

CSC415 Assignment 4

Brandon Hong

OpenEd: Analysis and Design

VM Server: csc415-server12.hpc.tcnj.edu

VM Username: student1

VM Project Path: /home/student1/assignments/assignment3/opened

Github: <https://github.com/bhong94/openEd>

OSS License: Apache 2.0

(+)Use Case Descriptions

Use Case: Login

Primary Actor: Tutor, Admin, Parent

Goal In Context: To let users sign-in to their accounts

Preconditions: Not currently logged in, and correct credentials have been entered.

Trigger: Login Button, accessible in several parts of the web app (i.e navbar, footer, home screen)

Scenario:

1. A user (tutor, admin or parent) navigates to the login view.
2. A user types their username and password
3. A user presses the "Login" form button

Exceptions:

1. Username is incorrect
2. Password is incorrect
3. Maximum number of tries exceeded

Priority: Moderate Priority. Implement after major functionality

=====

Use Case: Create Profile

Primary Actor: Tutor, Parent

Goal In Context: To let users create personal accounts.

Preconditions: Do not currently have a registered account and valid information has been entered.

Trigger: Register Button

Scenario:

1. A user navigates to the register view.
2. A user enters their email, username, and password.

3. User presses the “Register” form button

Exceptions:

1. Email is invalid
2. Username does not meet requirements
3. Password does not meet requirements

Priority: Moderate Priority. Implement after major functionality.

=====

Use Case: Search For Tutor

Primary Actor: Parent

Goal In Context: To allow parents/students to find compatible tutors

Preconditions: Parent is logged in to their account, and keywords/credentials have been specified.

Trigger: Search bar, “Search” button

Scenario:

1. Parent enters search constraints (i.e subject of interest, cost, Tutor name)
2. Parent clicks on “search” button

Exceptions:

1. No results found with given constraints.

Priority: High. This is one of the major functionalities of the application

=====

Use Case: Message Parent/Tutor

Primary Actor: Tutor, Parent

Goal In Context: To allow parents and tutors to communicate through the application

Preconditions: Sender of the message must be logged in to send a message and the recipient must be an existing user account. The recipient must be logged in to view the message and reply.

Trigger: “Send Message” button on profile page and “Send” button to actually send the message.

Scenario:

1. A user presses the “send message” button on another user’s page
2. A user types a message
3. A user clicks the “Send” button to send their message
4. A user clicks on “View Chat” to view existing chats

Exceptions:

1. There was an error while sending the message

Priority: Moderate/High. It is important to have this function implemented at a basic level

=====

Use Case: Update Profile

Primary Actor: Tutor, Parent

Goal In Context: Allow users to enter their profile information or update it with new information.

Preconditions: The user must have an existing account and the information entered must be valid/meet requirements.

Trigger: Automatic redirection to this use case after “Create Profile” OR an “Edit Profile” button

Scenario:

1. A new account has been successfully created OR the "Edit Profile" Button was clicked.
2. A user enters new profile information in provided fields.
3. A user clicks "Save Changes" button.

Exceptions:

1. Entered information is not valid.

Priority: High. Provides a way for users to input their account information to be saved in the database.

=====

Use Case: View Recommended Tutors

Primary Actor: Parent

Goal In Context: To allow parents to see tutors that my algorithm will recommend for them.

Preconditions: The parent has valid profile information for the algorithm to use.

Trigger: Parent clicks on "Recommended" tab in the dashboard

Scenario:

1. Parent clicks on "Recommended" tab.
2. Parent views recommended tutors and clicks on a profile.

Exceptions:

