CS 40700: Design Document

Team Name: CollegeApp.IO
Team Number: 12
Team Roster

- Gokul Yenugadhati - gyenugad@purdue.edu
- Harsha Lingareddy - hlingare@purdue.edu
- Sanat Mouli - smouli@purdue.edu
- Vishaal Bommena - vbommena@purdue.edu

Table of Contents

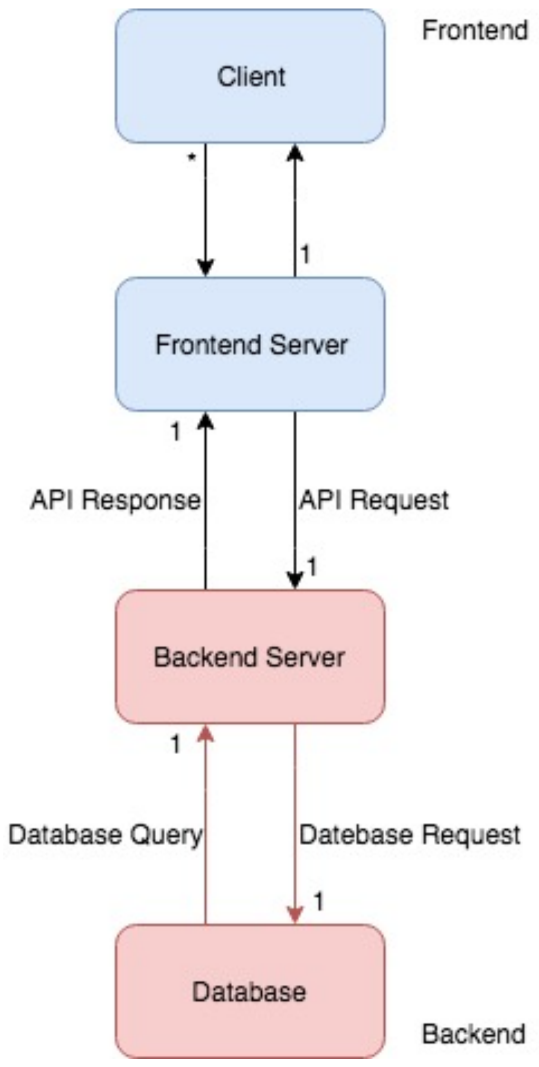
1. Purpose

Currently students apply to colleges using a portal called The CommonApplication. The issue with this application is that there is no source of data for students to understand what it takes to get into a college. Our application aims to solve this issue by providing the user data visualizations regarding the college they are applying for and how applicants in the past have done with a similar application. Our application also helps the college admission team decide on the applicants they want to hire by providing them a console with data visualizations that helps them choose the applicants they want to accept or reject.

2. Design Outline

2.1 High Level Overview/UML Diagram

Our project will use the client-server model. The front-end server will host the client and will request data from the backend server. The backend server will respond to the client requests, by sending queries to a fully functional database and will provide a response to the front-end server, which will then render the data that it received on the client. The diagram below is a UML diagram representing how the interactive between the 4 components works.



2.2 Individual System Components

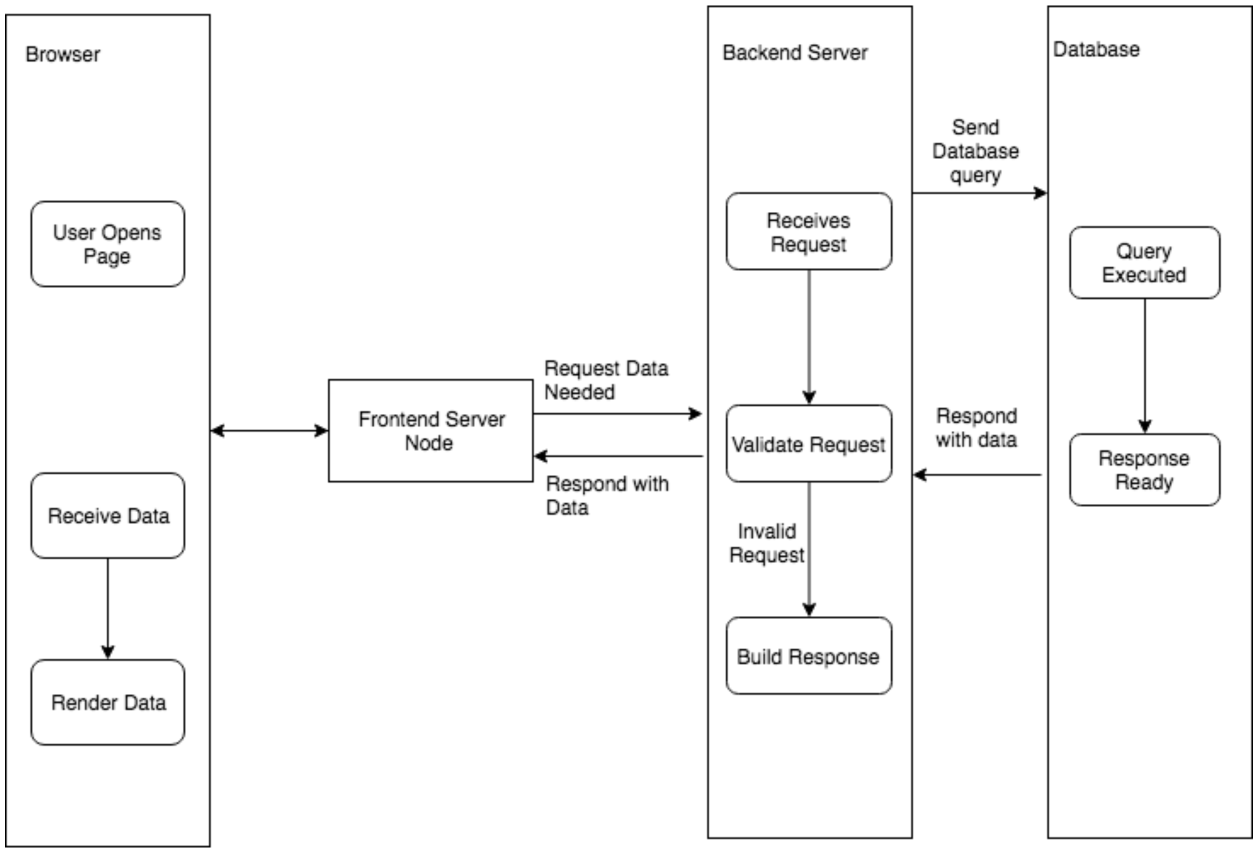
1) Web Client
   a) Client sends AJAX requests to the server API
   b) Client receives an AJAX response from the server
   c) Response data is rendered to the user using Angular

2) Frontend Server
   a) Hosts the Angular application using Node.js
   b) Manage node dependencies and libraries for Angular

3) Backend Server
   a) Processes AJAX requests from the web client
   b) Retrieves data from the database using Python/Flask
   c) Returns an AJAX response to the web client

4) Database
   a) The database will store all of the user data regarding their application and where they applied
   b) The database should be able to hold all the user data and college even if the API server or any other module crashes or does not function properly. The workings of the API server and other modules should have no effect on the user data which is to be stored
   c) The database will primarily store relationships between the student application table and the current_application table. The primary purpose of the database is to hold records of the student's applications and any historical data.

