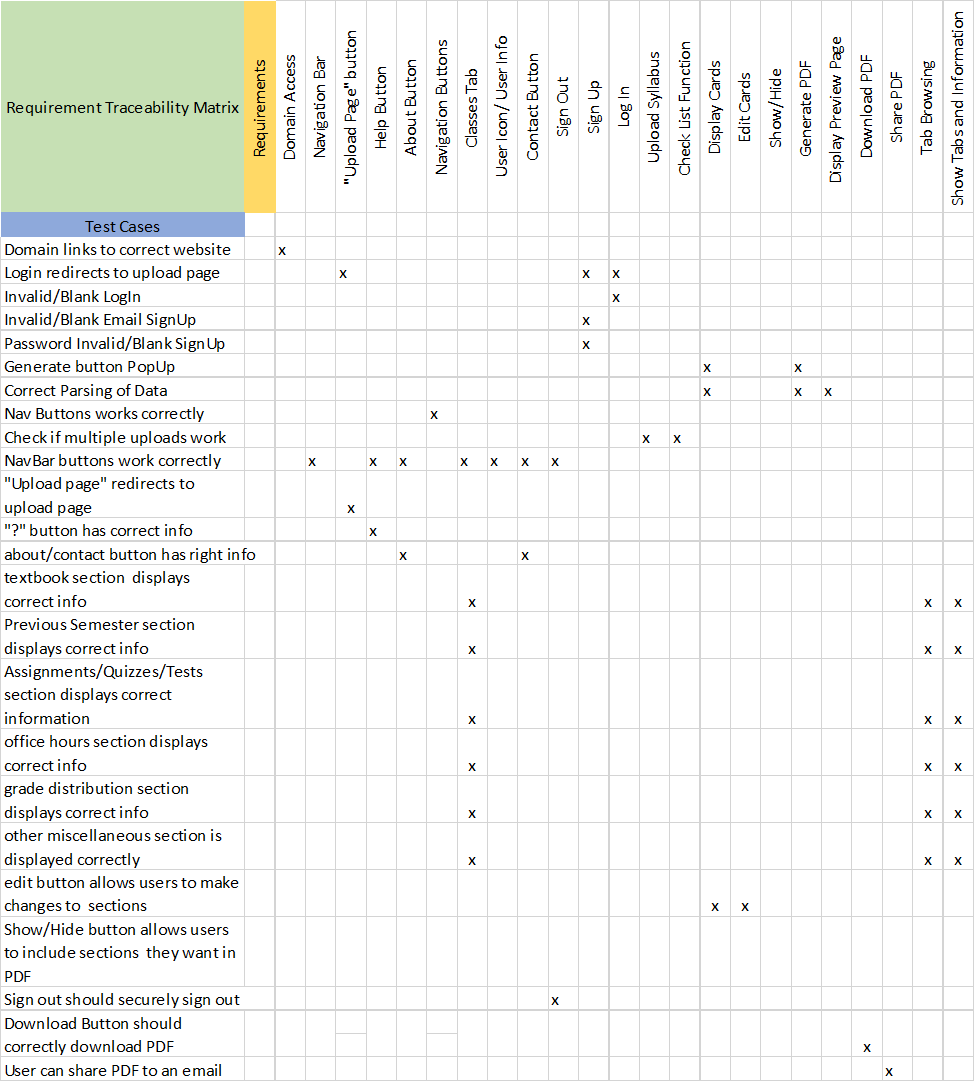
* Domain links to correct website (Divya)
* Check whether page redirects to landing page after valid login (Leonardo)
* Check whether error message shows after invalid/blank login is entered (Divya)
* Check whether error message shows when invalid/blank email is entered during signup (Divya)
* Check whether error message shows when password field is blank/invalid during signup (Leonardo)
* Check whether ’upload’ button opens pop up to upload PDF files (Saaket)
* Check whether the system can parse the data from the PDF files correctly (Saaketh)
* Check whether the ‘navigation’ button leads to the correct webpage (Syed)
* Check whether the system allows multiple uploads (Saaketh)
* Check whether the system navigation bar buttons link to the correct page (Saaketh)
* Check whether the ‘?’ button has the correct information. (Saaketh)
* Check whether the 'about’ and ‘contact’ button have the correct updated information (Syed)
* Check whether the textbook section of classes page displays correct information (Syed)
* Check whether the previous semester section of classes page displays correct information (Syed)
* Check whether the Assignments/Quizzes/Tests section of classes page displays correct information (Leonardo)
* Check whether the office hours section of classes page displays correct information (Syed)
* Check whether the grade distribution section of classes page displays correct (Leonardo) information (Leonardo)
* Check whether any other miscellaneous section is displayed correctly on classes page (Syed)
* Check whether the ‘edit’ button allows users to make changes to displayed sections (Divya)
* Sign out should securely sign out (Leonardo)
* Download Button should correctly download PDF (Leonardo)
* Upload page button will lead to upload page (Leonardo)