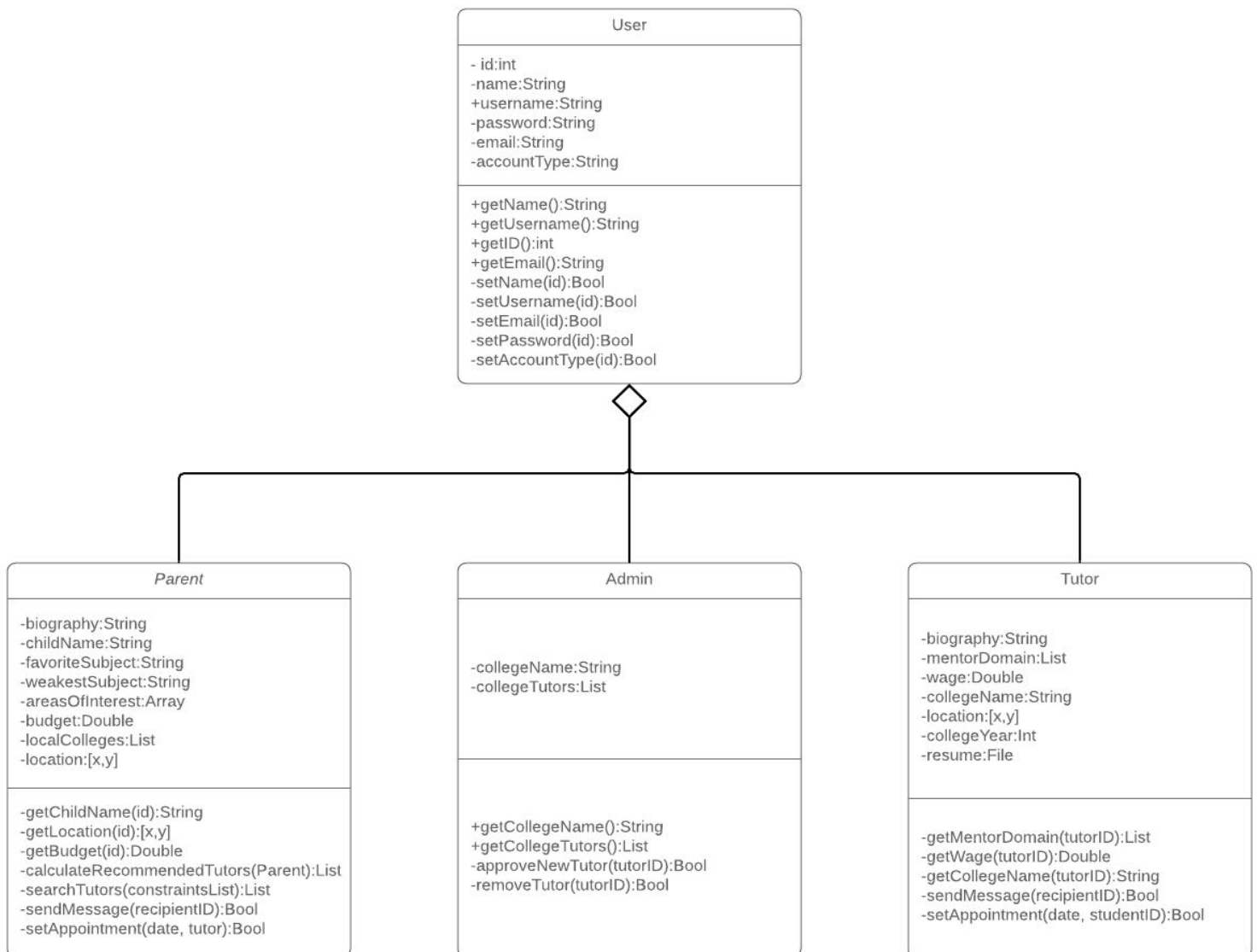
1. Vital profile information is missing that the algorithm requires

