The Walruses

Kenny Liang (kenny.liang.1@stonybrook.edu)
Adam Tringali (adam.tringali@stonybrook.edu)
Dylan Singh (dylan.singh@stonybrook.edu)
Nathan Melnyk (nathan.melnyk@stonybrook.edu)

Table Of Contents

Overview, Functional Requirements, and Other Requirements

1. Overview

c4me helps students decide where to apply for college. Its functionality is similar to several successful commercial software systems such as Naviance (licensed by many high schools) and commercial websites such as CollegeData.com (a service provided by 1st Financial Bank). c4me is for students planning to get a Bachelor's degree. An interesting challenge, beyond the scope of this project, is to make a flexible system that can easily be specialized to be suitable for students looking for different kinds of schools and degrees: undergraduate school, graduate school, law school, medical school, etc.

---

2. Data Sources: Colleges

To keep the dataset size manageable, the system stores data only for colleges listed in the configuration file colleges.txt, available on Blackboard. Some data sources might use variants of some colleges' names; the system must deal with this.

College Scorecard data files. https://collegescorecard.ed.gov/data/. Download and use the most recent institution-level data. From the DOCUMENTATION tab, download the Data Dictionary (brief description of the meaning of each field) and the Technical Documentation for Institution-Level Data Files (detailed description of the meaning of each field).

CollegeData.com. https://www.CollegeData.com/. The URL for the overview page for a college can be constructed automatically from the college's name. For example, the overview page for SBU is https://www.CollegeData.com/college/Stony-Brook-University/.

WSJ/THE College Rankings. The complete Wall Street Journal/Times Higher Education (WSJ/THE) 2020 College Ranking is available at https://www.timeshighereducation.com/rankings/united-states/2020#!/page/0/length/-1/sort_by/rank/sort_order/asc/cols/stats. I recommend this ranking mainly because it is easy to scrape, transparent (details of the methodology are public), and covers many colleges. I don't endorse its methodology over others.

---

3. Data Sources: Standardized Tests

Convert standardized test scores to percentiles. These are merely the first resources I found; if you find better ones, let me know.

https://collegereadiness.collegeboard.org/pdf/understanding-sat-scores.pdf

https://www.act.org/content/dam/act/unsecured/documents/MultipleChoiceStemComposite.pdf

https://secure-media.collegeboard.org/sat/pdf/sat-subject-tests-percentile-ranks.pdf

---

4. Data Sources: High Schools

Scraping data for all high schools at once would be very inefficient. Data about a high school should be scraped on-demand (i.e., after the high school is first mentioned in a student profile) and only once per high school. A few experiments with the search function shows that the URL for pages for a particular high school can be constructed from known information.

Niche High School Ranking. Rankings and other information about a high school's academic quality are available at https://www.niche.com/k12/rankings/. Sample URLs for a particular high school are https://www.niche.com/k12/ward-melville-senior-high-school-east-setauket-ny/ for general information and (by clicking on "More about ... academics" on that page) https://www.niche.com/k12/ward-melville-senior-high-school-east-setauket-ny/academics for more information about academics.

5. Data Sources: Student Profiles

A student profile dataset is stored in a pair of csv files: a students file, and an applications file. Each file has a header row containing field names, followed by data rows. A field value may be a string containing a comma, in which case the value is enclosed in double quotes ("). A students file contains the fields: userid, password, residence_state, high_school_name, high_school_city, high_school_state, college_class, major_1, major_2, SAT_math, SAT_EBRW, ACT_English, ACT_math, ACT_reading, ACT_science, ACT_composite, SAT_literature, SAT_US_hist, SAT_world_hist, SAT_math_I, SAT_math_II, SAT_eco_bio, SAT_mol_bio, SAT_chemistry, SAT_physics, num_AP_passed. password is the user's password in plaintext; storing plaintext passwords in these data files is acceptable, because these data files are intended only for test data, not production data. college_class is the expected year of college graduation. major_1 and major_2 are prospective college majors. num_AP_passed is the number of AP tests passed (i.e., score of 3 or higher). For simplicity, the system omits SAT foreign language subject tests and details of AP tests. An applications file contains the fields: userid, college, status. Each row indicates a college the student applied to, and the current status of the application. Possible values of status are: pending, accepted, denied, deferred, wait-listed, withdrawn.

Fields in the student profile database include, but are not limited to, fields in student profile data files. Inessential personal information (first name, last name, address, email, financial situation) is omitted from profiles, because all information in profiles is public. Students can omit information they don't want to share; almost all information in the profile is optional. Note that a student's financial situation can be taken into account using the cost-of-attendance filter.

6. Functionality for Students

6.1 Create account. This allows a new user to create an account by choosing a username and password. This does not require administrator approval.

6.2 Edit profile. A student can edit his/her profile, e.g., to update the status of a college application. When a student enters an acceptance decision, the system checks for statistical consistency between that decision and other information in the student's profile. If the acceptance decision is statistically unlikely based on other information in the student's profile, the system marks it as questionable and indicates this to the student. Questionable acceptance decisions are ignored by the system when computing all query results; for example, they are omitted from scatterplots. This provides some benefit even though information in student profiles is self-reported, provided most students are honest. If a student submits fake GPA, fake test scores, and fake acceptance decisions that are, taken together, statistically similar to correct data in honest students' profiles), the fake data won't significantly skew query results.

6.3 Search for colleges. This displays a list of colleges satisfying the specified search criteria, a.k.a. filters. The user can specify whether to use a strict or lax interpretation of filter conditions. Strict means that filter conditions involving missing data are treated as false; lax, that they are treated as true. Each filter does nothing in its default state; it limits the search results only if the user sets another value for it. The list can be sorted in various ways, including by name, admission rate, cost of attendance, and ranking. The following information is provided for each college, either directly in the list, or by selecting a college in the list to view additional information about it: institution type (public, private nonprofit, or private for-profit, from College Scorecard field CONTROL), admission rate, completion rate (from CollegeData.com field "Students Graduating within 4 years"), cost of attendance, median completed student debt (from College Scorecard field GRAD_DEBT_MDN), ranking, and size.

Supported filters include:

- admission rate: range for percentage of applicants admitted. The system gets admission rate from College Scorecard field ADM_RATE.
- cost of attendance: upper bound on the cost of attendance. For public state institutions, the in-state or out-of-state cost should be used as appropriate, based on the user's state of residence. The system gets cost of attendance from CollegeData.com.
- location: a list of states or a region in which the college should be located. A region is a shorthand for a set of states. For simplicity, use the four regions defined by the U.S. Census Bureau: Northeast, Midwest, South, and West.
- majors: up to 2 majors that should be offered. The system gets the list of undergraduate majors at a college from CollegeData.com. CollegeData.com sometimes uses long names for majors, e.g., "English Language and Literature, General", and "Mathematics, General". The system should treat these as equivalent to common shorter names, e.g., "English" and "Mathematics".
- name: a substring (not necessarily a prefix or suffix) of the college's name.
- ranking: range for the college's ranking.
- size: range for number of undergraduate students. The system gets that number from College Scorecard field UG if it is non-null otherwise from College Scorecard field UGDS.
- SAT Math, SAT EBRW, ACT Composite: range for average scores for enrolled freshmen. The systems gets this information from CollegeData.com; if CollegeData.com doesn't report the average, and it does report the range of middle 50%, use the midpoint of that range.

Although the original source of each data item is listed above, this function uses the copy of that data in the system's database.

6.4 College recommender. When viewing results of a college search, the student can ask the system to compute a recommendation score for each college in the search results, and sort the list by that score. The recommendation is based on where students with similar profiles applied and possibly other information. The student can select a college in the list to see the profiles of those similar students (to see how similar they are). Recommendation scores are computed upon request, rather than by default, because the computation is relatively expensive.

6.5 Find similar high schools. To get the most meaningful results from the applications tracker, a user needs to identify high schools similar to his/her own. This function helps the user do that. The user specifies a high school by a substring of its name and its zip code, and the system displays a list of the most similar high schools, with relevant information about them, sorted in descending order by similarity. The similarity metric should be based on a variety of relevant information about the academic quality of the high school, including information from niche.com (possibly including but not limited to the school's ranking) and information from this system (for example, standardized test scores or acceptance decisions in profiles of students from the high school).

6.6 Applications tracker (a.k.a. admissions tracker). For a specified college, the system displays information about other students who applied to the specified college and meet specified filter conditions. The user can specify whether to use strict or lax filtering. Supported filters include:

- college class: range of years of expected college graduation.
- high schools: a list of high schools, each identified by a substring of the name and by the zip code.
- application status: a subset of the possible statuses.

Information about matching student profiles may be displayed in two forms.

6.6.1 List of matching student profiles, sorted by application status, with key information from each, and a statistical summary of the listed profiles, specifically, average GPA, average SAT Math, average SAT EBRW, and average ACT composite, reported for all matching users, and for matching users with application status = accepted. The user can select a student profile to see all information in it.

6.6.2 Scatterplot of application status of matching student profiles. Each matching student is represented by a point on the scatterplot. The color of the point indicates the student's application status; for example, green = accepted, red = denied, yellow = other. The mean values on the horizontal and vertical axes are shown by a dashed vertical line and a dashed horizontal line, respectively. The vertical axis is GPA. The horizontal axis is based on standardized test scores. The user can choose between SAT (Math+EBRW), ACT Composite, or weighted average of percentile scores for standardized tests (except AP tests). The weights used in the weighted average are: 5% for each SAT subject test taken, and the remainder for SAT or ACT Composite (or split evenly between SAT and ACT Composite, if the student took both).