2.3 Flow of Events

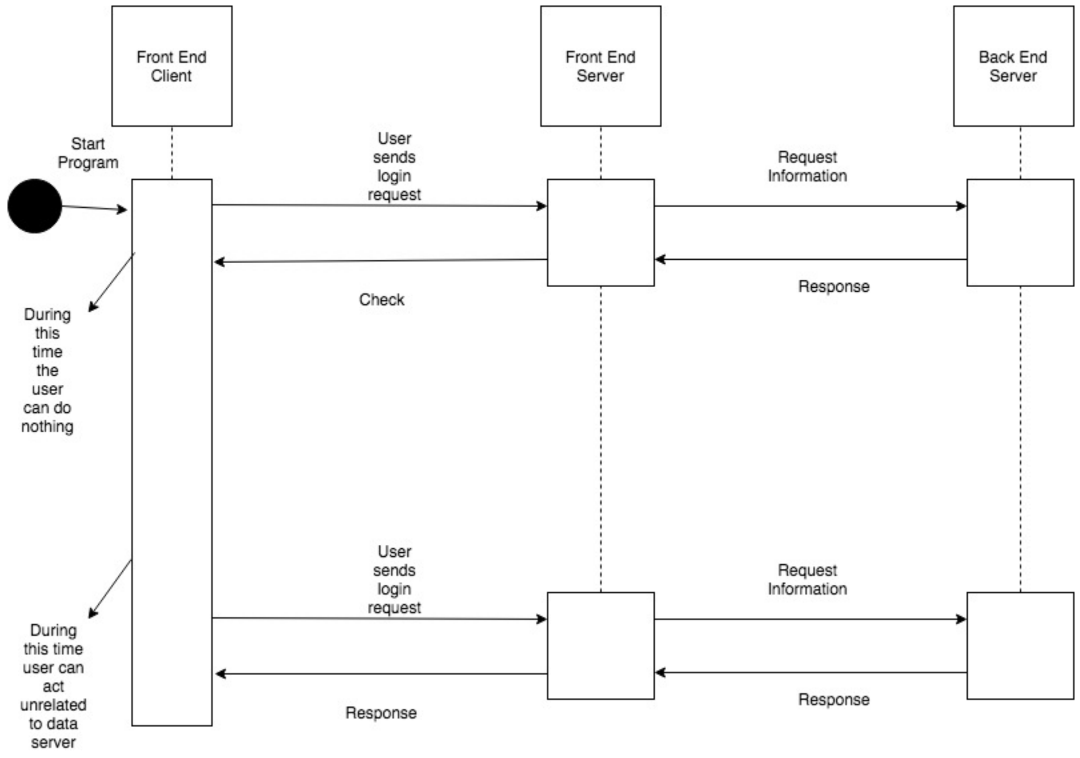


The diagram above shows the flow of events. The flow starts off with the user opening the application through their browser.

The frontend server manages the dependencies required to run the frontend and hosts the frontend client. The reason for having a front-end server is for it to be independent from the backend server just in case the in the small chance the backend server errors out. It will also help the frontend team to develop at their own pace and not be dependent on the backend server.

The backend server will first receive and validate the request from the frontend server. If the request is valid it will query the database, where the query will be executed and generate a response to send back to the server. If the request is invalid no query will be made and an error response will be sent. The data will then be sent back to the frontend server where it will render the data onto the page for the user to view it.

2.4 UML Sequence Diagram



When the program starts, the first step is to login and the server would compare the login information with the corresponding information from the database. After which it can send multiple requests to the server for information like the user application and other functionalities.

3. Design Issues

3.1 Functional Issues

Issue 1: How should users navigate through the available colleges?

a) **Option 1: Display all available colleges in on a central page.**

b) Option 2: Drop down form on a central page.

Decision: We opted to display all the available colleges on a central page that allows the user to navigate to the specific college and not have the user go through a huge drop down of colleges available.

Issue 2: When to ask for the user to fill out the profile?

a) **Option 1: On signup**

b) Option 2: Through a profile page

Decision: We opted to have the user fill in his information on signup rather than have the user fill in the information post registration since this would allow the user to apply directly rather than have the user register and then apply for colleges.

Issue 3: How many essay questions can a college have?

a) **Option 1: Up to three college questions**

b) Option 2: Have a single college question

Decision: We opted for colleges to be able to have up to three college questions instead of a single question since this allows the college to be able to test the student better.

Issue 4: How do college admission staff accept or reject students?

a) **Option 1: One person from the college admission staff can accept from a list of students.**

b) Option 2: Create a vote room for different college admissions staff to choose a student.

Decision: We chose to have a person from the college admission staff since this would allow for a CAS member to be able to make a final call, thereby saving them time taken to make decisions.

Issue 5: What data visualizations do we provide?

a) **Option 1: User can choose the rows and columns for the plots from a list of variables.**

b) Option 2: User can only view certain plots provided to them.

Decision: We chose to allow the users to choose the variables they want to compare against on scatter plots since this would help them make a more comprehensive decision based on factors that they deem important rather than preset variable plots.

3.2 Non-Functional Issues

Issue 1: What front-end framework to use.

a) **Option 1: Angular.js**

b) Option 2: React.js

Decision: We chose to use Angular.js instead of React.js since we can build intuitive user interfaces easily and quickly. Angular manipulates the DOM by extending HTML with directives. As a result of the two-way data binding, any changes to the view can be reflected in the data and vice versa.

Issue 2: Whether to use a specialized server to host the front end.

a) **Option 1: Node.js**

b) Option 2: Flask

Decision: We decided to use an individual server for the frontend, so we can parallelize development between the frontend and the backend teams. Node also gives us access to a huge array of packages and libraries to use to speed up front-end development.