*Non-Functional*

* Check whether end-to-end flow executes as expected on Google Chrome 87, Safari 10.11, Microsoft Edge 10, Firefox 83, Opera 60 for desktop. (Leonardo)
* Check whether HTML/CSS look consistent across browsers. (Divya)
* Check whether landing page displays the logo. (Divya)
* Check whether each page includes a title at the top right describing its purpose. (Divya)
* Check whether each page includes a button in the lower right corner navigating user to next step. (Saaketh)
* Check whether all buttons are enabled. (Syed)
* Check whether all buttons are round with a consistent corner radius. (Leonardo)
* Check whether entire app consists of only yellows, oranges, whites, greys, and blacks. (Leonardo)
* Check whether design elements have consistent HEX values. (ex: all titles are black) (Leonardo)
* Check whether all text share a font. (Leonardo)
* Check whether all text elements have consistent font sizes (ex: all button text is size 14.0) (Divya)
* To check for ease-of-use, user testing, at least 9/10 users should complete the flow without errors and have no questions. (Syed)
* Check whether our website is encrypted. (Divya)
* Confirm whether no trackers are enabled on the website. (Divya)
* To check service availability, test whether for every 100 logins, 99 succeed. (Divya)
* Check whether website undergoes maintenance every two weeks, from 4:00 AM to 6:00 AM on Sundays. (Leonardo)
* Check whether software accepts PDF file uploads. (Divya)
* Check whether software rejects non-PDF file uploads. (Saaketh)
* Check whether PDF parsing executes within 1 minute for uploads of <= 10 PDFs of <= 5 pages in length each. Must succeed for 99/100 attempts. (Leonardo)

**Requirements Traceability Matrix**

**Functional Requirements**



**Non-functional Requirements**



|  |  |  |  |
| --- | --- | --- | --- |
| Number | Test Cases (Pass or Fail) | Explanation | KEY |
| 1 | Domain links to correct website |  | GREEN = PASS |
| 2 | Check whether page redirects to landing page after valid login. |  | ORANGE = FAIL |
| 3 | Check whether error message shows after invalid login is entered. |  |  |
| 4 | Check whether error message shows either login field is blank. |  |  |
| 5 | Check whether ’upload’ button opens pop up to upload PDF files. |  |  |
| 6 | Check whether the system can parse the data from the PDF files correctly. |  |  |
| 7 | Check whether the ‘navigation’ button leads to the correct webpage. |  |  |
| 8 | Check whether the system allows multiple uploads. |  |  |
| 9 | Check whether the system navigation bar buttons link to the correct page |  |  |
| 10 | Check whether the ‘?’ button has the correct information. |  |  |
| 11 | Check whether the 'about’ and ‘contact’ button have the correct updated information. |  |  |
| 12 | Check whether the textbook section of classes page displays correct information. | Class page currently has hardcoded example data. |  |
| 13 | Check whether the previous semester section of classes page displays correct information. | Class page currently has hardcoded example data. |  |
| 14 | Check whether the Assignments/Quizzes/Tests section of classes page displays correct information. | Class page currently has hardcoded example data. |  |
| 15 | Check whether the office hours section of classes page displays correct information. | Class page currently has hardcoded example data. |  |
| 16 | Check whether the grade distribution section of classes page displays correct information. | Class page currently has hardcoded example data. |  |
| 17 | Check whether any other miscellaneous section is displayed correctly on classes page. | Class page currently has hardcoded example data. |  |
| 18 | Check whether the ‘edit’ button allows users to make changes to displayed sections. |  |  |
| 19 | Sign out should securely sign out. |  |  |
| 20 | Download Button should correctly download PDF. | Download functionality is not implemented. |  |
| 21 | Upload page button will lead to upload page. |  |  |

Deployment

**Deployment Research**

To deploy our software, we considered the **type of document** to parse, the **number of documents** that can be uploaded at once, and the **programming language** to build the software with. We needed to parse up to 10 PDF documents. We decided to use **React.js, HTML/CSS, for our front-end and Flask for our back-end** to be able to accept PDF input and parse it.

After the initial static website was created, each page was converted into a React component and worked on separately and built as per the requirements. To release our project, we used **AWS Amplify.** We researched AWS and found out that it allowed **quick, free** tools to deploy web apps. It also provided **authentication and database** services to manage users.

To further implement the software, we plan to use **Amazon Aurora, Amazon Document DB, and Apache Dell server.** The system will have multiple instances of the Apache Dell server and each of the databases. This is for **prevention of data loss** reasons as well as increase the **service speed.** The system also will pass through several layers of **firewall** for security reasons.

**Deployment Plan**

We deployed our web application through **AWS Amplify**, a platform for deploying and testing web and mobile applications.