---

7. Functionality for Administrators

All scrape and import functions overwrite old information in the system's database, if any.

7.1 Scrape college rankings. Scrape WSJ/THE 2020 rankings of all colleges in colleges.txt.

7.2 Import College Scorecard data file. Import information about all colleges in colleges.txt.

7.3 Scrape CollegeData.com. Scrape information about all colleges in colleges.txt.

7.4 Delete all student profiles.

7.5 Import student profile dataset.

7.6 Review questionable acceptance decisions. The system displays student profiles with questionable acceptance decisions. The administrator may validate some questionable acceptance decisions, so that they are no longer marked as questionable.

---

8. Other Requirements

8.1 Authentication. All access to the system requires authentication with a password. Passwords stored in the system database are salted and hashed. It is sufficient to have one pre-defined administrator account, with a fixed username and password. The system does not need to support creation of additional administrator accounts.

8.2 Concurrency. Synchronization is used to ensure correct behavior when multiple users access the system concurrently.

8.3 Configuration. It should be easy (e.g., by editing one line in one file) to change the hostname and path prefix of URLs for all data source websites, for example, to change from https://CollegeData.com/ to https://www.cs.stonybrook.edu/CollegeData/. I might create partial mirrors of some data source websites, to ensure availability and consistency of the content during demos.

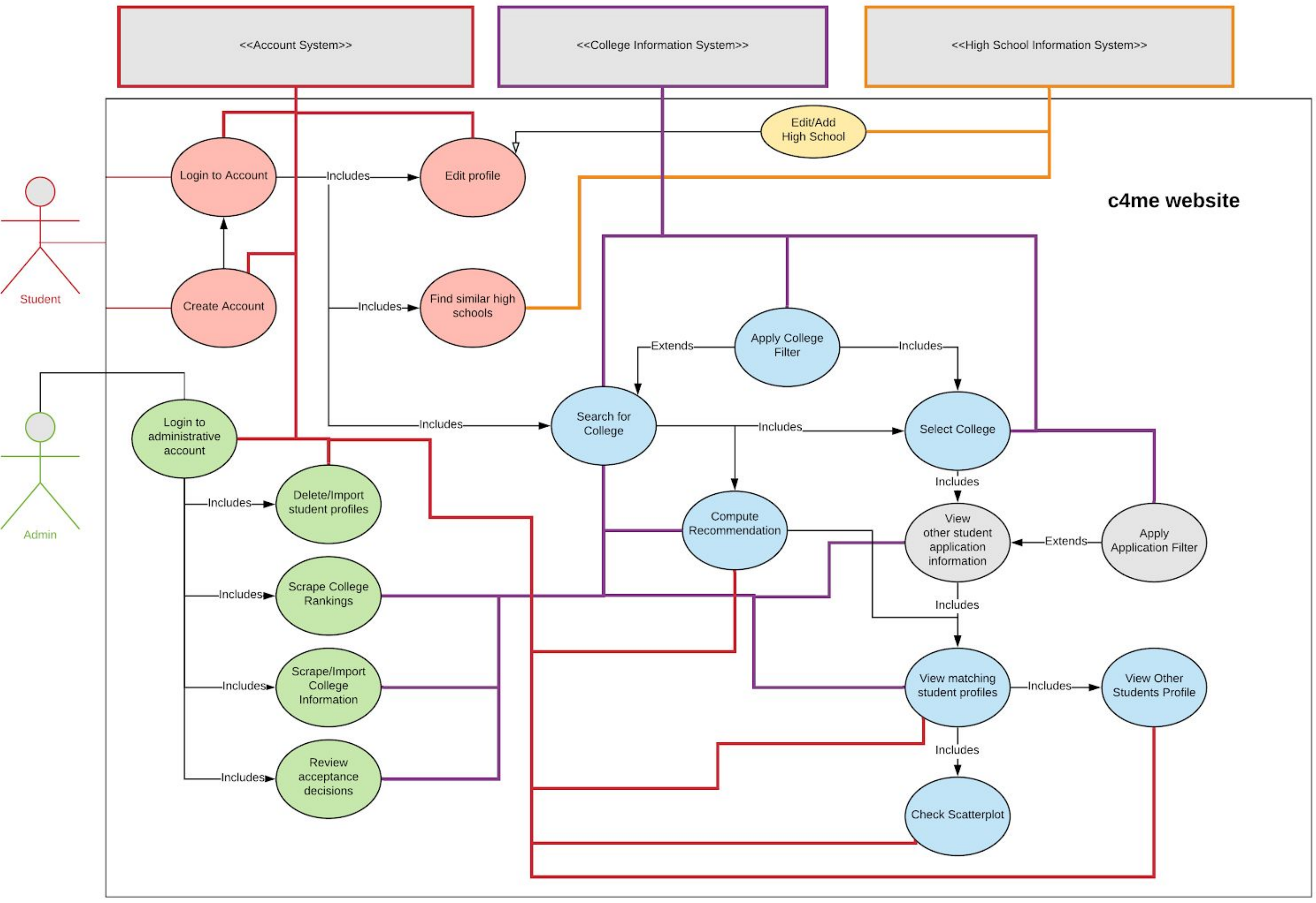
8.4 Ease of use. The system provides an easy-to-use, user-friendly web interface consistent with established UI design principles. The system handles invalid inputs gracefully, provide helpful feedback to the user when appropriate, etc.

8.5 Multi-host operation. The client and server can run on different hosts.

8.6 Network security. Network communication is secured using HTTPS or SSL. If your server does not have a public-key certificate signed by a certification authority trusted by the web browser, the web browser will show a security warning. The warning can be eliminated by creating a self-signed certificate, and installing the key in the browser, but that is optional. Requiring the user to tell the browser to proceed despite the security warning is acceptable.

8.7 Robustness. The system handles missing information gracefully. For example, almost all information in a student profile is optional (students should omit information they don't want to share, since all information in the profile is public), and some information about colleges is missing on CollegeData.com (it's shown as "Not reported"). The system handles unreachable data sources gracefully, for example, if CollegeData.com is unreachable when scraping, or niche.com is unreachable when querying high school information.

Use Case Diagram



<<Account System>>
<<College Information System>>
<<High School Information System>>
c4me website
Student
Admin
Login to Account
Edit profile
Edit/Add High School
Create Account
Find similar high schools
Apply College Filter
Search for College
Select College
Login to administrative account
Delete/Import student profiles
Compute Recommendation
View other student application information
Apply Application Filter
Scrape College Rankings
Scrape/Import College Information
View matching student profiles
View Other Students Profile
Review acceptance decisions
Check Scatterplot
Includes
Extends

Figure A: Use Case Diagram

Use Cases

Use Case Name	Create Account
Primary Actor	Student
Secondary Actor	**Account System**
Primary Flow	1. Student presses Sign Up link 2. Web client directs to register page 3. Web client prompts student for username and password 4. Student enters username and password 5. Student presses Register button 6. Web client sends registration request to **Account System** 7. **Account System** sends approval of registration 8. Web client directs to login page
Alternative Flow	**Existing Username** Precondition: The username already exists in the **Account System** 6.1 **Account System** sends “Existing Username” message 6.2 Web client displays “A username with that username already exists”.

Use Case Name	Login to Account
Primary Actor	Student
Secondary Actor	**Account System**
Pre-condition	Student is logged out and has registered an account.
Primary Flow	1. Student presses login link 2. Web client directs to login screen 3. Web client prompts student for username and password 4. Student enters username and password 5. Student presses login button 6. Web client sends login request to **Account System** 7. **Account System** sends approval of login request 8. Web client directs to profile page
Alternative Flow	**Nonexistent username** Pre-condition: The entered username does not exist in the account system. 6.1 **Account System** sends “Nonexistent username” 6.2 Web client displays “The username does not exist” **Invalid password** Pre-condition: The entered password does not match the password corresponding to the username 6.1 **Account System** sends “Invalid password” 6.2 Web client displays “The password is incorrect”

Use Case Name	Edit profile
Primary Actor	Student
Secondary Actor	Account System
Pre-condition	Student is logged in and is viewing the profile page
Primary Flow	1. Student presses Edit Profile link 2. Web client redirects to edit screen 3. Web client sends user info request to account system 4. Account system sends corresponding user info 5. Web client displays user info in editable form (except userid) 6. Student updates one or more textfield 7. Student presses save button 8. Web client sends updated user info to Account System 9. Account System updates user info 10. Web client directs to profile page

Use Case Name	Edit/Add High School
Primary Actor	Students
Secondary Actor	High School Information System
Pre-condition	Student is logged in and is viewing the edit profile page.
Primary Flow	1. Web client displays user info in editable form 2. Student changes textfield containing the existing high school 3. Student presses save button 4. Web client sends updated user info to **Account System** 5. Account System updates user info 6. Web client directs to profile page
Alternative Flow	**High School Doesn't Exist** Precondition: The user doesn't have a high school saved in the **Account System** 3.1 Student clicks the "Add High School" button. 3.2 The web client adds a field for High School to the editable fields