Priority: High. Main functionality/algorithm for this application

(+)Design Class Diagram

OpenEd Design Class Diagram

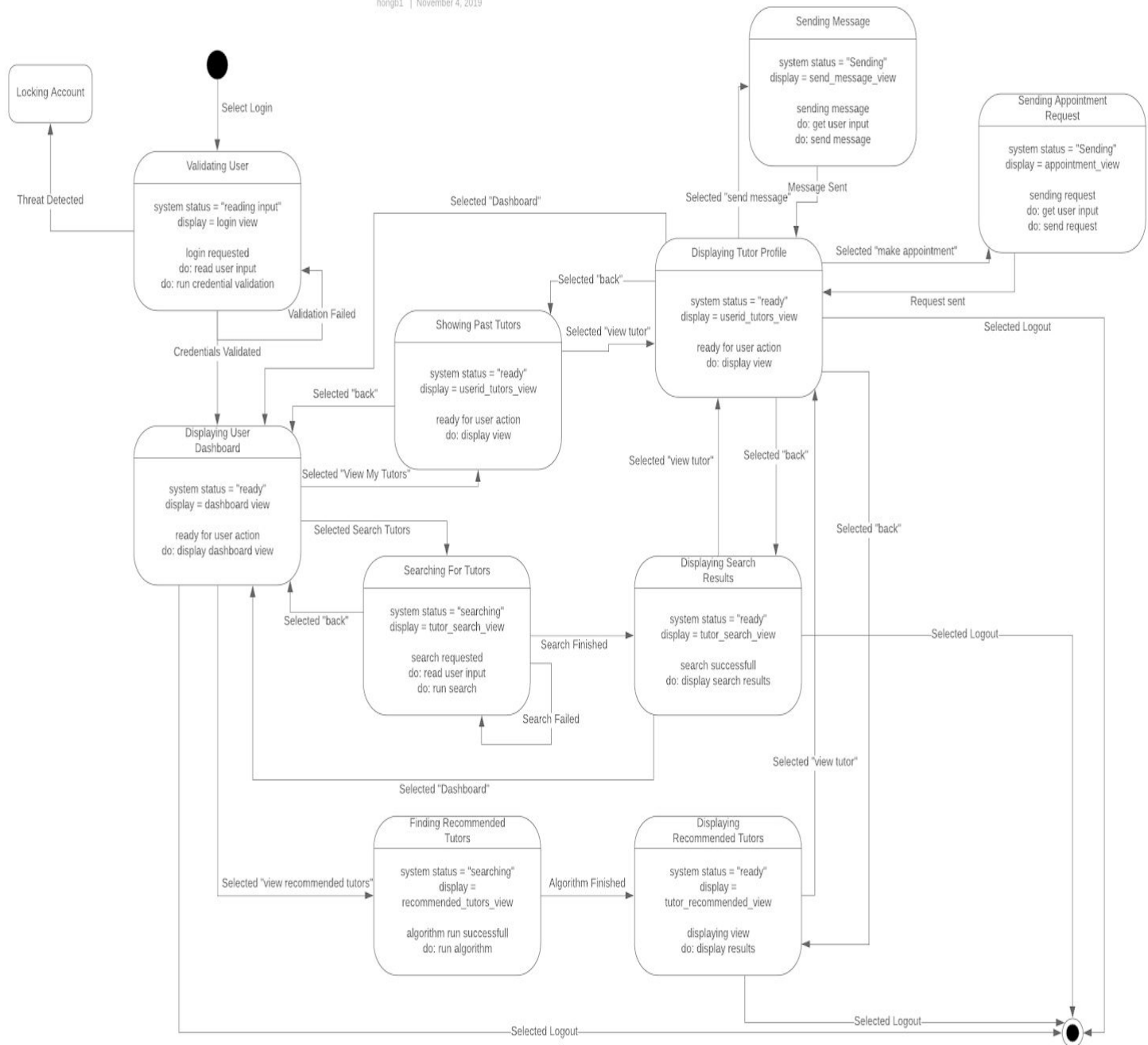
hongb1 | November 4, 2019



(+)Statechart Diagram

OpenEd State Diagram

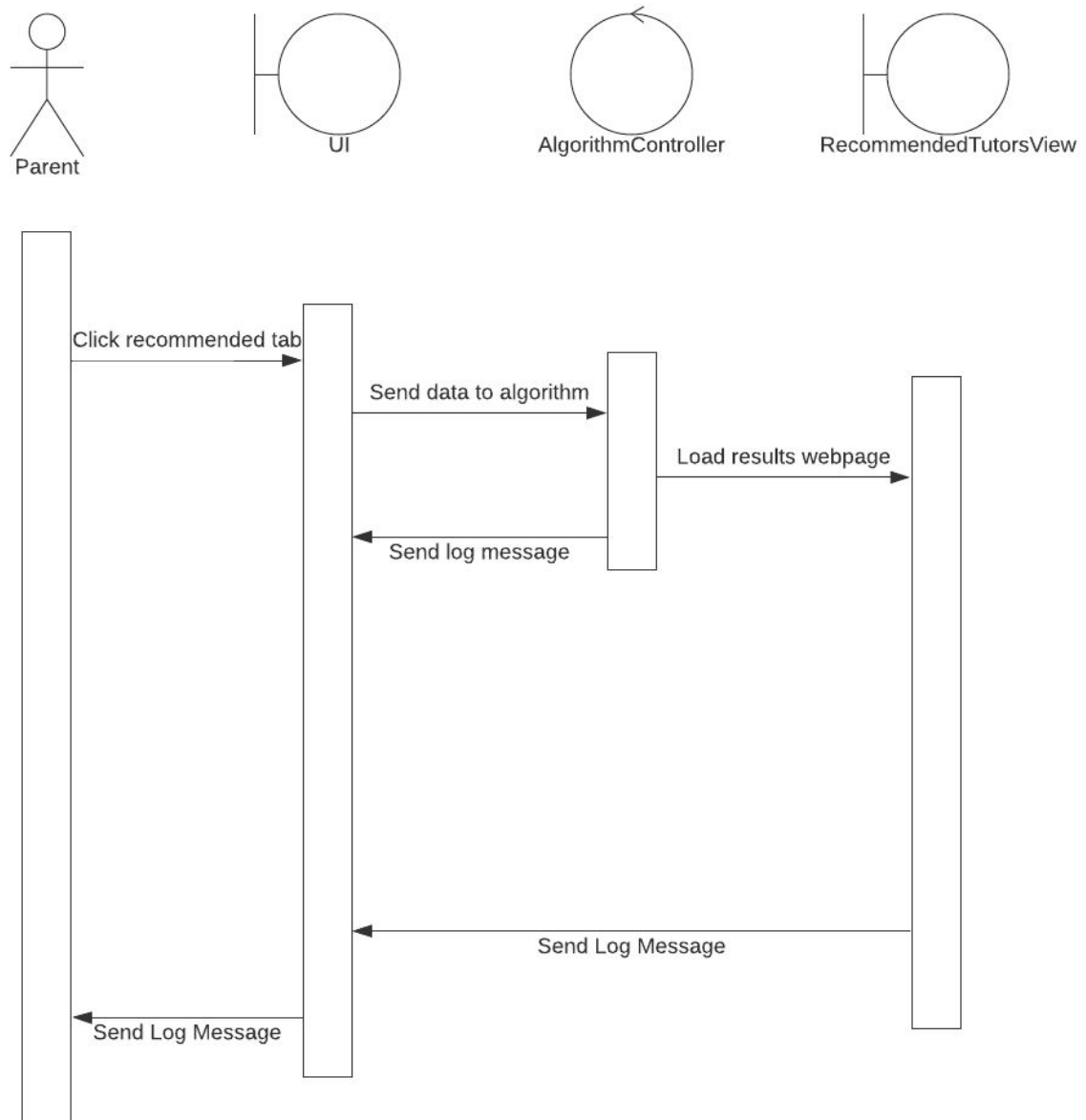
hongb1 | November 4, 2019



(+)System Sequence Diagrams

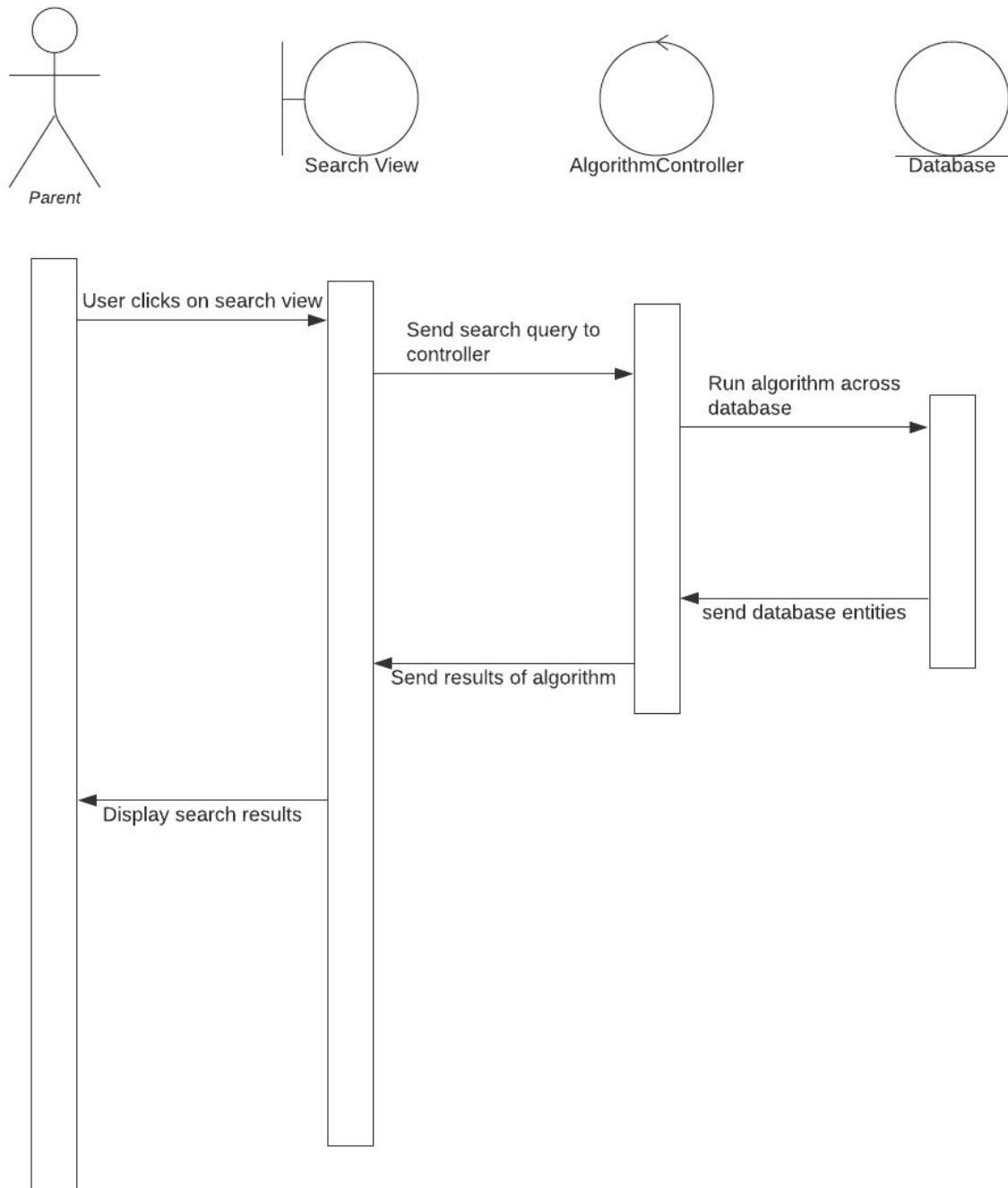