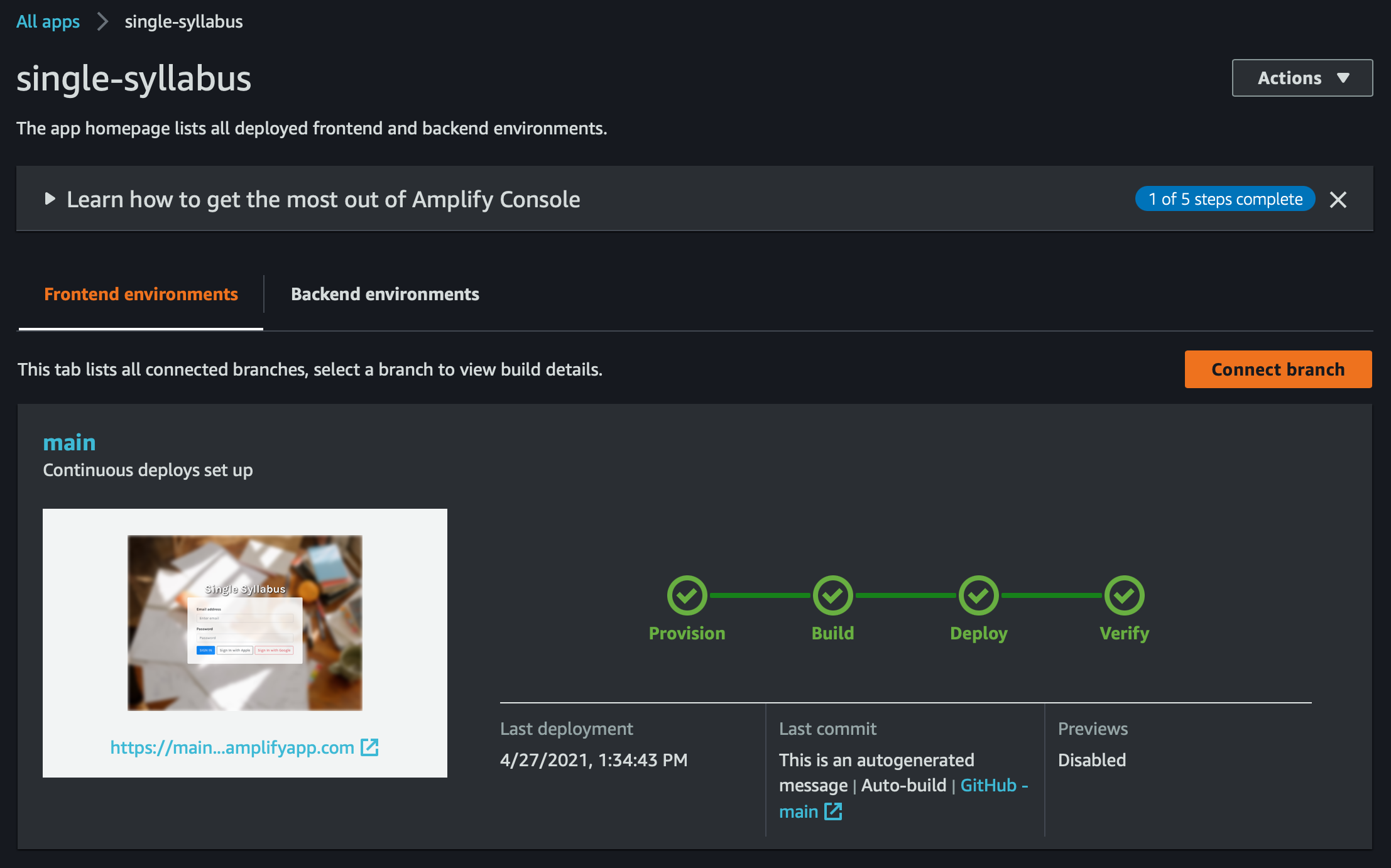
We started by creating a free AWS account. Then, we created an Amplify instance and connected the **GitHub repository** to it. Amplify then generated a link to our website. The Amplify instance also tracks changes in our project repository and updates the deployed instance **(CI/CD).** Through Amplify, we can now access our website and manage deployment. Furthermore, Amplify provides tools to run tests on our project.

We will also be using an **AWS Web Application Firewall (WAF)** to protect the system from threats. The WAF takes policies and rules we create about access points and users and uses that and other commonly used protocols to block against exploits and vulnerabilities.

For the future, we would like to create and use a custom domain name for Single Syllabus, so users can access our tool easily. Amplify provides functionality for this.

For authentication, we will be using **Amazon Cognito**, a managed user identity service, through the **Amplify CLI.** This service will be imported into our React application so that the login page implements the functionality of this service to authenticate users.

*AWS Amplify Console for Single Syllabus*



Glossary of Terms

Aggregate: To extract information from all uploaded syllabi for the purpose of producing a consolidated syllabus.

Consolidated/single syllabus:A syllabus that contains the key information of multiple syllabi. For example, the “Assignments” section of a consolidated syllabus lists all the assignments contained in the source syllabi. The “Office Hours” section contains all the professors and office hours of the sources.

Extracted Data: The text parsed from multiple syllabi identified as belonging to a section.

Section: A set of information found on a syllabus relating to a particular aspect of a course. Sections can, for example, describe course assignments, quizzes, office hours, or textbooks.

Semester: A time period during which a student has a fixed set of course syllabi and creates a consolidated syllabus.

Syllabus/syllabi: A structured, text document containing course information for a particular class. A syllabus consists of multiple sections.