Use Case Name	Find similar high schools
Primary Actor	Students
Secondary Actor	High School Information System
Pre-condition	Student is logged in and is viewing the profile page.
Primary Flow	1. Student presses Find similar high schools button 2. Web client directs to similar-school page 3. Web client prompts the student for a high school 4. Web client sends similar-school request to the High School Information System along with the user specified high school 5. High School Information System responds with a list of the most similar high schools that are known in the system 6. Web client displays the information received from the High School Information System
Alternative Flow	**High school not found** Pre-condition: User enters a high school that is not found in the system. 6.1 High School Information System sends “High school not found” 6.2 Web client displays “The specified high school was not found”

Use Case Name	Search for college
Primary Actor	Students
Secondary Actor	College Information System
Pre-condition	Student is logged in.
Primary Flow	1. Student presses Search for College button 2. Web client redirects to search college screen 3. Web client prompts the student for a college 4. Web client sends a search request to the College Information System with the user specified college 5. College Information System responds with the list of the most similar colleges that are known in the system 6. Student clicks on the one that most resembles their query 7. Web client sends a query to the College Information System of the desired college 8. College Information System sends the data to the Web Client 9. Web Client displays the information received from the College Information System
Alternative Flow	**College Not Found** Precondition: Student enters a college that is not found in the system 5.1 College Information System sends “College not found” 5.2 Web client displays “The specified college was not found”

Use Case Name	Compute Recommendation
Primary Actor	Students
Secondary Actor	Account System, College Information System
Pre-condition	Student is logged in and on Search for College Page.
Primary Flow	1. Student presses the Compute Recommendation button. 2. Web client sends a query to the Account System to gather the student’s data 3. The Account system sends the desired data to the web client 4. Web client sends a query to the College Information System with Student’s data 5. College Information System responds with a list of the best matching colleges 6. Web client displays the information received from the College Information System

Use Case Name	Apply College Filter
Primary Actor	Students
Secondary Actor	College Information System
Pre-condition	Student is logged in and on Search for College page.
Primary Flow	1. Web client is displaying the Search for College page with default search options enabled 2. Student edits the search filters, including whether or not to use strict or lax filtering. 3. Web client sends a query to the College Information System with the filter request. 4. College Information System responds with a list of the best matching colleges based on filters. 5. Web client displays the information received from the College Information System.

Use Case Name	Select College
Primary Actor	Students
Secondary Actor	College Information System
Pre-condition	Student is logged in and has searched for a college
Primary Flow	1. Student clicks on desired College 2. Web client sends a query to the College Information System with the desired college’s name 3. College Information System responds with the corresponding data 4. Web Client displays the information received from the College Information System

Use Case Name	View other student application information
Primary Actor	Students
Secondary Actor	College Information System
Pre-condition	Student is logged in.
Primary Flow	1. Student executes Select College 2. Student requests to view applications of other students. 3. Web client sends a query to the College Information System with the College name 4. College Information System responds with list of application data 5. Web client displays information received from College Information System

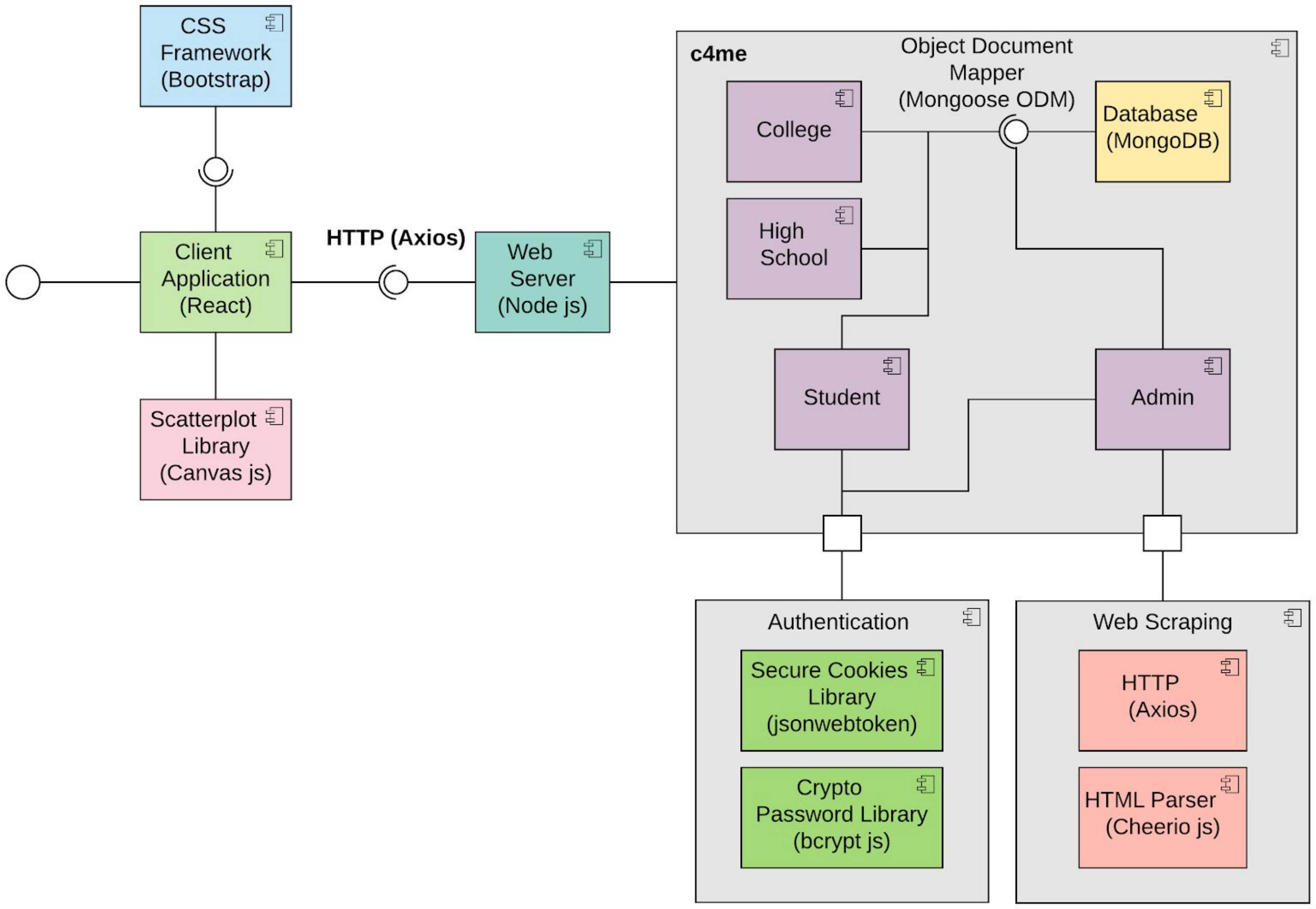
Use Case Name	Apply Application Filter
Primary Actor	Students
Secondary Actor	College Information System
Pre-condition	Student is logged in and have a college selected.
Primary Flow	1. Web client is displaying a list of available filters 2. Student selects desired filters, and whether to use strict or lax filtering. 3. Student clicks search button 4. Web client sends a query to the College Information System with the desired filters 5. College Information System responds with corresponding data 6. Web client displays information received from the College Information System.

Use Case Name	View Matching Student Profiles
Primary Actor	Students
Secondary Actor	College Information System Account System
Pre-condition	Student is logged in and similar student applications have been found.
Primary Flow	1. Student clicks on matching profiles button 2. Web client sends a query to the College Information System to get all username of applications to that college 3. Web client receives list of username and sends a query to the Account System to find matching profiles 4. Account System send matching profiles of the user back to the Web Client 5. Web client displays information received by the Account System