ViewRecommendedTutors Sequence Diagram

hongb1 | November 5, 2019



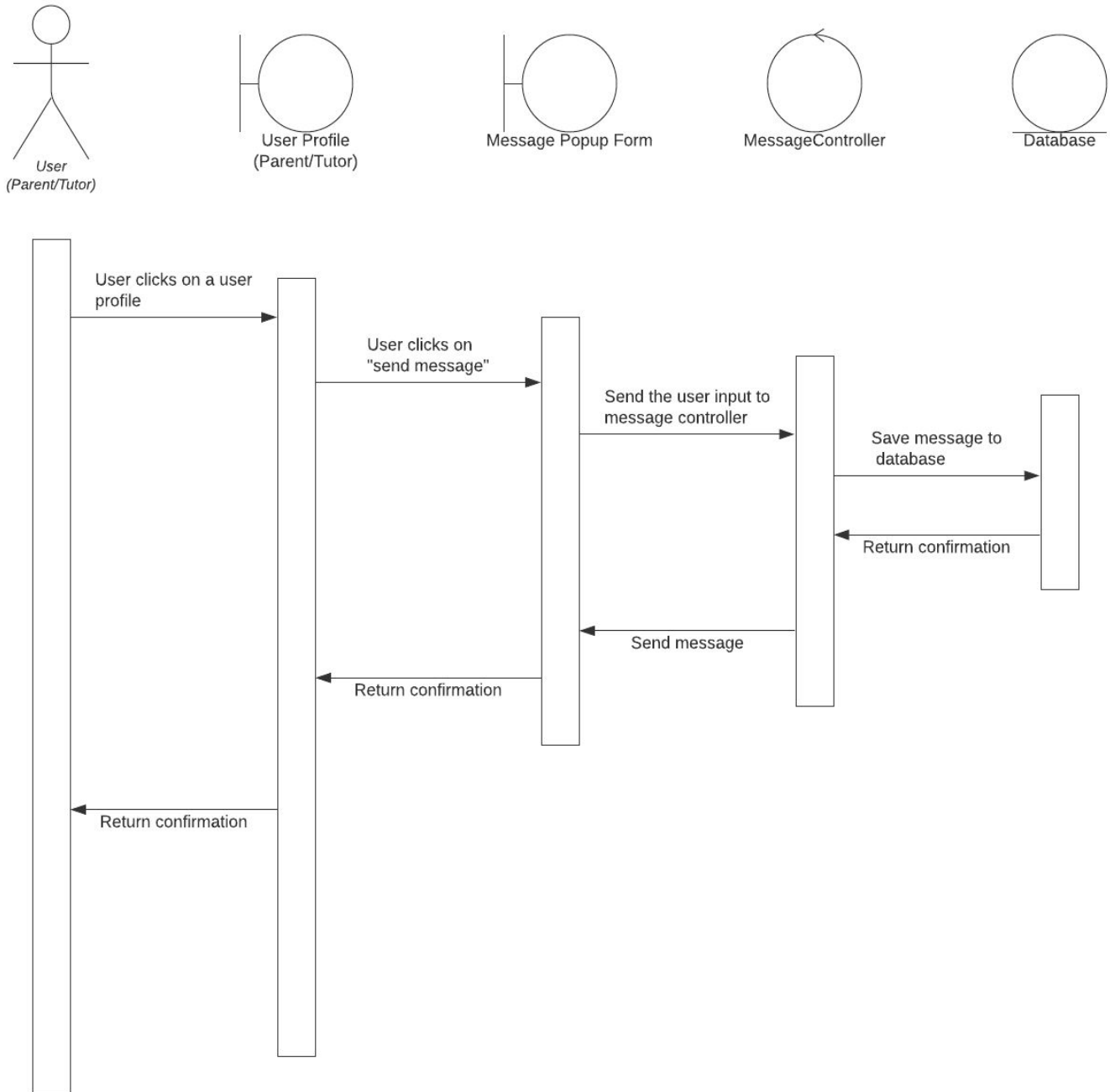
SearchTutor Sequence Diagram

hongb1 | November 5, 2019



SendMessage Sequence Diagram

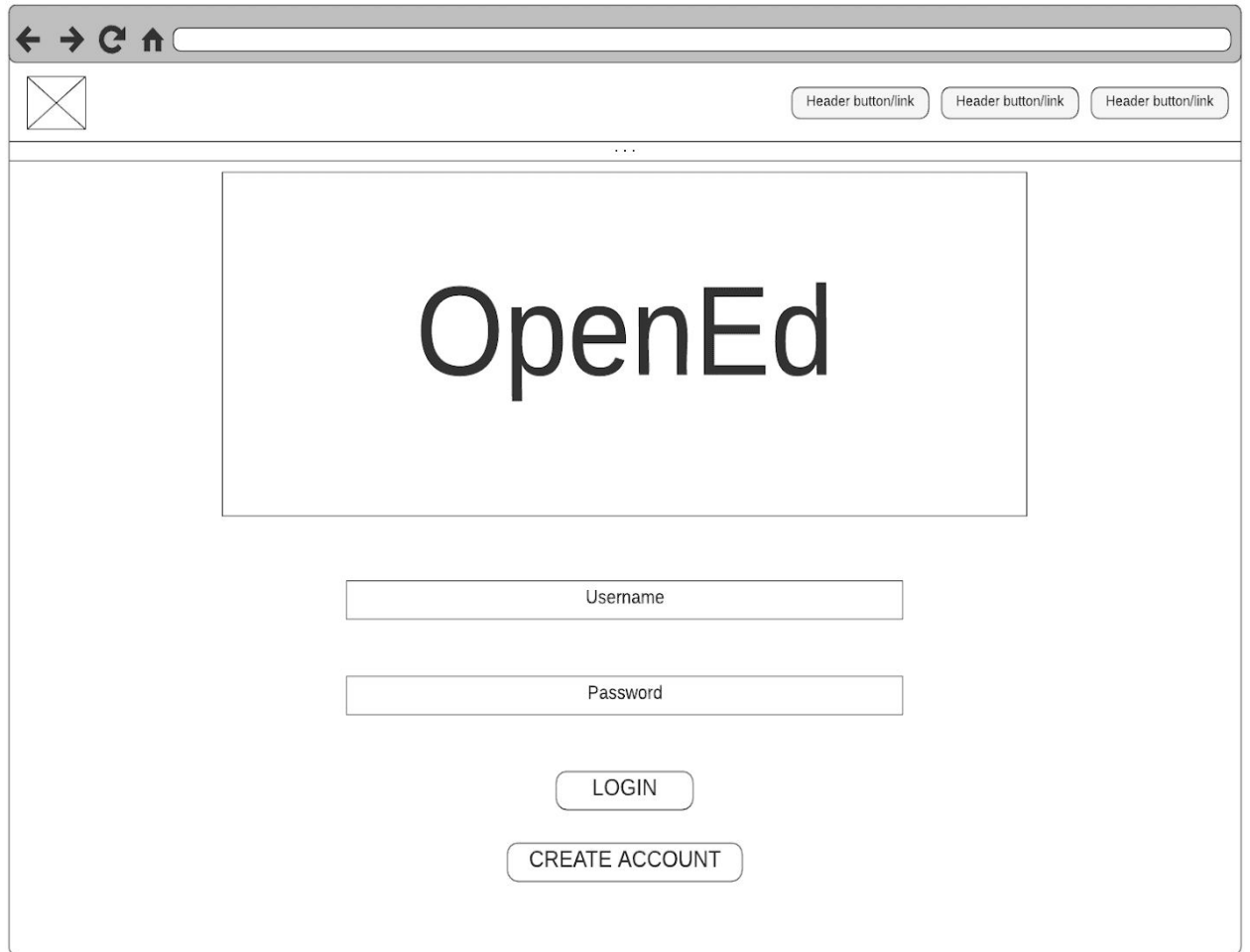
hongb1 | November 5, 2019



(+)User Interface Design

Login Page Basic Wireframe

hongb1 | November 5, 2019



A wireframe of a login page for 'OpenEd'. The page is enclosed in a browser window frame. At the top left is a placeholder for a logo (a square with an 'X'). To the right are three header buttons, each labeled 'Header button/link'. Below the header is a large rectangular box containing the text 'OpenEd' in a large, bold, sans-serif font. Below this box are two input fields: the first is labeled 'Username' and the second is labeled 'Password'. Below the input fields are two buttons: 'LOGIN' and 'CREATE ACCOUNT', stacked vertically. The entire page is centered and has a clean, minimalist design.

Header button/link Header button/link Header button/link

OpenEd

Username

Password

LOGIN

CREATE ACCOUNT

Sign-Up Page Basic Wireframe

hongb1 | November 5, 2019

←

→

↺

⬆

Header button/link

Header button/link

Header button/link

...