Issue 3: What type of backend framework is most appropriate for our data?

a) Option 1: Node.js

b) **Option 2: Flask**

c) Option 3: Spring

Decision: We chose to use Flask for the backend since it is extremely flexible and makes routing URLs really easy. We opted against spring since we are more familiar with Flask.

Issue 4: What database software should we sue for our data?

a) Option 1: MongoDB

b) Option 2: Firebase

c) **Option 3: PostgreSQL**

Decision: For efficient functionality we decided that we would require an SQL database over a NoSQL database, since there are many data points that need to have a key value implementation. Also, through the PostgreSQL we can have a wide variety of data types, not limiting us to a certain data type for a table.

Issue 5: Which HTTP connector should we use to connect the front-end to the back-end?

a) **Option 1: AJAX**

b) Option 2: WebSocket

c) Option 3: SSE

Decision: We chose to use AJAX to connect the frontend to the backend since it allows for asynchronous exchanging of small amounts of data with the server without affecting the remaining of the page.

Issue 6: How was security of student data implemented in historical DB?

a) **Option 1: When inserting into the historical DB all personal information is removed**

b) Option 2: Include all data regarding a student in the historical DB

Decision: We chose to use option 1 because we don't want to store any sensitive data regarding old applicants other than the key information to perform the data visualizations.

Issue 7: How do you check that students don't want their data to be stored for the future?

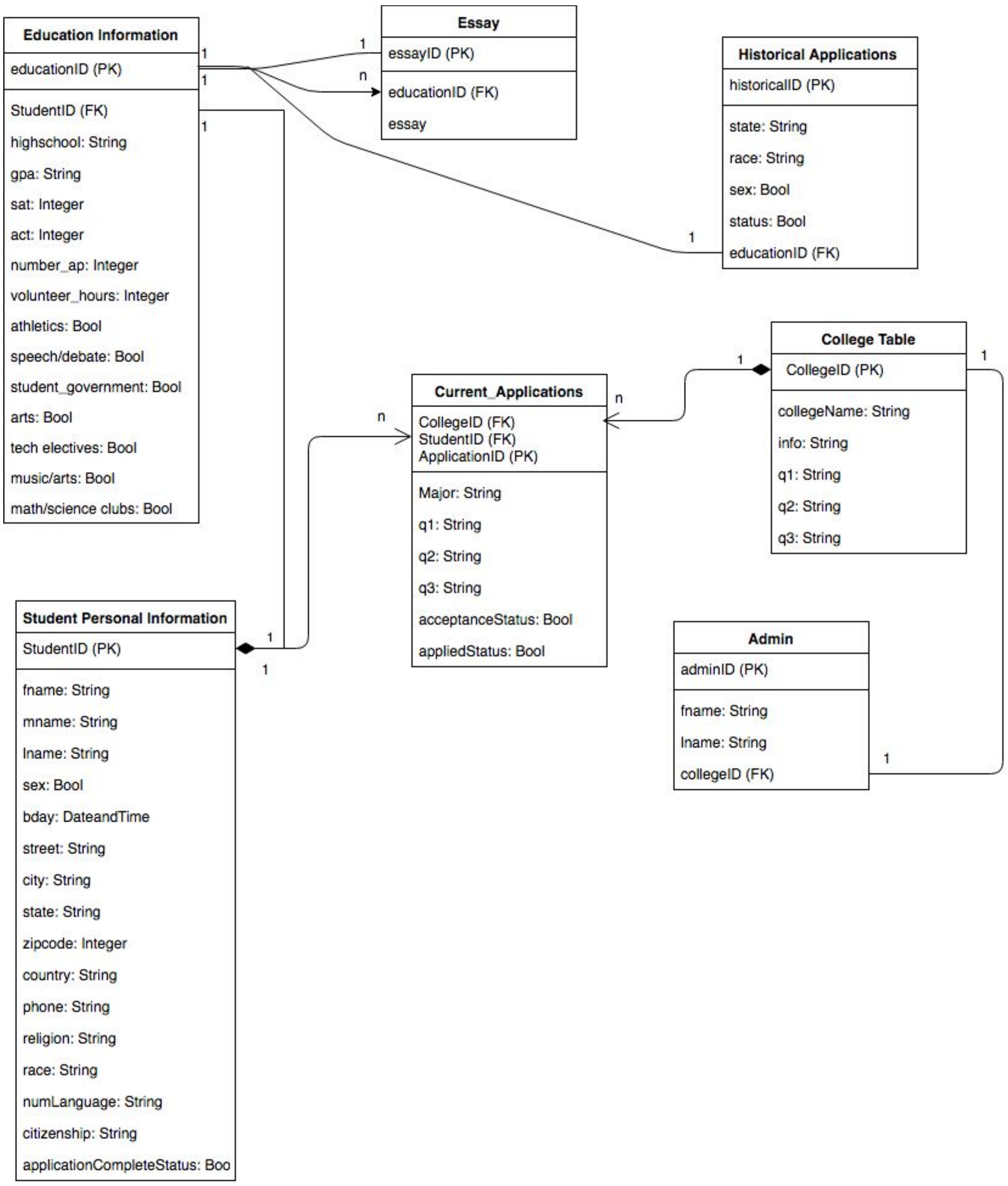
a) **Option 1: Have a checkbox to know their decision in if they want data be stored**

b) Option 2: If student doesn't want data to be stored stripe all personal information before storing

Decision: We chose to use option 1 so the student has more control of the data and they can decide the state. We can combine that with issue 6 to make sure all the data is kept private and, in a manner, which follows regulations.

4. Design Details

4.1 Data Class Level Design



4.2 Description of Data Class Level Design:

The picture above is a mockup of our database schema. As of now, this is our design, but it may change in the future as some columns may be moved around and tables may be added to support more features.

1) Student Personal Information:
   a) This table holds a student's personal information such as a personal home address, a birth date and a contact number. It also stores a Boolean variable called ApplicationCompleteStatus which, when enabled, allows the application of that student to be viewed by administrators for the particular universities involved.

2) Education Information:
   a) This table holds more information regarding the student's personal information. Since we   do a lot of computation and visualization with our student data, we decided to separate out education from student personal information. Our database will maintain a one to one (1 - 1) relationship between the Education information and Student personal information tables. Education Information holds scores of a particular student such as the SAT and ACT scores and High school GPA's.

3) College Table:
   a) This table holds entries of all the colleges which exist on our system. A college can register itself on our system through an administrator. The administrator will be created before a college is created on the college table. College specific questions regarding the applications can also be set in the college table. Information about the college, which is common to all students can be stored in the college table.