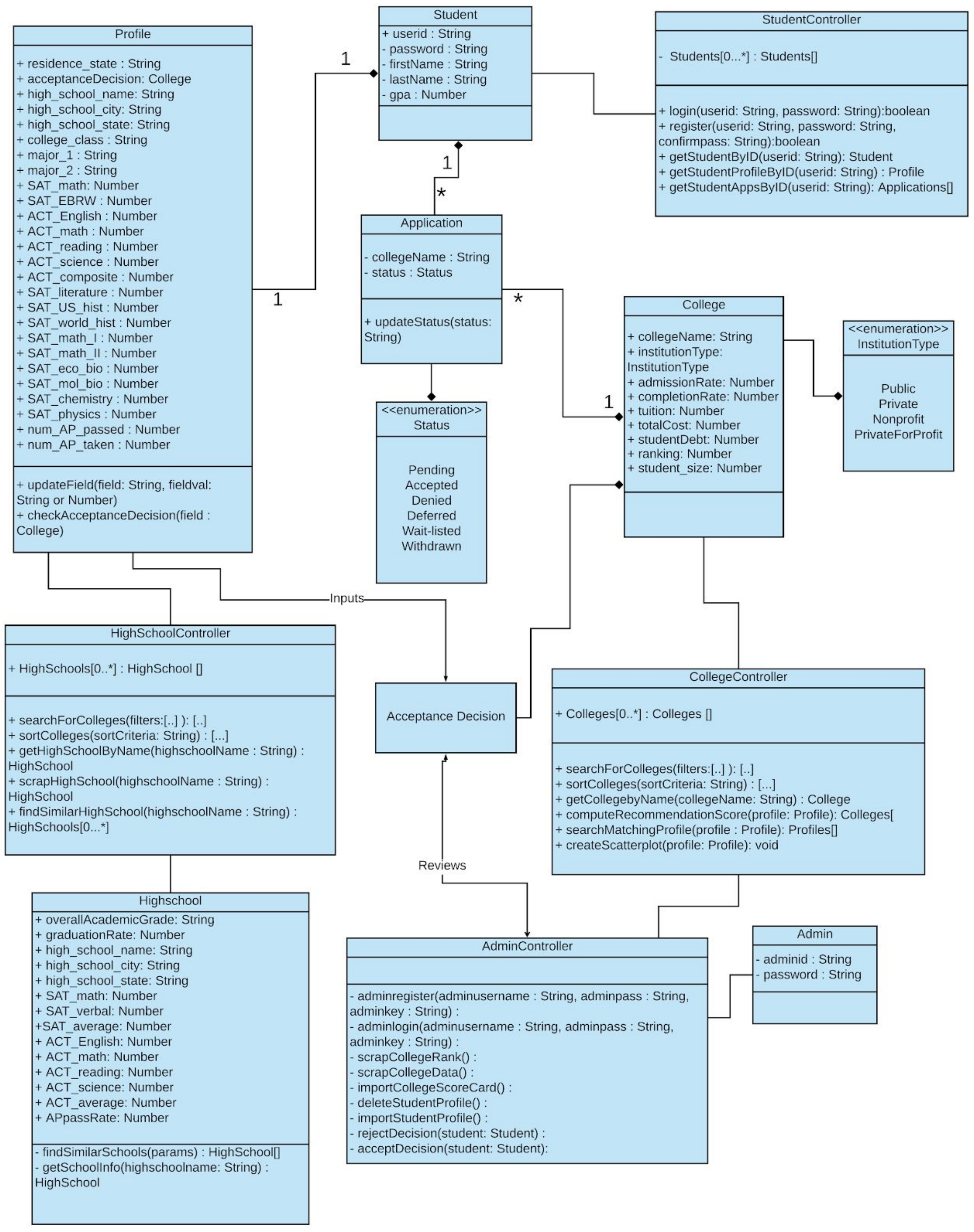
Use Case Name	View Other Student Profiles
Primary Actor	Students
Secondary Actor	Account System
Pre-condition	Student is logged in and have found matching student applications.
Primary Flow	1. Student clicks another students profile 2. Web client sends student userid data to Account System. 3. Account system find the specific student 4. Account system responds to web client with desired student profile 5. Web client displays desired student profile

Use Case Name	Check Scatterplot
Primary Actor	Students
Secondary Actor	Account System
Pre-condition	Student is logged in and have found matching student applications.
Primary Flow	1. Student clicks Generate Scatter Plot button 6. Web client send scatter plot request and matching profiles to Account System 7. Account system generates a scatter plot and send to web client 8. Web client displays scatterplot

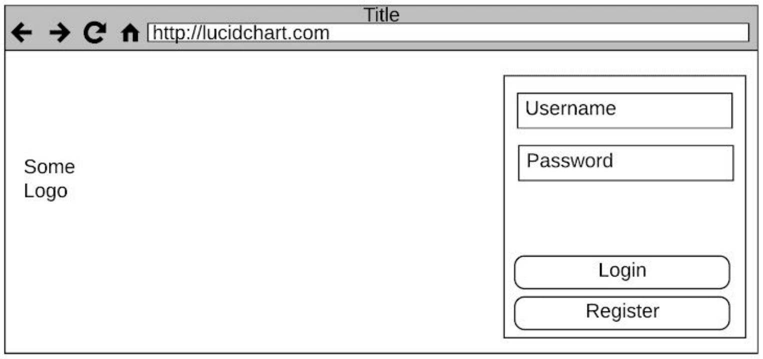
System Architecture


CSS Framework (Bootstrap)
Client Application (React)
Scatterplot Library (Canvas js)
HTTP (Axios)
Web Server (Node js)
c4me
Object Document Mapper (Mongoose ODM)
College
High School
Database (MongoDB)
Student
Admin
Authentication
Secure Cookies Library (jsonwebtoken)
Crypto Password Library (bcrypt js)
Web Scraping
HTTP (Axios)
HTML Parser (Cheerio js)

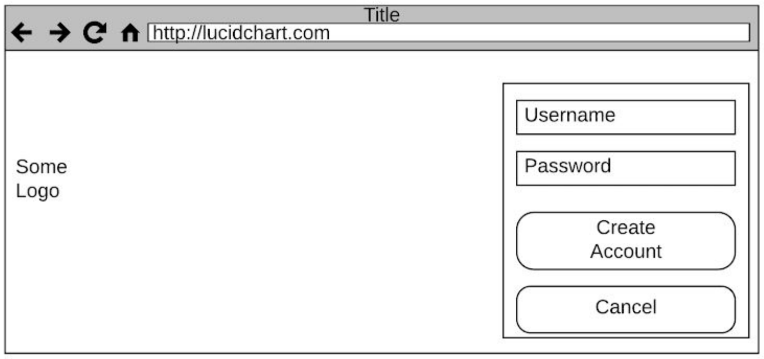
Class Diagram


Profile
+ residence_state : String
+ acceptanceDecision: College
+ high_school_name: String
+ high_school_city: String
+ high_school_state: String
+ college_class : String
+ major_1 : String
+ major_2 : String
+ SAT_math: Number
+ SAT_EBRW : Number
+ ACT_English : Number
+ ACT_math : Number
+ ACT_reading : Number
+ ACT_science : Number
+ ACT_composite : Number
+ SAT_literature : Number
+ SAT_US_hist : Number
+ SAT_world_hist : Number
+ SAT_math_I : Number
+ SAT_math_II : Number
+ SAT_eco_bio : Number
+ SAT_mol_bio : Number
+ SAT_chemistry : Number
+ SAT_physics : Number
+ num_AP_passed : Number
+ num_AP_taken : Number
+ updateField(field: String, fieldval: String or Number)
+ checkAcceptanceDecision(field : College)
1
1
Student
+ userid : String
- password : String
- firstName : String
- lastName : String
- gpa : Number
StudentController
- Students[0...*] : Students[]
+ login(userid: String, password: String):boolean
+ register(userid: String, password: String, confirmpass: String):boolean
+ getStudentByID(userid: String): Student
+ getStudentProfileByID(userid: String) : Profile
+ getStudentAppsByID(userid: String): Applications[]
1
*
Application
- collegeName : String
- status : Status
+ updateStatus(status: String)
*
<<enumeration>>
Status
Pending
Accepted
Denied
Deferred
Wait-listed
Withdrawn
1
College
+ collegeName: String
+ institutionType: InstitutionType
+ admissionRate: Number
+ completionRate: Number
+ tuition: Number
+ totalCost: Number
+ studentDebt: Number
+ ranking: Number
+ student_size: Number
<<enumeration>>
InstitutionType
Public
Private
Nonprofit
PrivateForProfit
Inputs
HighSchoolController
+ HighSchools[0..*] : HighSchool []
+ searchForColleges(filters:[..]): [..]
+ sortColleges(sortCriteria: String) : [...]
+ getHighSchoolByName(highschoolName : String) : HighSchool
+ scrapHighSchool(highschoolName : String) : HighSchool
+ findSimilarHighSchool(highschoolName : String) : HighSchools[0...*]
Acceptance Decision
CollegeController
+ Colleges[0..*] : Colleges []
+ searchForColleges(filters:[..]): [..]
+ sortColleges(sortCriteria: String) : [...]
+ getCollegebyName(collegeName: String) : College
+ computeRecommendationScore(profile: Profile): Colleges[
+ searchMatchingProfile(profile : Profile): Profiles[]
+ createScatterplot(profile: Profile): void
Reviews
Highschool
+ overallAcademicGrade: String
+ graduationRate: Number
+ high_school_name: String
+ high_school_city: String
+ high_school_state: String
+ SAT_math: Number
+ SAT_verbal: Number
+SAT_average: Number
+ ACT_English: Number
+ ACT_math: Number
+ ACT_reading: Number
+ ACT_science: Number
+ ACT_average: Number
+ APpassRate: Number
- findSimilarSchools(params) : HighSchool[]
- getSchoolInfo(highschoolname: String) : HighSchool
AdminController
- adminregister(adminusername : String, adminpass : String, adminkey : String) :
- adminlogin(adminusername : String, adminpass : String, adminkey : String) :
- scrapCollegeRank() :
- scrapCollegeData() :
- importCollegeScoreCard() :
- deleteStudentProfile() :
- importStudentProfile() :
- rejectDecision(student: Student) :
- acceptDecision(student: Student):
Admin
- adminid : String
- password : String

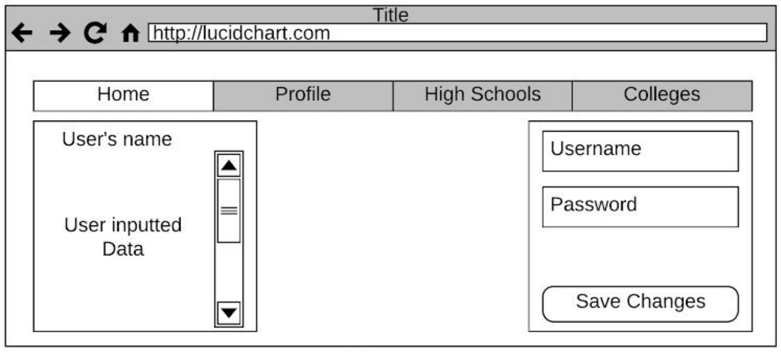
Graphical User Interface and System Flow