First Name

Last Name

Email

Password

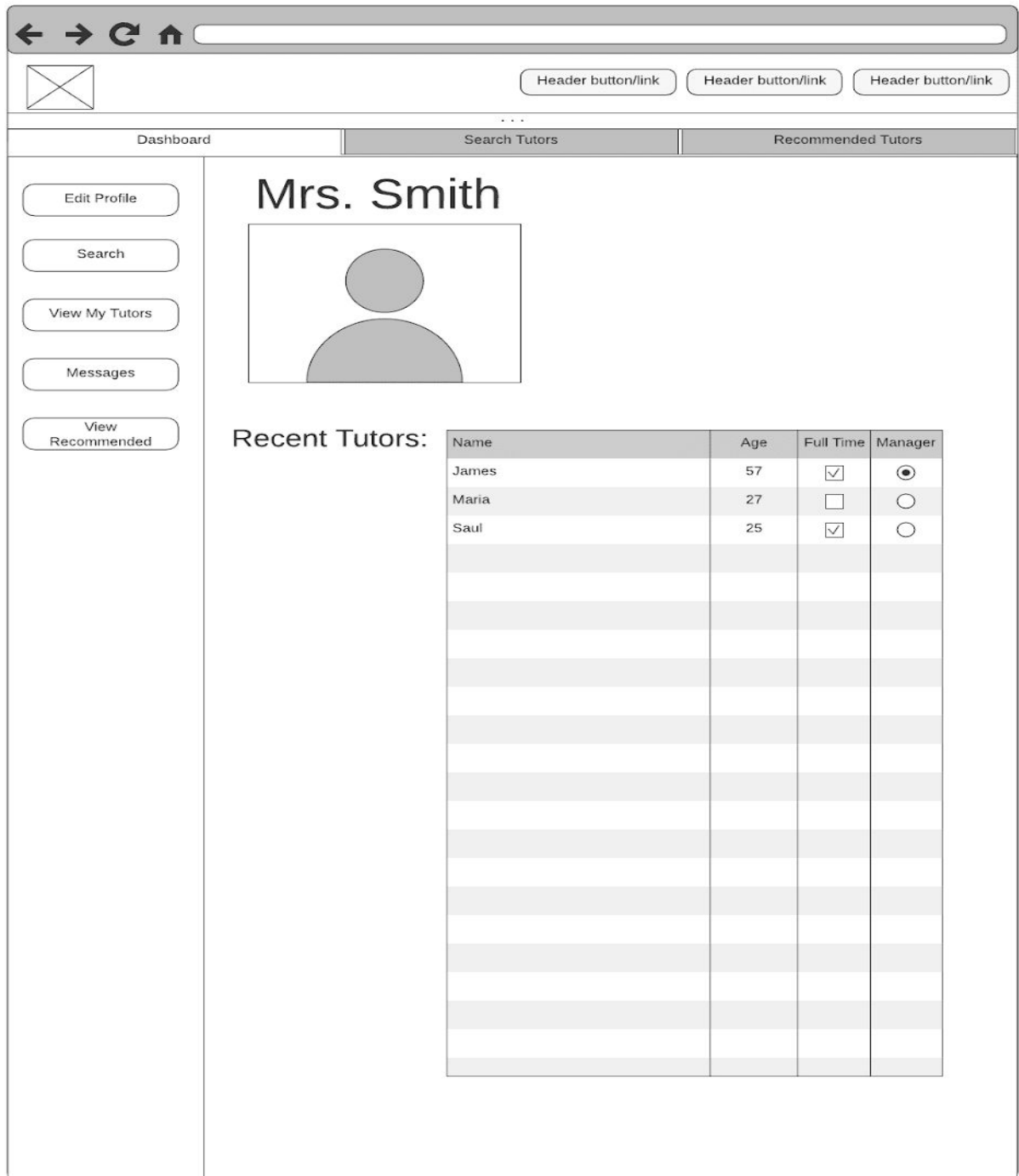
☒ Student

☐ Tutor

Register

hongb1 | November 5, 2019

hongb1 | November 5, 2019



8 Golden Rules of UI Design:

1. Strive for Consistency

My UI will have the same theme throughout the entire web app. In addition, the same navbar menu is displayed on every single page so that the same functions and features are always accessible in the same place no matter where the user is in the system.

2. Seek universal usability

The ui design is quite simple but structured. Everything in the UI is easy to see and use because there is no clutter. This will allow the UI to be usable for anyone.

3. Provide informative feedback

The UI has meaningful button names and field names. Any action the user performs will have a corresponding “result message” that provides feedback for the user. User feedback will be provided primarily through on-page messages or browser alert messages.

4. Design dialogues to yield closure

All the dialogues are short and concise. They provide the important information that the user needs to perform their desired action, and that's it. The dialogues are clear in what they are saying, and provide meaningful messages to the user.

5. Prevent errors

Errors will be handled in the system as browser alert messages in order to prevent crashing the system. I.e instead of the system crashing, the application will send a browser alert explaining exactly what went wrong.

6. Permit easy reversal of actions

All actions in my system are reversible. I will provide the necessary buttons in convenient and accessible locations that allow the user to change/fix any data that they are permitted to in the system.

7. Keep users in control

In order to keep users in control, the user interface is extremely simple and clutter-free. All of the most commonly used functions are located up in the navbar, where the user can access them in the same place anytime, anywhere on the web application.

8. Reduce short-term memory load

The navbar and simple structure of the dashboard bar provide an easy-to-remember location for all web app functionality.

(+)Project Requirements

The classes and their corresponding attributes and functions were designed with modularity and encapsulation in mind. All of the class attributes are set as private with public getter and setter methods used to modify these attributes. In addition, each function that is contained within a class is designed to perform just one task and to not utilize any other modules in order to complete that task. For example, the `calculateRecommendedTutors()` method will simply run the algorithm with a given set of parameters and return the result. No other methods will have to be accessed within this module, resulting in low coupling. All of the subtasks within this method are closely related and contribute to the one end result, resulting in high cohesion. This ensures that the source code will exhibit good modularity and be very reusable.

The algorithms to be implemented will utilize the most efficient data structures possible that is a good match for the task at hand. My analysis of all the ruby data structures lead to the choices of data structures that were specified in the OpenEd:Proposal and Specs document.

