Waitlisters Requirements (TA office hours waitlist)

Waitlisters

Requirements and Specification Document

2023-03-08, version 1.0

2023-04-07, version 2.0

2023-04-21, version 3.0

2023-05-02, version 4.0

Project Abstract

Our web application, called Waitlisters, is designed to help college students and instructors organize office hours queues efficiently. It helps students view estimated wait times for help in the courses that they are enrolled in. Moreover, it allows instructors to manage students desiring help in the course that they are teaching. Additional features include notifying students in real-time of their status on the queue.

Document Revision History

Rev. 1.0 2023-03-08: initial version

Rev 1.1 2023-03-27: Figma diagrams updated to reflect less complex UI

Rev 1.2 2023-03-27: updated database schema and documentation for API calls

Rev 1.3 2023-03-04: remaining API endpoints added

Rev 1.4 2023-04-06: Added priorities for each use case

Rev 2.0 2023-04-07: updated formatting prior to submission.

Rev 2.1 2023-04-17: updated database schema to clarify purpose of some fields

Rev 2.2 2023-04-18: updated joinExistingCourseAsStudent endpoint spec

Rev 2.3 2023-04-20: updated instructor view figma diagram.

Rev 3.0 2023-04-20: updated formatting prior to submission.

Rev 3.1 2023-04-24: added GetCourseJoinCodes API endpoint

Rev 3.2 2023-04-25: updated CreateCourse API endpoint

Rev 4.0 2023-05-02: updated UI mockups to current screenshots of the application

Customer

Our application is designed for college students and college instructors (usually TAs) who want to attend and manage teaching assistant office hours. A problem that students often run into is long waiting times to get help from experienced teaching assistants. Our application allows students to plan out their day in advance and avoid unexpected queues when visiting office hours. Currently, the TAs for ECE 552, a class one of the group members is enrolled in, uses a whiteboard for managing office hours. They could benefit from this application.

User Requirements

This web application provides the ability to create a profile with a wisc.edu email address. This profile can then be used to create, join, and enter courses as an instructor, and/or join and enter courses as a user. Once the user has entered a course as an instructor, they can manage the queue and remove students from the top of the list.

1. Register
    a. Enter an email and password to create a new account. It must be a @wisc.edu email.
2. Login
    a. Box to enter username and password to log in
3. Home
    a. First group of options: Instructor options:
        i. Create a course as an instructor. You will be automatically added as an instructor. You will receive both an instructor code and a student code.
        ii. Add an existing course as an instructor. You'll need to enter an instructor code for this. This is for courses where there is more than one instructor.
        iii. List of all courses where you are an instructor. You can click on one of the courses to enter it via instructor view.
    b. Second group of options: Student options:
        i. Add course as a student. You'll need to enter a student code for this option
        ii. List of all courses where you are a student. You can click on one of the courses to enter it via student view.
4. Instructor View
    a. Has information about the building and room they are in. This information can be edited.
    b. Take top person out of the queue and move to "currently helping"
    c. Combine people with similar questions into one help group (based on the question that the top student had). Combine is only possible if the student selected you as an instructor.
    d. Remove no-show/late students from the queue manually and notify that they have been removed via email.

5. Student View
    a. Checkin
        i. Display wait time for each of the instructors.
        ii. Explain details of the help needed
        iii. Specify amount of estimated time required for each instructor: 5-20 min (in increments of 5)
    b. Show status of queue including remaining wait time
    c. Notify the students via email/ pop-up when help is available.
        i. Notification has building and room information
    d. Leave the queue if help is no longer needed.

Use Cases

Use cases that support the user requirements in the previous section. Every major scenario should be represented by a use case, and every use case should say something not already illustrated by the other use cases. Diagrams (such as sequence charts) are encouraged. Ask the customer what are the most important use cases to implement by the deadline. You can have a total ordering, or mark use cases with “must have,” “useful,” or “optional.” For each use case you may list one or more concrete acceptance tests (concrete scenarios that the customer will try to see if the use case is implemented)

Name	**Register**	**Login**	**Home**
Importance	Must-Have	Must-Have	Must-Have
Actor	Any first-time user of the site	Any user of the site	Any authenticated user of the site
Trigger	Click on the register link from the login page	The user navigates to the website. This is the landing page.	The user has successfully logged into the application

Events	• User navigates to the website and does not have an account
	• The user will be able to sign up with a wisc.edu email address and will create a password.
	• The user supplies a first and last name
		• User is provided with a box to enter their wisc.edu email and a box to enter their password.
		• There is also a submit button, and a link for register
			• Instructor options:
			  ○ Add course as an instructor
			  ○ Create course as an instructor
			  ○ Enter an enrolled course as instructor
			• Student options:
			  ○ Add course as student
			  ○ Enter an enrolled course as student

Exit Condition	The user has supplied their wisc.edu email and password and have created their account	The user has supplied their wisc.edu email and password and have logged into their account	The user selects one of the options available on the screen.
Post-Conditions	A confirmation screen/ notification saying that the account has been successfully created. An option to log in is then provided.	The user is redirected to the home page if the account is found, or otherwise, an error message is displayed for an invalid email/password.	The user is redirected to the appropriate page based on the options above.
Acceptance Test	The user is able to successfully log into their newly created account	The user is able to access the home page of the application	The user is on the correct page that they selected.