Landing Page
Control Flow: User Clicks on Login -> Login Screen
Control Flow: User Clicks on Register -> Register Screen

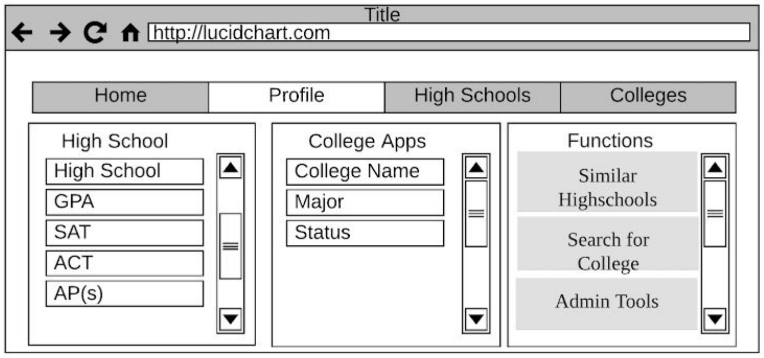


Create Account Page
Control Flow: User Clicks on Create Account -> Main Page
Control Flow: User Clicks on Cancel -> Landing Page

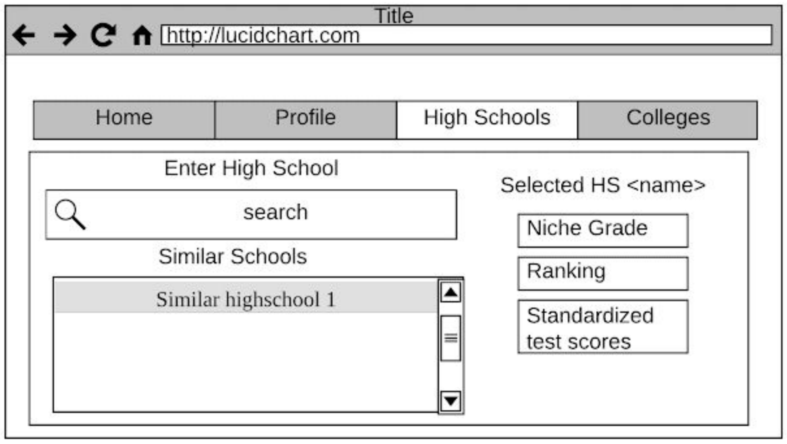


Main Page

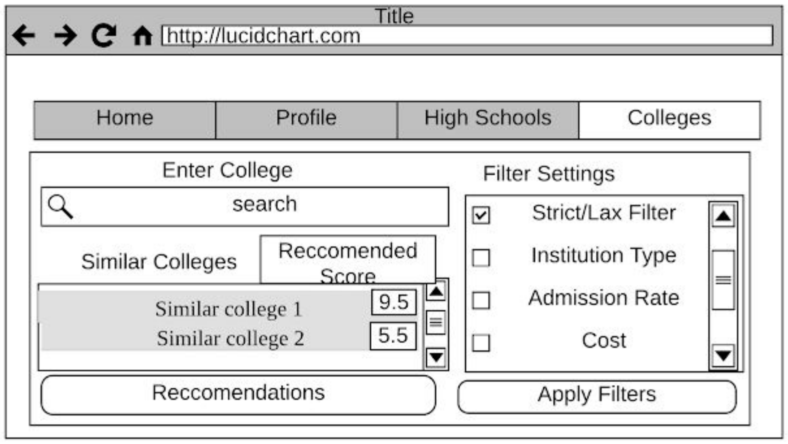
Control Flow: User Uses NavBar to select page -> Selected Page



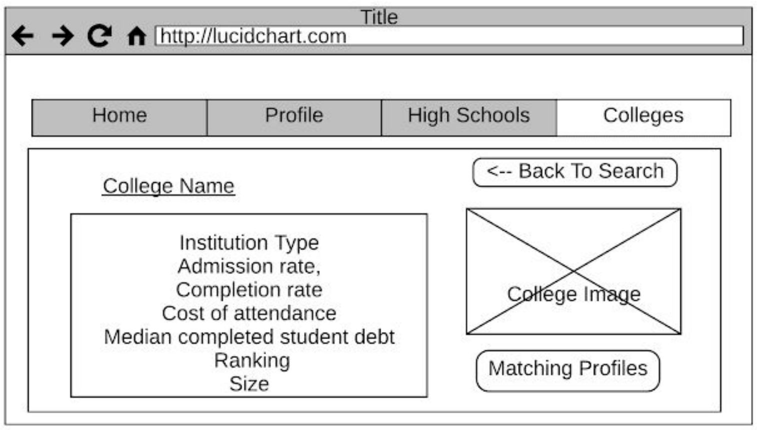
Edit Profile Page
Control Flow: User Uses NavBar to select page -> Selected Page



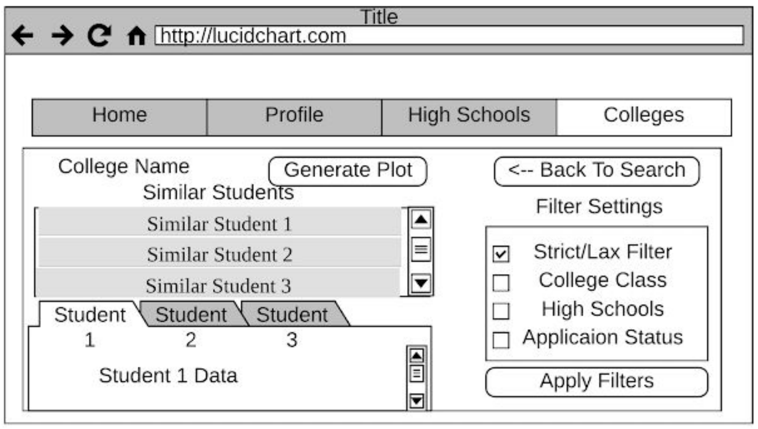
Find similar high school Page
Control Flow: User Uses NavBar to select page -> Selected Page
Control Flow: User clicks on a listed High School -> High School info Page



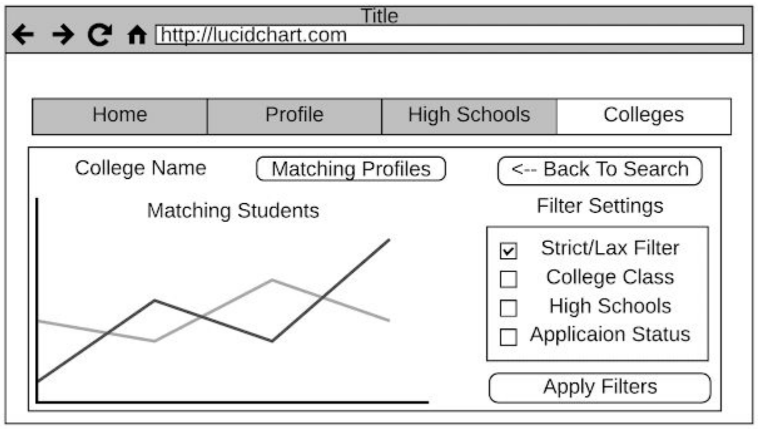
Search for college Page
Control Flow: User Uses NavBar to select page -> Selected Page
Control Flow: User clicks on "Similar college" -> College Page 2



College Info Page
Control Flow: User Uses NavBar to select page -> Selected Page
Control Flow: User clicks on Back to search button -> Search for college page
Control Flow: User clicks on Matching Profile button -> Matching profile page



Matching Profiles Page
Control Flow: User Uses NavBar to select page -> Selected Page
Control Flow: User clicks on Back to search button -> Search for college page
Control Flow: User clicks on Generate Plot button -> Scatterplot page



Scatterplot Page
Control Flow: User Uses NavBar to select page -> Selected Page
Control Flow: User clicks on Back to search button -> Search for college page
Control Flow: User clicks on Matching Profile button -> Matching profile page

