Design and Planning Document

SNU Peer Tutoring - SWPP Group 8

1 Members

SWPP Group 8 - 이동민 오지현 김동현 김종진

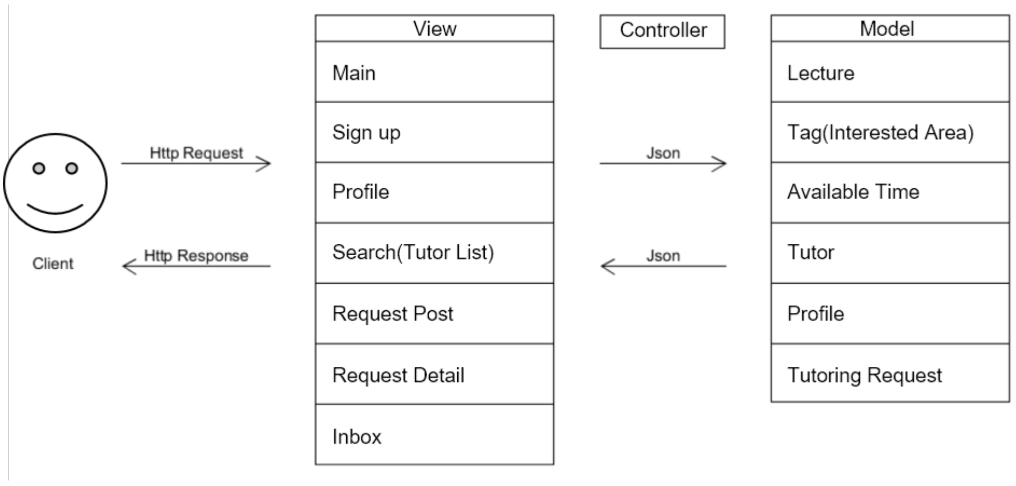
2 Revision History

The current revision is 0.9, 2019-04-22.

Initial revision (0.9): Tentative design for backend added. Basic interaction models are provided, but complete frontend design is out of scope at this time.

3 System Architecture

3.1 MVC



The project consists of seven views and six models, as shown in the diagram above.

3.2 View

The user interface model was first developed in the Requirements and Specification document. Originally, a two-way data flow between tutors and tutees was envisioned, but to reduce unnecessary implementation complexity, the architecture was simplified to a one-way data flow from tutees to tutors. As a result, the Tutee model and related views have been deleted. The following are the revised specifications.

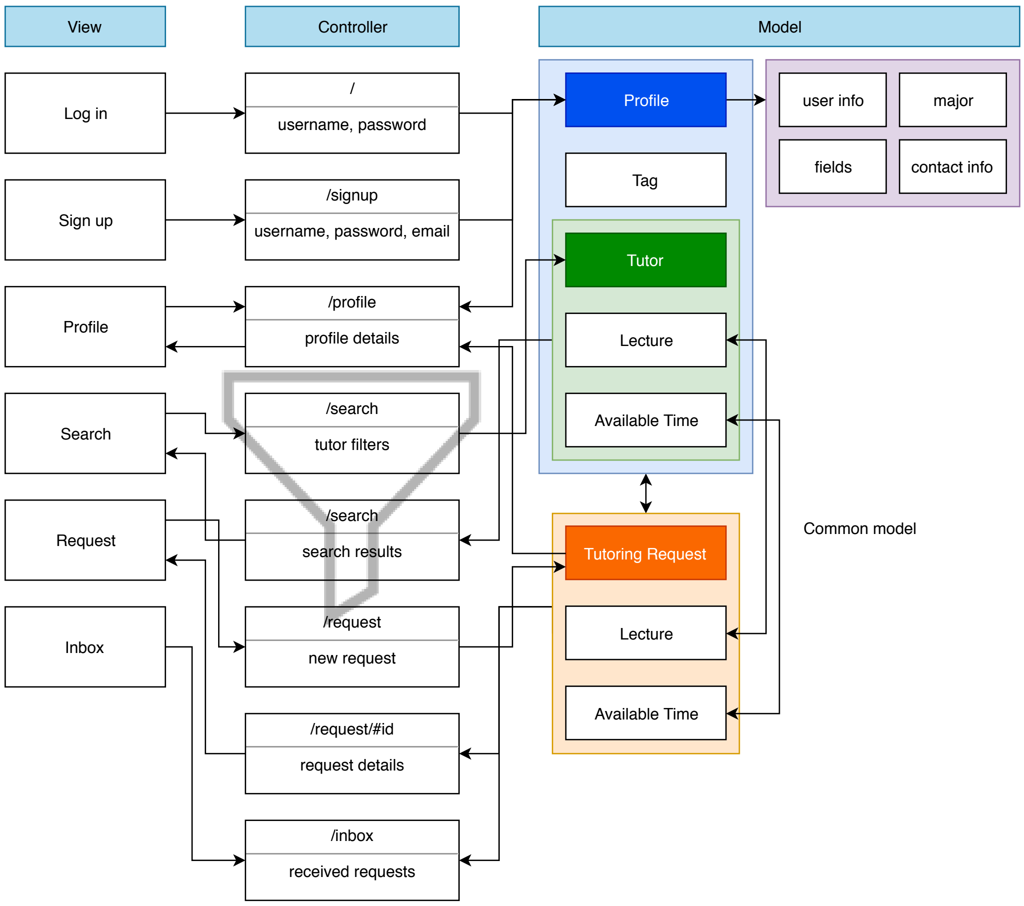
1. Sign up page (/signup)
   (a) New users can create a new account.
   (b) Takes username, password, and email as inputs.
   (c) The email may also be verified to be a valid SNU email.
   (d) CAPTCHA technology may be used to prevent automatic registrations.
2. Login/Main Page (/)
   (a) If the user is not logged in, show the log in view as necessary.
   (b) Show service information, and other website activity such as recently registered tutors.
   (c) Move to the PROFILE PAGE when the profile button is pressed.
   (d) Move to the SEARCH PAGE when the tutor search button is pressed.
   (e) Move to the INBOX PAGE when the alerts button is pressed.
3. Profile page (/profile)
   (a) Show user info, such as major, tags, and tutor information.
   (b) Allow modification of user info.
   (c) Show information about pending tutoring requests.
   (d) Move to the REQUEST DETAIL PAGE when a tutoring request is pressed.
4. Search page (/search)
   (a) Takes tags, lectures, and available times as search filters.
   (b) Show the list of tutors that match the given search query.
   (c) Move to the REQUEST PAGE when a tutor is pressed.
5. Request page (/request)
   (a) Takes tutor, lecture, and tags as inputs.