4) Current_Application:
   a) This table serves as a way to have a many to many relations between the college table and the student personal information. The table contains a column to hold the Primary Key for the College Table as well as for the Student Personal Information. This means that one student can be associated with many colleges and 1 college can be associated with many students. The table also keeps track of the students major and the answers to the 3 college specific questions.

5) Essay:
   a) This table serves as a way to store many essays for a particular instance of an education information table. 1 record in the Education Information table will be associated with many essays (1 - many) in the Essay table. The essays are stored as a long string of variable length.

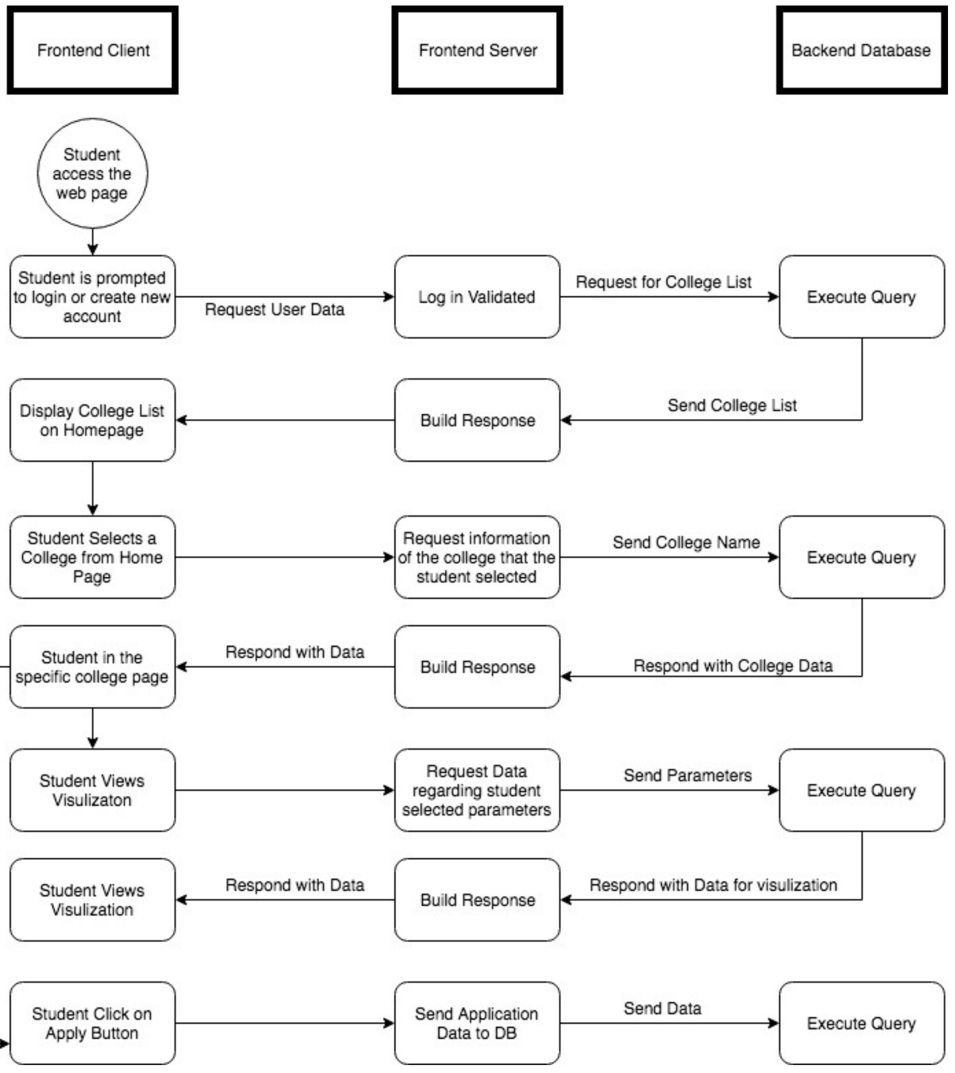
6) Admin:
   a) Each University will have one Administrator, which gets to make the decisions regarding the applicants into the university. The admin account will store admin details such as the name and address of the administrator. It will also have a one to one (1 - 1) relationship with the college table as a way to store the details of the college which the administrator is a part of.

7) The Historical Applicants table contains all the data about the previous applicants. It serves as a common table to store data about all student applicants for all colleges. There is a one to one (1 - 1) relationship between the applicants and the education information table which will help to collect important information without having duplicate information.

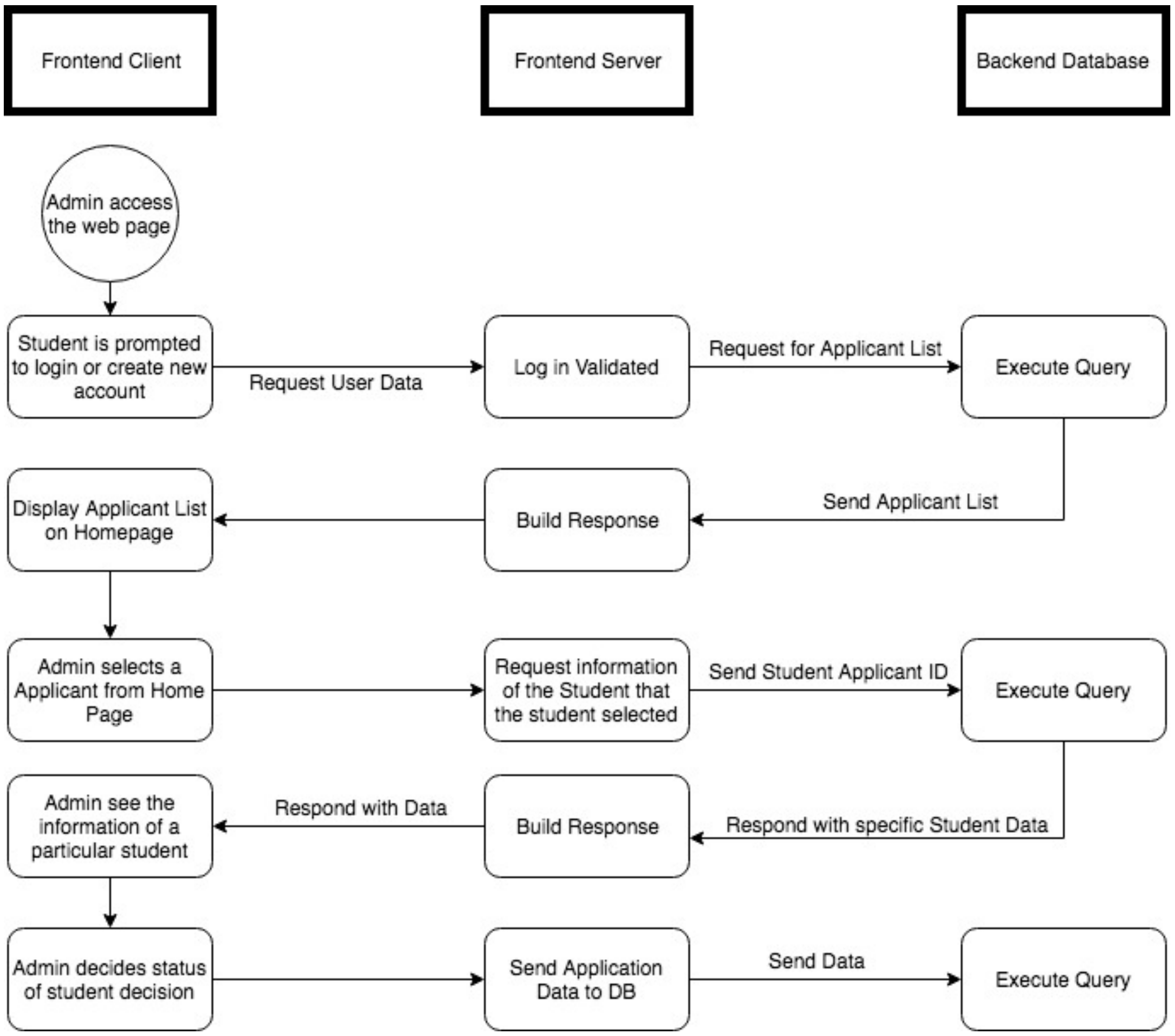
4.3: Broad Overview of Sequence of Events (Sequence Diagram)