(+)Test Case Design

Functionality Tested	Inputs	Expected Output	Actual Output
User Login	Email/Username, password	User is logged into the system	
Create Profile	Name, Email, Username, Password	User account is created in database	
Update Profile	Name, Email, Username, Password, Favorite Subjects, Weakest Subject, Budget	User account details are updated in the database	
Search for Tutor	Distance, tutor name, subject of study, dollar cost per hour, etc.	Displays tutors that match the given search constraints	
View Recommended Tutors	Distance, favorite subjects, weakest	Matches student with tutors that match the	

	subjects, cost per hour	given constraints	
Send Message	Message, recipient id	Sends the message to the specified recipient	
Approve New Tutor	Tutor id	Approves a tutor, confirms creation of the tutor's account. The tutor's account is created.	

Unit Testing - For unit testing, I will run each unit (function) of code under many different circumstances to observe the outputs and identify any errors. For example, I will test the "recommended tutors" algorithm with many different inputs, missing inputs, invalid inputs, etc. I will repeat this process for each significant unit or module of code.

Integration Testing - Once all the units are tested, we can begin integrating them into the system and testing how they work with each other. For this, I will repeatedly perform a sequence of related functions under different circumstances for each iteration. For example, I will test the search function with varying parameters, then select one of the results, and finally I will try sending a test message to the tutor that I selected. I will repeat this process for all possible combinations of related units working together.

System Testing - Once I know that the units that I tested integrate well with each other, it is time to test the system as a whole. I will adapt a similar method for system testing as I used for Integration testing. Because there are so many test cases to review, a tool such as rspec might be helpful. However, I will know further into development whether or not it is necessary for me to use such a tool.

Testing Tools: As of now, I do not plan on using any special testing tools besides the built-in debugger that is included with ruby mine. This is because the functions in the application are not complex enough where I would need to look at memory usage/leaks or use some other special testing feature. Breakpoints and viewing the stack trace will be enough to thoroughly test the system.

(+) OSS Licences - Comparative Analysis + Choice

3 Popular OSS Licenses:

Apache 2.0

A permissive license that allows users to do what they would like with the software as long as the required notices are included. This license does contain a patent license from the contributors of code.

Software protected under this license **CAN**:

- Be used commercially
- Be modified
- Be distributed
- Be sublicensed
- Be used privately
- Use patent Claims
- Place warranty on the licensed software

Software protected under this license **CANNOT**:

- Hold liability
- Use trademark

Software protected under this license **MUST**:

- Include copyright
- Include license
- State significant changes made to the software
- Include "NOTICE" file (if file exists)

GNU GPL v3.0

The GNU General Public License v3.0 allows users to copy, distribute and modify the software as long as they track changes/dates in source files. Any modifications or extensions of software licensed under GPL must also be made available under GPL along with build and install instructions.

Software protected under this license **CAN**:

- Be used commercially
- Be modified
- Be distributed
- Place warranty
- Use patent claims

Software protected under this license **CANNOT**:

- Sublicense
- Hold liability

Software protected under this license **MUST**:

- Include copies of original software OR instructions to obtain original software
- State significant changes made to the software
- Disclose any source code linked with GPL 3.0
- Include license
- Include original copyright
- Include install instructions if being used commercially

MIT License

A permissive software license that allows users to do as they please with the software as long as the original copyright and license notice is included in any copy of the source code.

Software protected under this license **CAN**:

- Be used commercially
- Be modified
- Be distributed
- Be sublicensed
- Be used privately

Software protected under this license **CANNOT**:

- Hold liability

Software protected under this license **MUST**:

- Include original copyright notice
- Include license notice

My Choice: Apache 2.0

Rationale: Because the purpose of my project is to address a social issue, I would like to keep the software (and source code) extremely accessible and easy to work with for everyone. The GNU GPL provides a little more security in terms of creditship to the original author, but is also a little more restrictive when it comes to reuse of the code (no sublicensing).

MIT, on the other hand, seems unorganized and a little too unrestrictive. Apache 2.0 is a good middle ground that fits this project well. It will keep the software code completely accessible to all potential contributors, while maintaining more organization through the NOTICE file and requirement to state changes.

(+)Open Source Maintenance and Communication Processes

Issues and Bugs:

Any issues or bugs in the code must be reported to the OpenEd github repo.

If you find an issue or bug, open a new issue on the github repo:

When opening a new issue, be sure to:

- describe, in detail, the scenario that lead to the bug
- the behavior caused by the bug

Reviewing issues:

You can review open issues on the OpenEd github repo.

Open issues will be discussed on the issue page (in github).

If you find a solution to the issue, you must first propose the solution in the issue discussion.

Once the solution has been approved, you may close the issue and commit the code changes.

Contributing Code:

Anyone can contribute to this project.

In order to contribute to this project, you must create a separate branch of the project and work there.

Once your changes are ready to be committed, you may submit the code to github to be reviewed.

Once your code is reviewed and approved, the changes will be merged to the master branch.