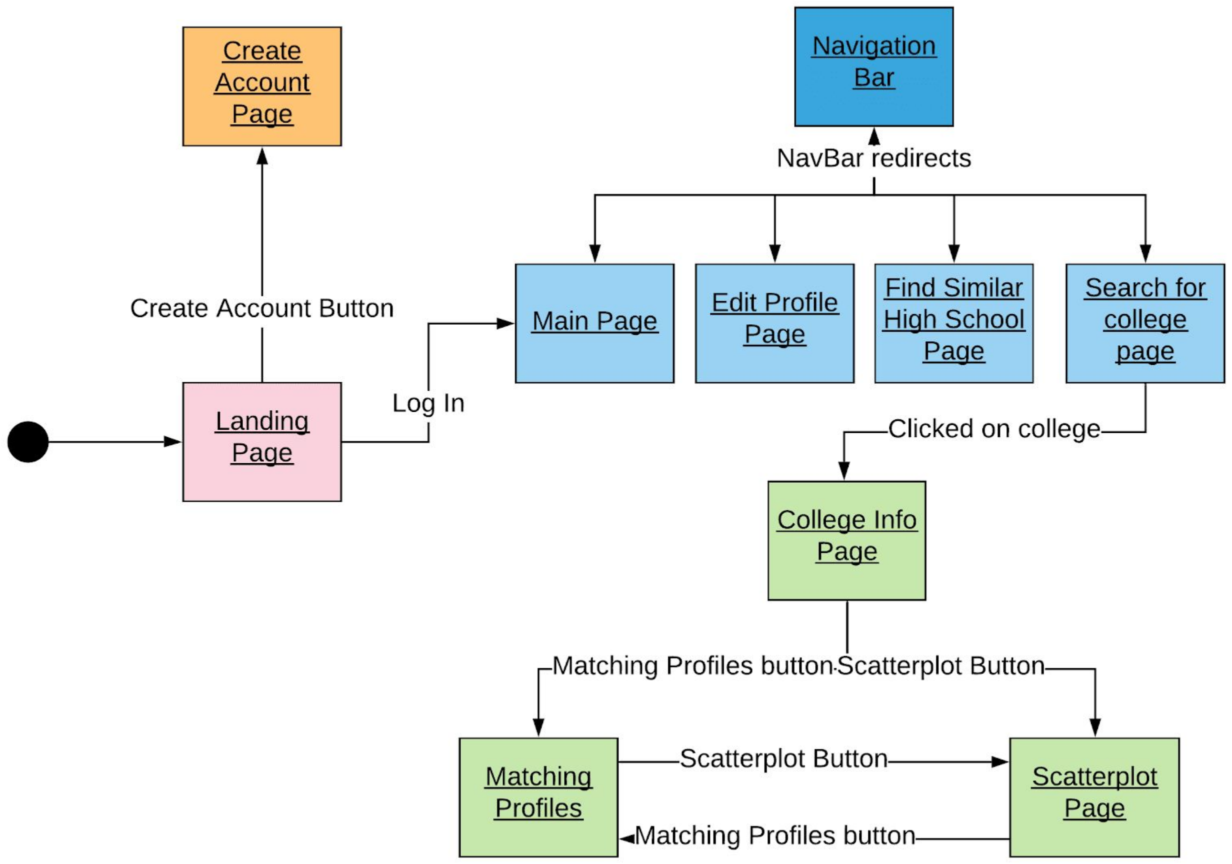


Figure : GUI Control Flow

Languages, Technologies, and Tools

Tools

UML Diagrams were created using **Lucidchart**.

Languages

JavaScript based frameworks and technologies will be used to implement the frontend and backend components.

Technologies

React - JavaScript library for building user interfaces
Express js - a node js based web application framework
Axios - HTTP client for data transfer and communication
MongoDB - highly scalable NOSQL database
Canvas JS - frontend graph library to construct Scatterplots
JWT - JavaScript Web Tokens for secure cookies
Bcrypt JS - library for easy password salting and hashing
Cheerio JS - HTML parser
Bootstrap - CSS framework

Algorithms

Find Similar High Schools

Description: Provides a list of similar high schools to students based on the students high school

Parameters: highschool_collection - a list of high schools

highschool - student high school

Returns: similar_HS_List - a sorted list of colleges based on recommendation score

```
function compute_similar_HighSchools(HighSchool[] hs_collection , HighSchool userHS){
    for(hs in hs_collection){
        Add hs with corresponding similarity score from calculate_similarHS_value(userHs, hs) to
        similar_hs_list
    }
    Sort similar_hs_list by generated similarity rating
    return top 10 high schools in sorted list
}
```

Calculate similar highschool value

Description: Provides a percentage of how similar a high school is to another.

Parameters: hs1- high school we are going to compare

hs2 - high school we are comparing to

Returns: similarity - a weighted percentage of how similar two high schools are

Information on Criteria - Used in function calls(colored)		
Criteria	Total	Weight (%)
Academic Grade	4.0	25%
Graduation Rate	100	20%
State Test Reading	100	15%
State Test Math	100	15%
SAT Score	36	12.5%
ACT Score	1600	12.5%

```
function determine_similarHS_value(HighSchool hs1,HighSchool hs2){
        1. HS Academic Grade
            a. grade2 = convertGrade(hs1 grade)
            b. grade1 = convertGrade(hs2 grade)
            c. WeightedHS = calculate_weighted_score(grade1- grade2, TotalAcademicGrade,
               AcademicWeight)
        2. SAT/ACT
            a. WeightedSAT = calculate_weighted_score(Difference b/w SAT Scores,
               TotalSATScore, SATWeight)
            b. WeightedACT = calculate_weighted_score(Difference b/w ACT Scores,
               TotalACTScore, ACTWeight)
        3. Graduation Rate
            a. WeightedGradRate = calculate_weighted_score(Difference b/w Graduation Rates,
               TotalGraduationRate , GraduationRateWeight)
        4. State Tests
                i. WeighteStateMath = calculate_weighted_score(Difference b/w State Math
                   Tests, TotalStateTests, StateTestWeight)
               ii. WeigthedStateReading = calculate_weighted_score(Difference b/w State
                   Reading Tests, TotalStateTests, StateTestWeight)
        5. similarity = Overall Grade Similarity + SAT/ACT Similarity + HS Similarity + AP Score
           Similarity

    return similarity

}
```

Calculate Weighted Similarity Score

Description: Calculates a weighted similarity score of a given criteria

Parameters:

difference : difference between two values in criteria

total : total value of criteria

weight : percentage out of 1.0, that you want the criteria to be worth

Returns: similarity - a weighted percentage of how similar two values are

```
function calculate_weighted_score(double difference, double total, double weight)
{
        To calculate each weight, return the value of the equation:
        $\left(1 - \left|\frac{Difference\ In\ Criteria}{Total\ Criteria\ Value}\right|\right) * weight$
}
```

Compute College Recommendation Score

Description: Provides a score out of 10.0 for each entered colleges, based on several pieces of information

Parameters: college_collection - a pre-filtered list of colleges with no recommendation score

userprofile - the profile of student to compute college recommendation for

Returns: recommended_Colleges_List - a sorted list of colleges based on recommendation score out of 10.0

Information on Criteria - Used in function calls (colored)		
Criteria	Total	Weight (%)
GPA	1.5[1]	30%
SAT/ACT Score	ACT = 36 , SAT = 1600	30%
Similar Student Acceptances	Total number of students who were accepted	40%

```
function compute_College_Recommendation_Score(College[] college_collection , Profile userprofile){
    for(college in college_collection){
        1. Check if the college has one of your listed majors. If not, assign score 1.0 to the college and
           continue to next one
        2. If it does have the listed major do these calculations to get a accurate score:
            a. GPA - If the difference between the two is greater than TotalWeight, the similarity is
               just 0.
                i. gpaScore = calculate_weighted_score(GPA Average - student GPA, GPATotal,
                   GPAWeight)
            b. SAT - ACT ranks
                i. if(Student took ACT)
                    1. WeightedACT = calculate_weighted_score(Difference b/w ACT
                       Scores of college and users, ACTTotal , ACTWeight)
                ii. if(Student took SAT)
                    1. WeightedSATEBRW = calculate_weighted_score(Difference b/w
                       SATEBRW Scores of college and user, SATTotal/2, SATWeight/2)
                    2. WeightedSATMATH = calculate_weighted_score(Difference b/w
                       SATMATHScores of college and users, SATTotal/2, SATWeight/2)
            c. Check how many similar students were accepted to the school
                i. Loop through all acceptance decisions for the college
                    1. Compare each student to user using determine_Similar_Profiles()
                    2. numSimStudents = number of students with 75% or more similar
                       profile to the user
                    3. SimilarStudentsScore= numSimStudents/ total * SimilarStudentWeight
            d. Final Computed Score = (GPA Rank + SAT/ACT Rank+ Similar Student Score) * 10.0
                i. Multiply by 10.0 above to get the score out of 10.0
                ii. Assign score to college and move onto next one
    }
    sort and return recommended_Colleges_List
    }
```

1 - Use 1.5 for gpa because it is an inflated stat. Since almost all GPAs are greater than 2.5, out demonator becomes 4-2.5 = **1.5**

Determine similar profiles

Description: Provides a percentage of how similar a student profile is to another.

Parameters: profile1 - student we are going to compare

profile2 - student we are comparing to

Returns: similarity - a weighted percentage of how similar two profiles are

Information on Criteria - Used in function calls (colored)		
Criteria	Total	Weight (%)
GPA	1.5[1]	35%
SAT/ACT Score	ACT = 36 , SAT = 1600	35%
Overall High School Grade	4.0	15%
AP Passing Rate	100	15%

```
function determine_Similar_Profile(Profile profile1,Profile profile2){
    1. GPA - If the difference between the two is greater than 2.5, the similarity is just 0.
        a. WeightedGPA = calculate_weighted_score(Difference b/w GPA, GPATotal, GPAWeight)
    2. AP Scores
        a. WeightedAP=calculate_weighted_score(Difference b/w AP%, APTotal, APWeight)
    3. High School
        a. WeightedHS = determine_similar_hs_amount(p1.highschool, p2.highschool) * HSWeight
    4. SAT - ACT ranks
        a. Both student took ACT
            i. WeightedACT = calculate_weighted_score(Difference b/w ACT Scores, APTotal,
               APWeight)
        b. Both students took SAT
            i. WeightedSATEBRW = calculate_weighted_score(Difference b/w SATEBRW Scores,
               SATTotal/2, SATWeight/2)
            ii. WeightedSATMATH = calculate_weighted_score(Difference b/w SATMATHScores,
                SATTotal/2, SATWeight/2)
        c. Both Students took Neither Test
            i. Both students are equal in terms of testing so its 100% similarity for that weight:
            ii. weightedTestRank = 0.35
        d. Else (mismatching tests)
            i. Convert the test score of student 1 of respective test to match student 2's taken test
               using the method converSATACTScore(), then calculate_weighted_score.

    similarity = GPA Similarity + SAT/ACT Similarity + HS Similarity + AP Score Similarity

    return similarity
}
```