The following figure shows an overview of the typical sequence of events that will occur when a user accesses our application; showing the relationship between the client, server, and database. The client requests information from the server, the server will generate the new information if it possesses it. If the server does not have the resources to complete the request, it will send a request to the database for the needed data to complete the request. This basically shows the interaction of the user and how it will interact with system design.

4.3.1 Student Sequence Diagram

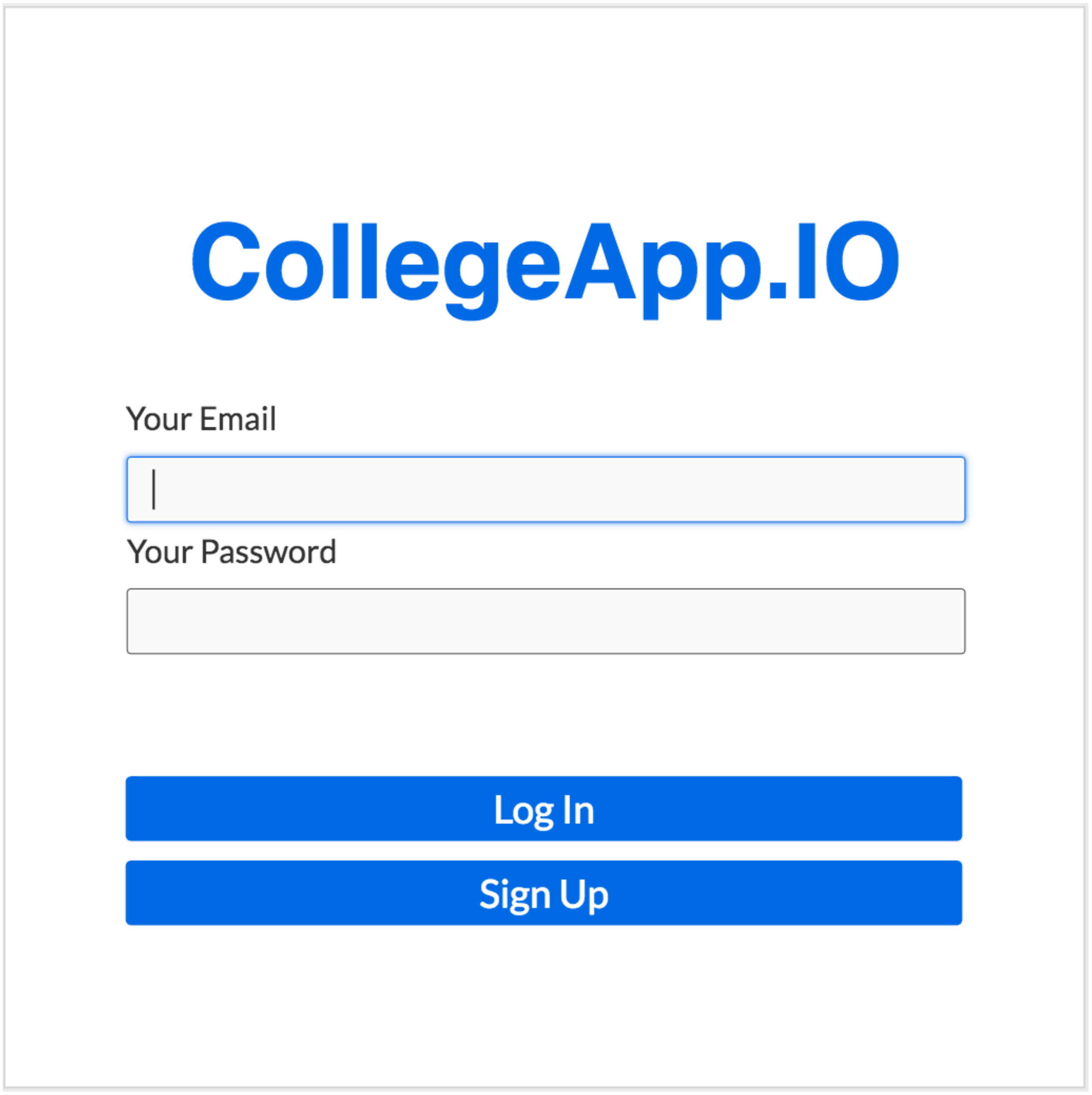

Frontend Client
Frontend Server
Backend Database
Student access the web page
Student is prompted to login or create new account
Request User Data
Log in Validated
Request for College List
Execute Query
Send College List
Display College List on Homepage
Build Response
Student Selects a College from Home Page
Request information of the college that the student selected
Send College Name
Execute Query
Student in the specific college page
Respond with Data
Build Response
Respond with College Data
Student Views Visulizaton
Request Data regarding student selected parameters
Send Parameters
Execute Query
Student Views Visulization
Respond with Data
Build Response
Respond with Data for visulization
Student Click on Apply Button
Send Application Data to DB
Send Data
Execute Query