   (b) Automatically populate the input fields based on the previous search query.
   (c) Optionally take additional info and payment options as inputs.
   (d) Create a request and return to the MAIN PAGE when the request button is pressed.
   (e) Return to the SEARCH PAGE when the cancel button is pressed.

6. Request detail page (/request/:id)
   (a) Show detailed information about the tutor request.
   (b) For tutors:
      i. Remove the request and notify the tutee when the decline button is pressed.
      ii. Mark the request as accepted and notify the tutee when the accept button is pressed.
   (c) If the tutor has accepted the request, contact details of both parties are shown.
   (d) For tutees:
      i. Modify the information and notify the tutor when the edit button is pressed.
      ii. Remove the request when the cancel button is pressed.
      iii. Mark the request as completed when the complete button is pressed.
   (e) Mark the request as suspended when either party presses the suspend button.

7. Inbox page (/inbox)
   (a) Show a list of inbound notifications.
   (b) Mark notifications as read when opened.
   (c) Remove a notification when the delete button is pressed.
   (d) Move to the REQUEST DETAIL PAGE when a notification is pressed.

3.3 Controller



4 Design Details

4.1 Algorithms for Filtering Requests

The tutor matching algorithm must consider factors such as schedules, payments, experience, and more to accurately pair tutors and tutees. Factors such as schedules and experience can be automatically filtered at the server, while negotiable factors like payments and duration of tutoring can be decided with tutoring requests. After there is an agreement, the pairing is considered successful and contact information is given to both parties.

Filtering based on schedules is nontrivial, since calculating the intersection of many disjoint timeframes may scale poorly when dealing with many users. Even storing the schedule of each user is not simple. To solve this problem, the schedules of each tutor is saved in seven 64 bit unsigned integers, each corresponding to a single weekday. By dividing each day into 30-minute intervals, a person's schedule for each day can be encoded into a 48 bit unsigned integer. This allows the efficient storage and retrieval of schedule data. To calculate whether a tutor can accept a tutee, the intersection between two schedules must be calculated. This can be done by the bitwise AND operation, so that only times when both parties are available can be selected. To factor for traffic and other overhead, calculating the bitwise AND operation for multiple bit-shifted schedules can ensure that time overhead does not affect tutoring time.

By using bitwise operations as described above, it is possible to efficiently filter tutors based on their schedules. Further tuning based on the internal SQL database may be possible.

4.2 RESTful API

Model	API	GET	POST	PUT	DELETE
Profile	/signup	X	Create new profile	X	X
	/signin	X	Log in	X	X
	/signout	Log out	X	X	X
	/profile	Get profile	X	Edit profile	X
Tutor	/signup	X	Create new tutor	X	X
	/tutor	Get tutors with filter	X	Edit tutor	X
	/tutor/:id	Get specified tutor	X	X	X
Tutoring Request	/request	X	Create new tutoring request	X	X
	/request/:id	Get specified tutoring request	X	Edit tutoring request	Delete tutoring request
	/request/:profile_id	Get tutoring requests by profile id	X	X	X

5 Implementation Plan

Since there are circular dependencies between many of the required models, it is challenging to incrementally implement the project. The best approach will be to finish implementing the models and backend first, then move on to the API and frontend.

First, the backend API must be finalized by iteration 2. User registration and login functionality is also required.

By iteration 3, searching tutors based on filters must be implemented.

By iteration 4, creating and receiving requests must be finished.

By iteration 5, the inbox feature and any other pages must be completed. UX improvements can also be done if time permits.

6 Testing Plan

A simple Travis-CI based testing setup has been implemented. It utilizes django's unit tests for the backend. All tests must pass before a pull request can be merged into master.

After frontend development begins, Jest is expected to be used for frontend unit tests. It will also be integrated into Travis-CI so that code can be checked before being merged.

Currently only functional testing is performed, but coverage testing is to be implemented by iteration 2.