1 - Use 1.5 for gpa because it is an inflated stat. Since almost all GPAs are greater than 2.5, out demonator becomes 4-2.5 = **1.5**

Convert Letter Grade

Description: Method is a helper function to convert Letter grades to numbers on the 4.0 scale

Parameters: letter_grade: A letter grade ENUM

Returns: score - a 4.0 scaled score corresponding to the letter grade

```
function convertLetterGrade(Grade g){
        Return corresponding value
        A+ = 4.0
        A = 3.6
        A- = 3.3
        B+ = 3.0
        B = 2.7
        B- = 2.4
        C+ = 2.1
        C = 1.8
        C- = 1.5
        D+ = 1.2
        D = 0.9
        D- = 0.6
        F = 0.3
}
```

1 - Use 1.5 for gpa because it is an inflated stat. Since almost all GPAs are greater than 2.5, out demonator becomes 4-2.5 = **1.5**

Convert SAT/ACT Scores

Description: Method is a helper function to convert SAT and ACT scores when needed

Parameters: score : Score of ACT/SAT

type: Type of test (ether SAT/ACT)

Returns: computed_score - a score corresponding to test depending on parameter

```
function convertSATACTScore(score, type){
    Use this scale to convert SAT/ACT Scores, and return the value depending on the 2 parameters
```

SAT Range	ACT Score
1570-1600	36
1530-1560	35
1490-1520	34
1450-1480	33
1420-1440	32
1390-1410	31
1360-1380	30
1330-1350	29
1300-1320	28
1260-1290	27
1230-1250	26
1200-1220	25
1160-1190	24
1130-1150	23
1100-1120	22
1060-1090	21
1030-1050	20
990-1020	19
960-980	18
920-950	17
880-910	16
830-870	15
780-820	14
730-770	13
690-720	12
650-680	11
620-640	10
590-610	9

```
}
```

1 - Use 1.5 for gpa because it is an inflated stat. Since almost all GPAs are greater than 2.5, out demonator becomes 4-2.5 = **1.5**

Detect Questionable Acceptance Decisions

Description: After a user enter an acceptance decision, this function is called to detect whether the acceptance decisions seemed correct or incorrect

Parameters: acceptanceDecision-an acceptance decision which contains a college

userprofile - the profile of student to check

Returns: marked_Acceptance_Decision - an acceptance decision marked if it is questionable or not

Information on Criteria - Used in function calls (colored)		
Criteria	Total	Weight (%)
GPA	1.5[1]	30%
SAT/ACT Score	ACT = 36 , SAT = 1600	30%
User Better Than Student Acceptances	Total number of students who were accepted	40%

```
function detect_Questionable_Decision(AcceptanceDecsion decision, Profile userprofile){
  1. Check if user's grades in each criteria is statistically less than the college averages
     a. GPA - If the difference between the two is greater than 1.5, the similarity is just 0.
        i. gpaScore = calculate_acceptance_score(Student GPA, College GPA Avg, GPATotal, GPAWeight)
     b. SAT - ACT ranks
        i. if(Student took ACT)
           1. ACTScore = calculate_acceptance_score(Student ACT Score, College ACTAvg, ACTTotal , ACTWeight)
        ii. if(Student took SAT)
           1. SATEBRWScore = calculate_acceptance_score(Student SATEBRW Score, College SATEBRW Avg, SATTotal/2, SATWeight/2)
           2. WeightedSATMATH = calculate_acceptance_score(Student SATMATHScore, College SATMATHAvg, SATTotal/2, SATWeight/2)
     c. Check how many worse students were accepted to the school
        i. Loop through all acceptance decisions for the college
           1. Compare each student to user and determine if the user has better academic achievements than the other students (numBetterStudents).
              a. Using calculate_isBetter()
           2. SimilarStudentsScore= numBetterStudents/ total * SimilarStudentWeight
     d. Difficulty Weighted Computed Score
        i. Accounting for difficulty of entry of the college we use weights assigned by convertDifficulty()
        ii. score = (GPA Rank + SAT/ACT Rank + Similar Student Score)
        iii. weighted_score = score * convertDifficulty(College Difficulty)
     e. If weighted_score < 0.65, we consider that acceptance decision questionable
  2. Finally, assign a mark and return the acceptance decision if it is questionable or not
}
```

1 - Use 1.5 for gpa because it is an inflated stat. Since almost all GPAs are greater than 2.5, out demonator becomes 4-2.5 = **1.5**

}

Calculate Weighted Questionable Acceptance Score

Description: Calculates a weighted questionable acceptance score of a given criteria.

Parameters: student_score: score of student for criteria

college_avg: average score of college for criteria

total : total value of criteria

weight : percentage out of 1.0, that you want the criteria to be worth

Returns: score- a weighted percentage of how negatively far the student value is from the college average.

```
function calculate_acceptance_score(double student_score, double college_avg, double total, double weight)
{
        ○ If student_score < college_avg, then the student's score is lower, and we use a weight formula:
            ■ And return : $\left(1 - \left|\frac{Difference\ In\ Criteria}{Total\ Criteria\ Value}\right|\right) * weight$
        ○ If student_score > college_avg, then the student's score is higher, and we say the decision for
          this criteria is 100% likely
            ■ So we just return weight
}
```

Calculate Student Worse than User

Description: Determines if the user has better academic records than a given student

Parameters: student: another student profile

user: user's profile

Returns: isBetter - boolean T/F , if the user has better academics than the student

```
function calculate_isBetter(Profile student, Profile user)
{
    1. Use a similar algorithm to "Determine Similar Profile" combined with "Calculate Weighted
       Questionable Acceptance Score"
    2. If user value for criteria is greater than student, then can give the entire weight value for that criteria
    3. GPA = Compare GPAs, * weight
    4. SAT = Compare SAT's,* weight
    5. HS = Compare HS, * weight
    6. AP = Compare AP%, * weight
    7. better = GPA + SAT + HS + AP
    8. Return true is weighted better value is > 0.5, false otherwise
}
```

Convert Difficulty

Description: Method is a helper function to convert difficult of school weights used to calculate acceptance decision scores.

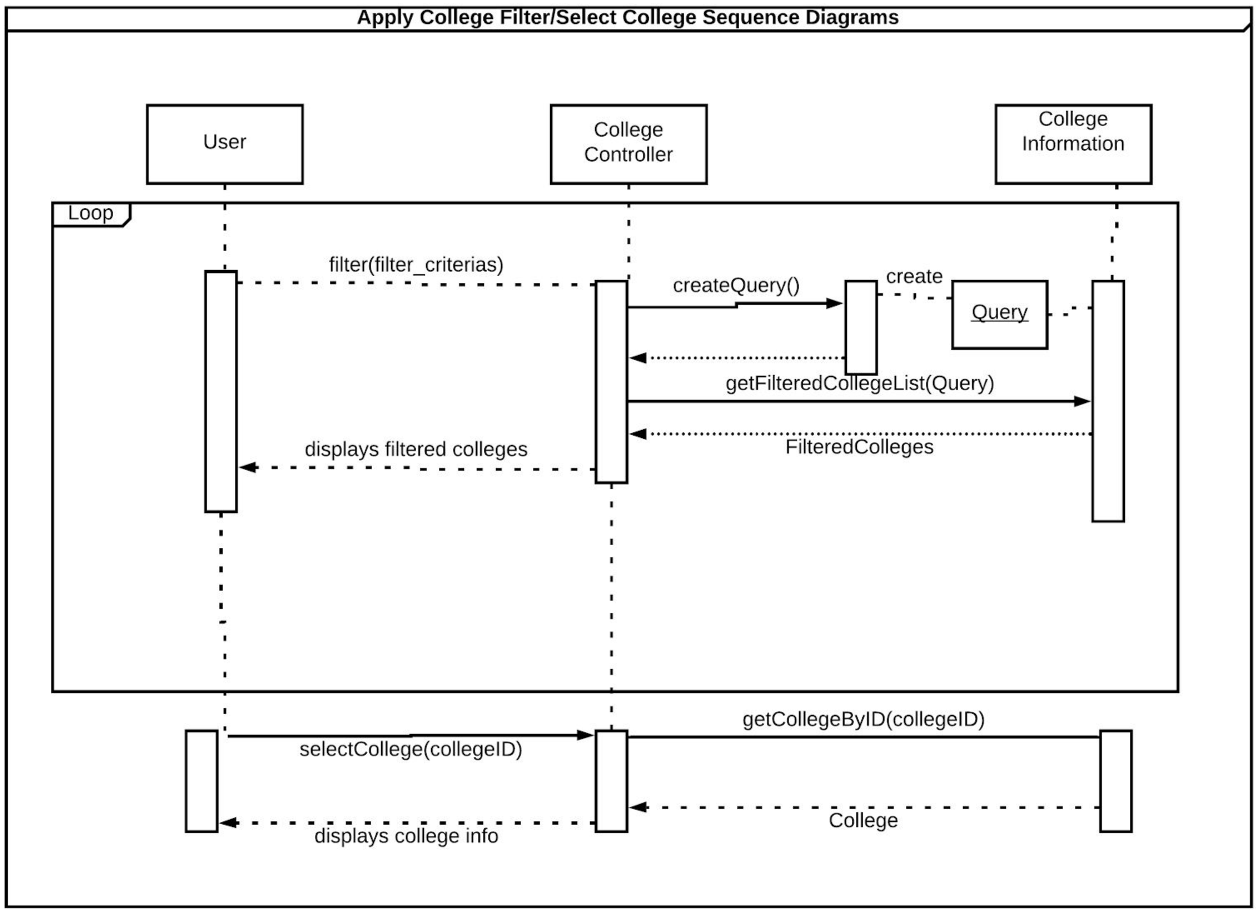
Parameters: letter_grade: A Difficulty ENUM