4.3.2 Admin Sequence Diagram

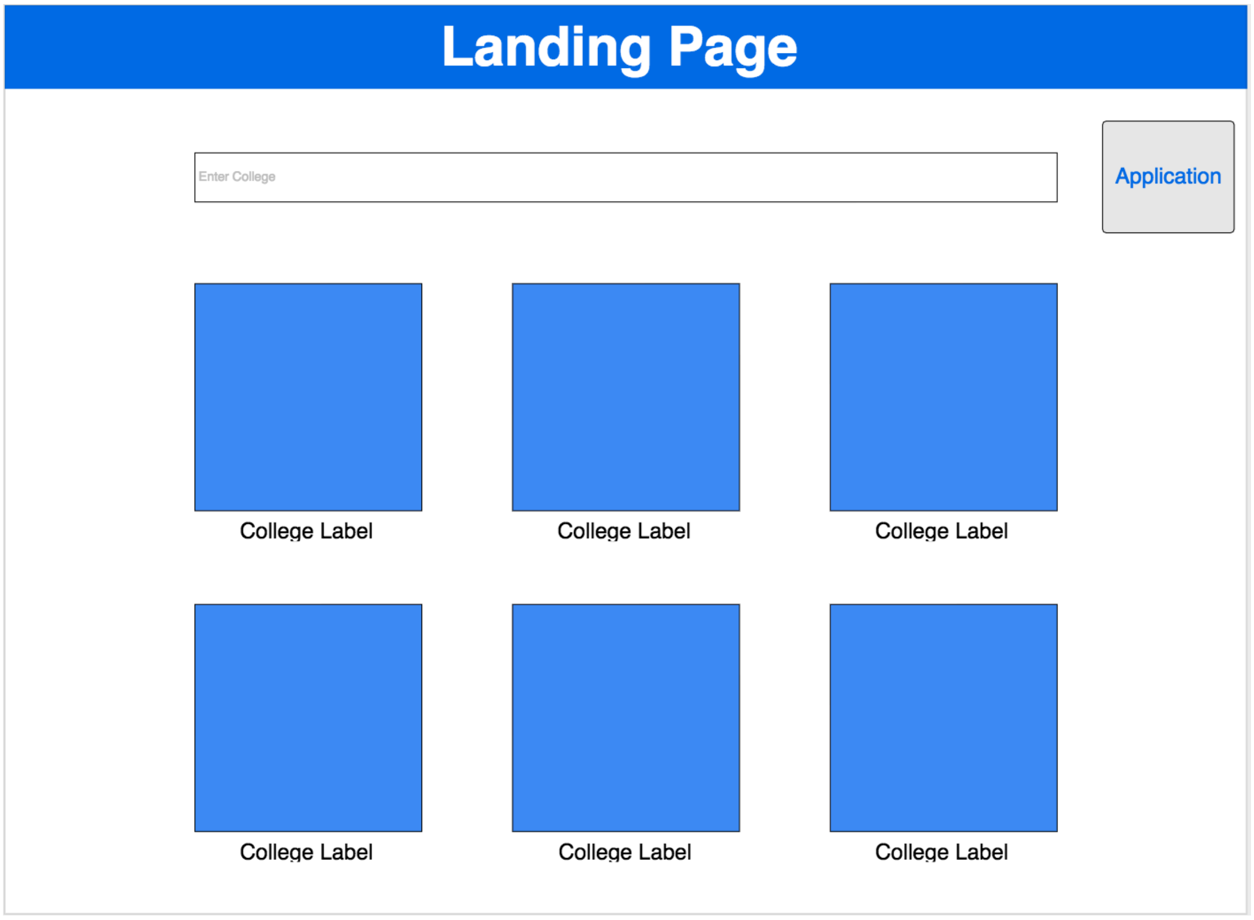

Frontend Client
Frontend Server
Backend Database
Admin access the web page
Student is prompted to login or create new account
Request User Data
Log in Validated
Request for Applicant List
Execute Query
Send Applicant List
Build Response
Display Applicant List on Homepage
Admin selects a Applicant from Home Page
Request information of the Student that the student selected
Send Student Applicant ID
Execute Query
Respond with specific Student Data
Build Response
Respond with Data
Admin see the information of a particular student
Admin decides status of student decision
Send Application Data to DB
Send Data
Execute Query