Returns: score - a 1.0 scaled weight corresponding to the Difficulty level

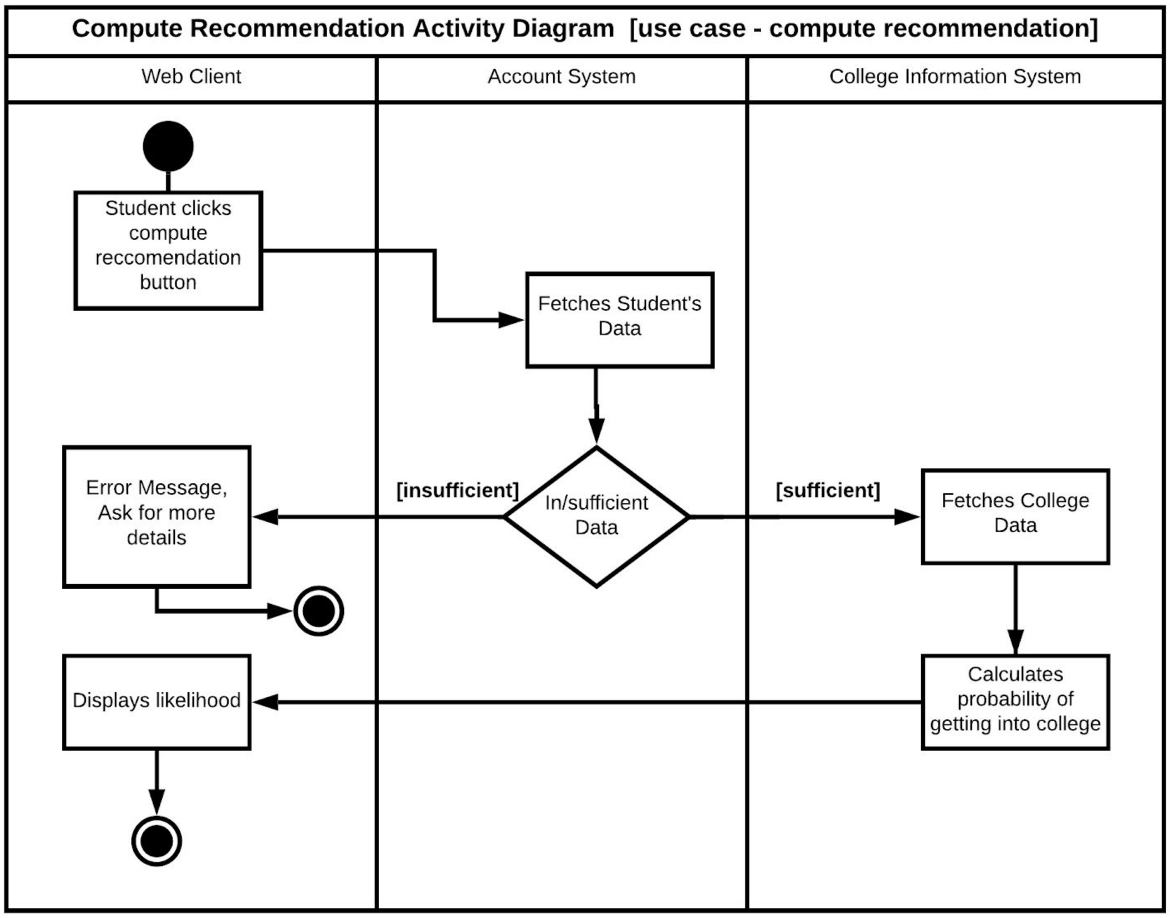
```
function convertDifficulty(Difficulty d){
        Return corresponding value:
        Very = 0.87
        Moderate = 0.9
        Minimall = 0.93
        Noncompetitive = 0.96
        Most = 1.0
}
```

Dynamic Model

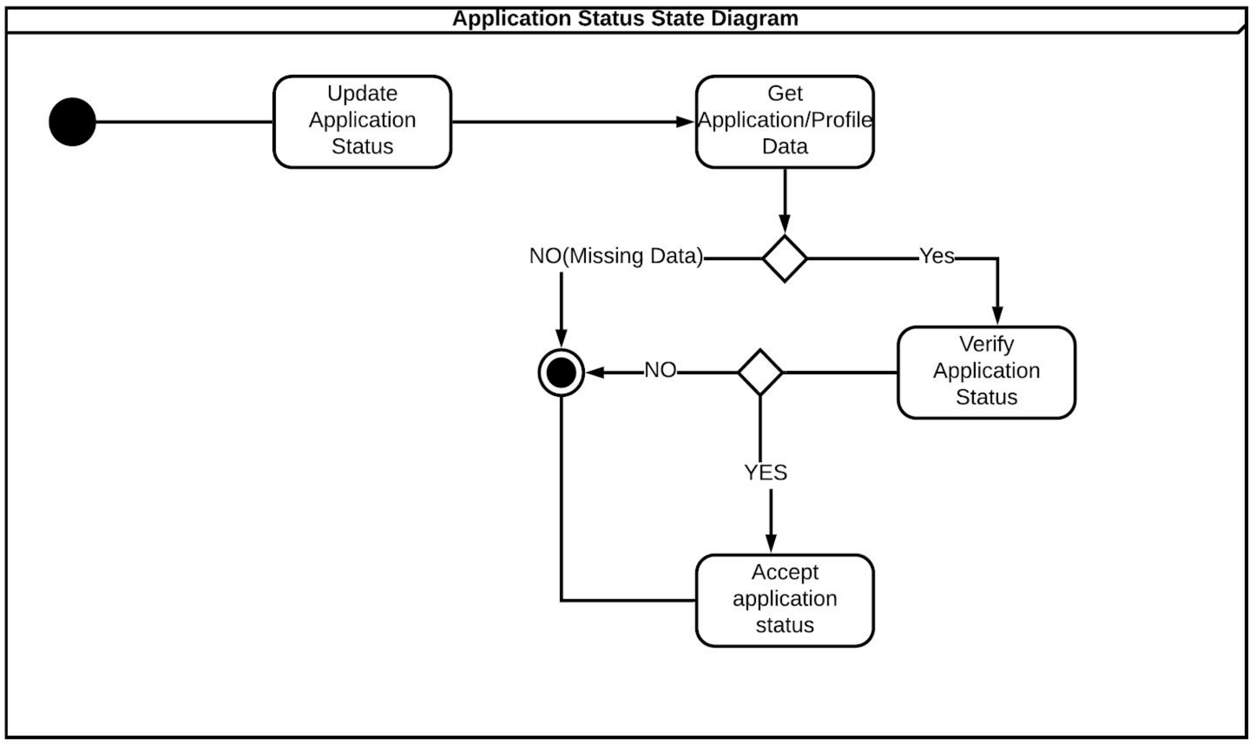
Sequence Diagram



Activity Diagram


Compute Recommendation Activity Diagram [use case - compute recommendation]
Web Client
Account System
College Information System
Student clicks compute reccomendation button
Fetches Student's Data
In/sufficient Data
[insufficient]
[sufficient]
Error Message, Ask for more details
Fetches College Data
Calculates probability of getting into college
Displays likelihood

State Diagram


Application Status State Diagram
Update Application Status
Get Application/Profile Data
NO(Missing Data)
Yes
Verify Application Status
NO
YES
Accept application status

Contributions

Kenny Liang

1. Setting up team communication system and collaborative requirement analysis document outline
2. Use cases worked on: Create Account, Login to Account
3. Edited and reviewed use cases and use case diagram
4. Worked on Component Diagram, Class Diagrams, and Languages/Technologies/Tools
5. Reviewed Algorithms and Dynamic model

Adam Tringali

1. Use cases worked on:Edit profile, Search for college, Find similar high schools
2. Edited and reviewed use cases and use case diagram
3. Wireframes worked on: Find similar HS page, Edit profile page, Search for colleges
4. Worked on Class Diagrams
5. Reviewed and edited Dynamic Model

Dylan Singh

1. Worked on the majority of the use case diagram
2. Did View Other Student Profiles use case
3. Edited and reviewed several use cases
4. Worked on Wireframes and Class Diagrams
5. Worked on all Algorithms

Nathan Melnyk

1. Uses cases worked on: Edit/Add High School, Search for College, Compute Recommendation, Apply College Filter, Select College, See Other Student Application Information, Apply Application Filter, Check Scatterplot
2. Wireframes worked on: Landing page, Create Account, Main page, Edit Profile, Search for College pages 2 and 3
3. Wireframes Edited: Search for College Page
4. Worked on dynamic model