4.4 UI Mockups

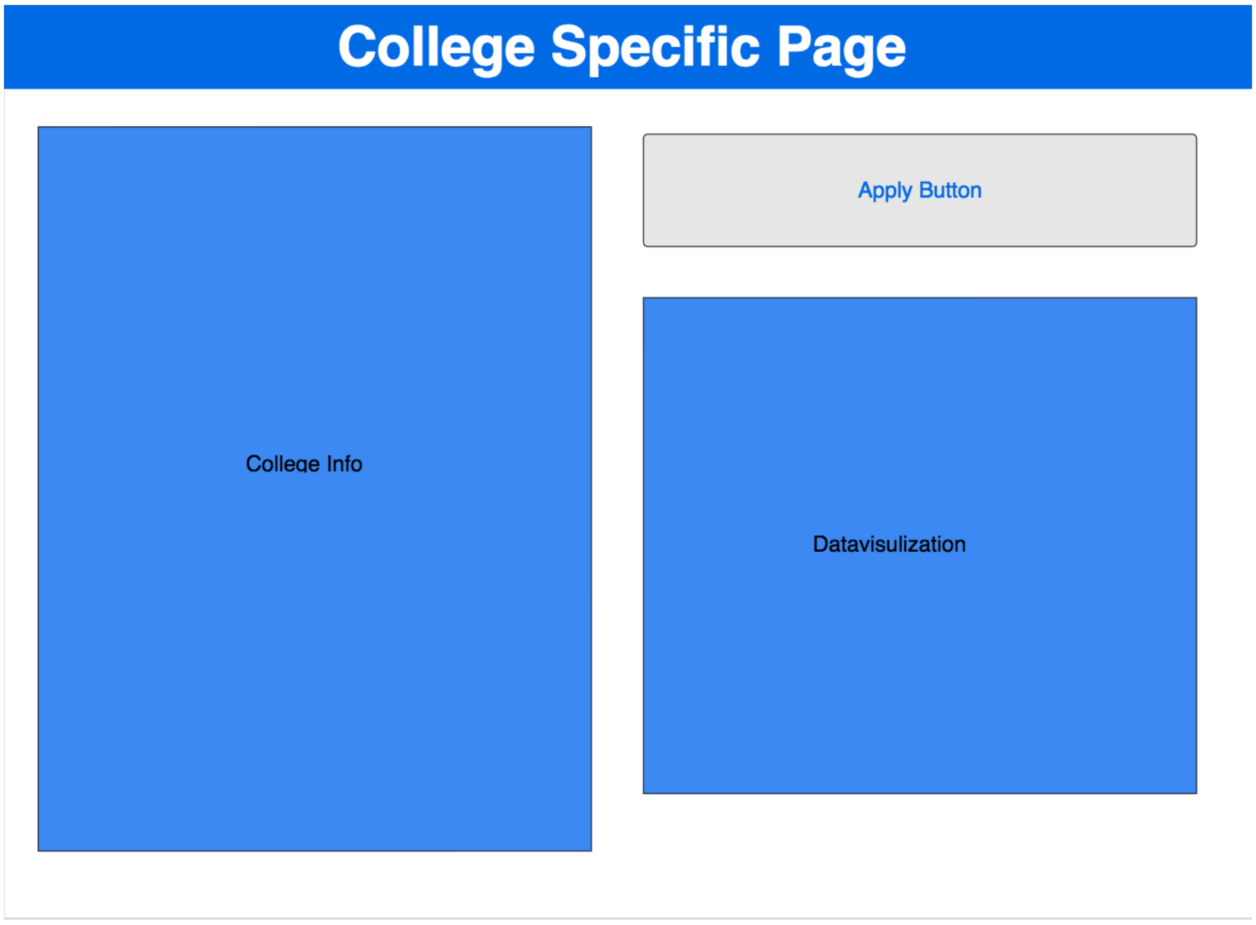
4.4.1 Login Page



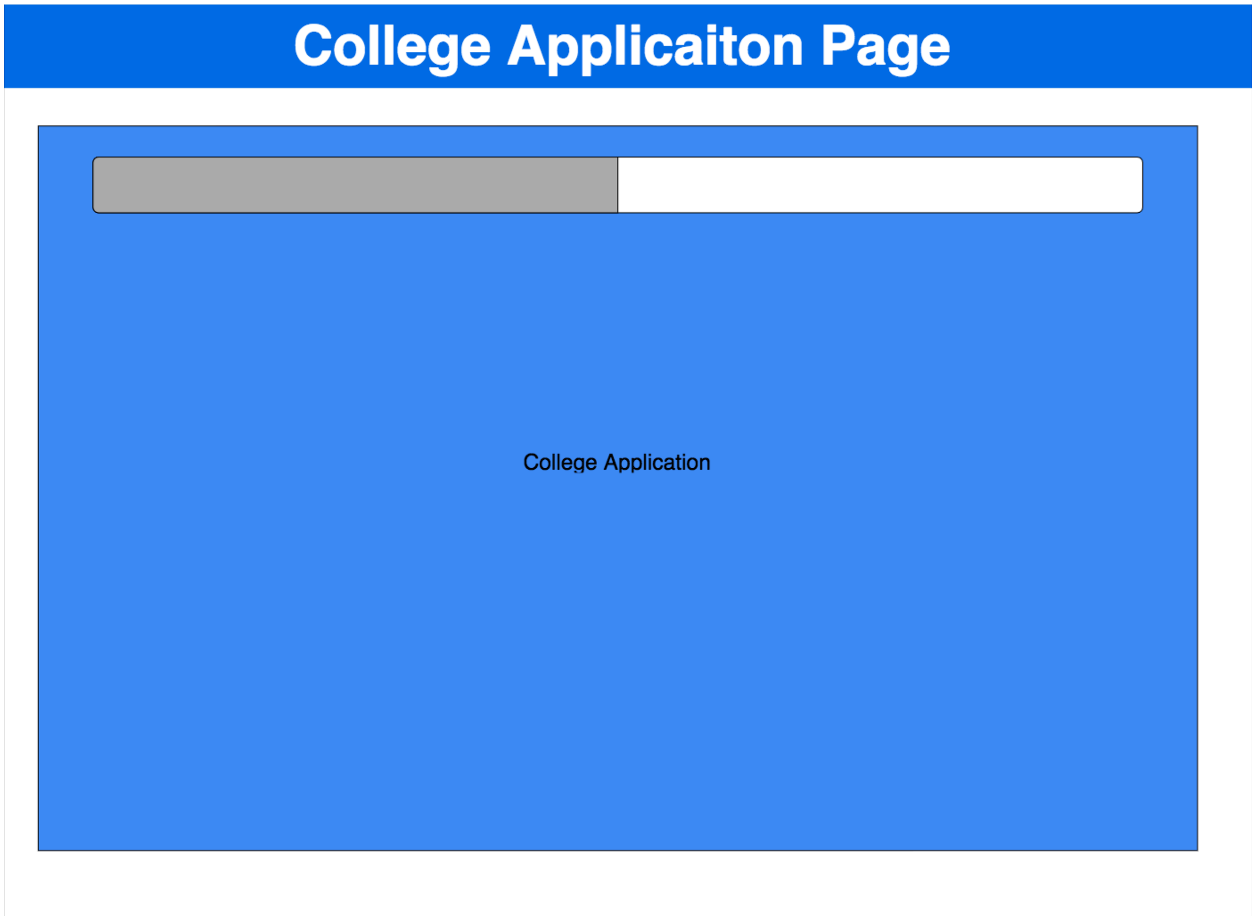
4.4.2 Homepage



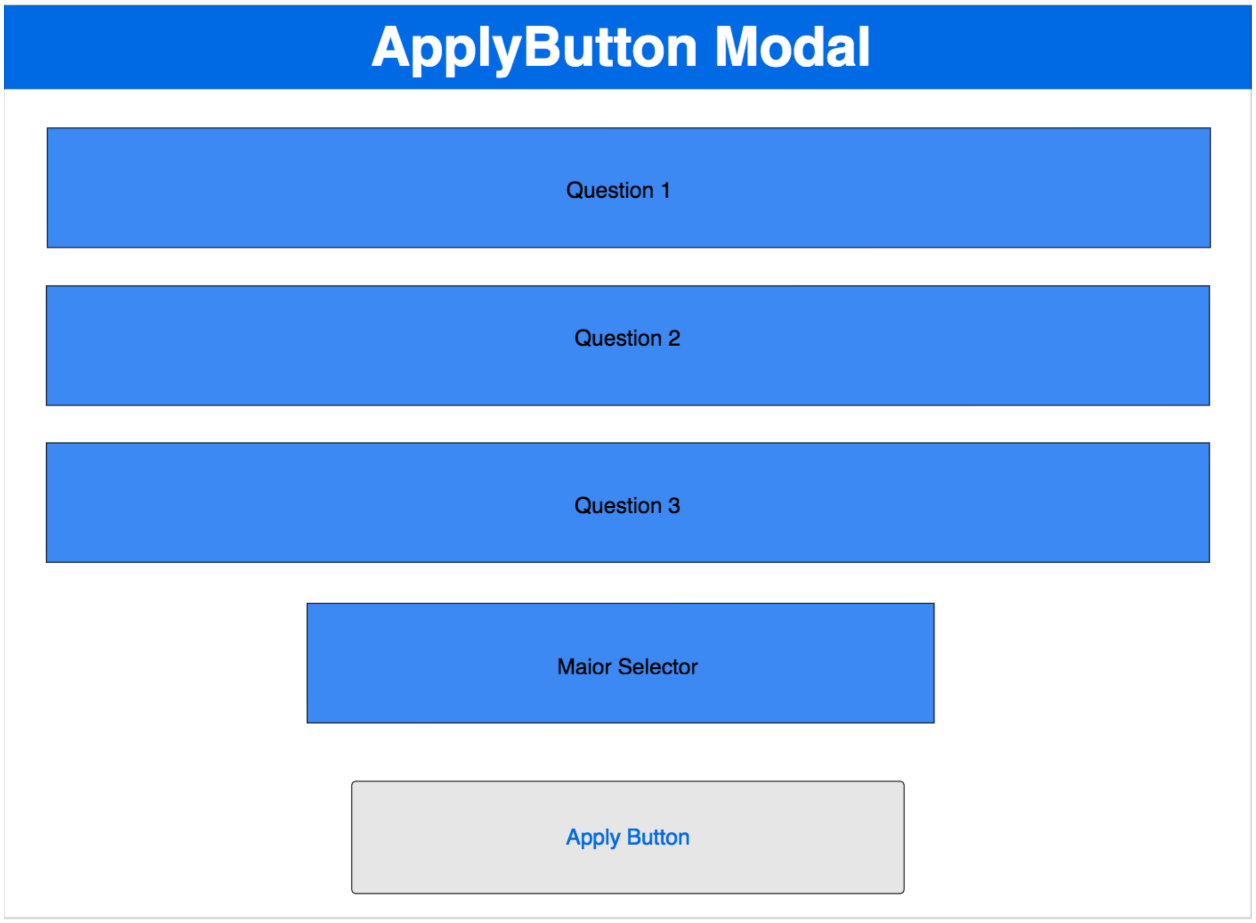
4.4.3 College Specific Page



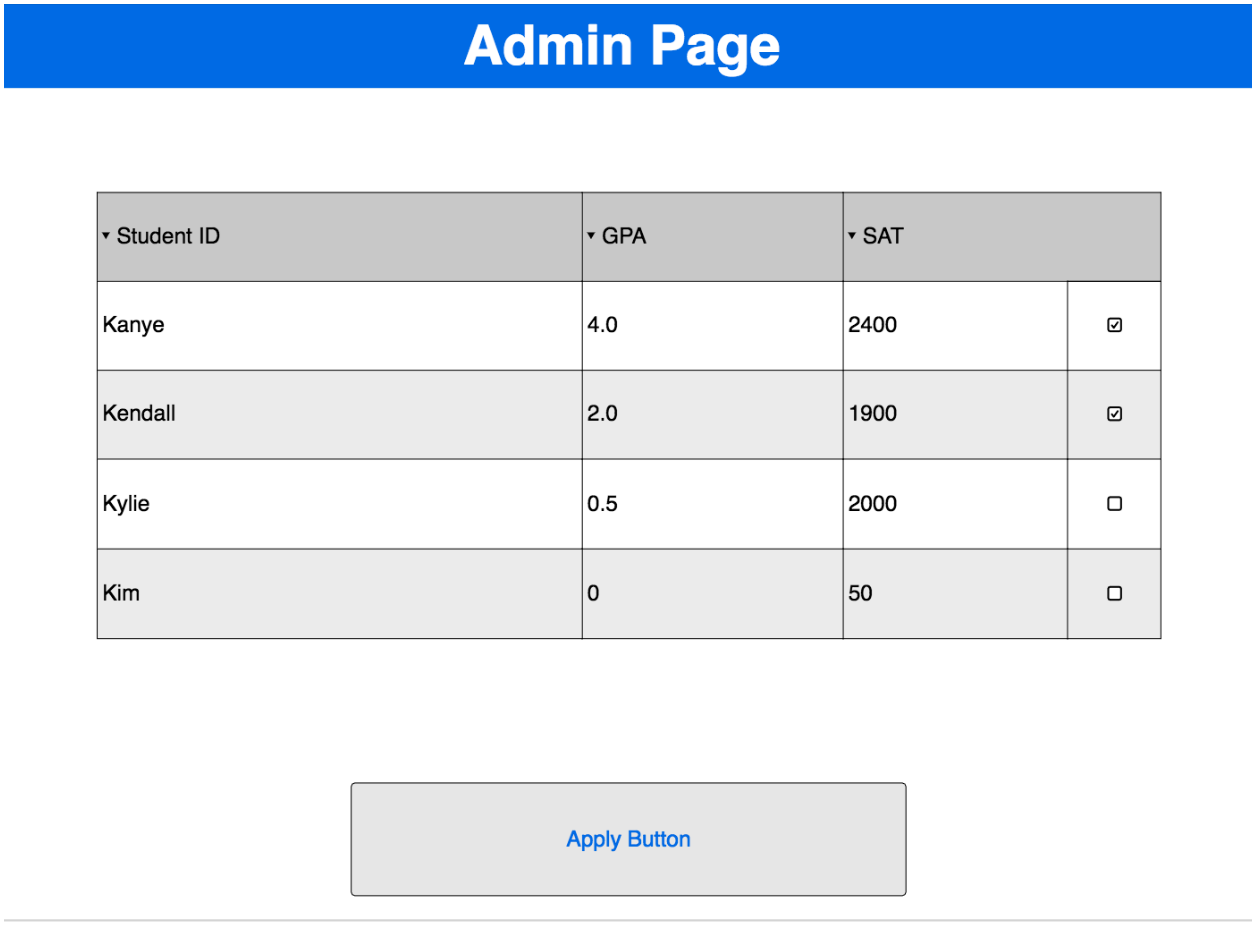
4.4.4 College Application Page



4.4.5 College Apply Modal



4.4.6 Admin Page



Admin Page

▾ Student ID	▾ GPA	▾ SAT	
Kanye	4.0	2400	☑
Kendall	2.0	1900	☑
Kylie	0.5	2000	☐
Kim	0	50	☐