Name	**Add course as instructor**	**Add course as student**	**Create course as instructor**
Importance	Useful	Must-Have	Must-Have
Actor	Any authenticated user of the application	Any authenticated user of the application	Any authenticated user of the application
Trigger	User chooses “Add course as instructor” from homepage	User chooses “Add course as student” from homepage	User chooses “Create course as instructor” from homepage
Events	• Text entry box for user to supply instructor join code • Text box entry for your office location • Button to submit	• Text entry box for user to supply student join code • Button to submit	• Text entry box for name of the course • Button to submit
Exit Condition	A valid instructor join code is provided. An email is sent to the user that they have successfully joined a course as an instructor. Otherwise, an error indicating that an invalid code was entered is provided.	A valid student join code is provided. An email is sent to the user that they have successfully joined a course as a student. Otherwise, an error indicating that an invalid code was entered is provided.	The application sends an email to the user who created the course a summary of the newly created course with the unique instructor and student join code for this course.
Post-Conditions	The user is redirected to the homepage.	The user is redirected to the homepage.	The user is redirected to the homepage.
Acceptance Test	The course has been added under an option to enter as instructor	The course has been added under an option to enter as student	The course has been added under an option to enter as instructor

Name	**Instructor view:**	**Instructor view: Take next**	**Instructor view:**	**Instructor view: Combine students**

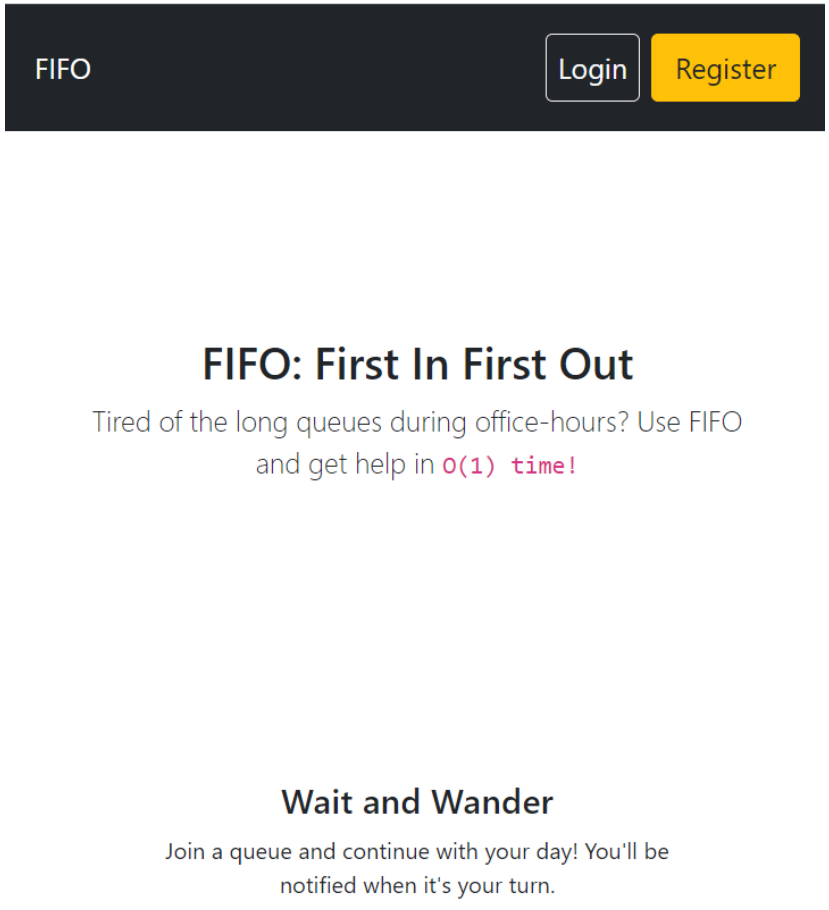
	Edit room info	**person off of queue**	**Complete Appointment**	**with similar questions**
Importance	Useful	Must-Have	Must-Have	Optional
Actor	Users as Instructors	Users as Instructors	Users as Instructors	Users as Instructors
Trigger	User clicks on “edit room info” link	User clicks on “help next student” button	User clicks on “Complete appointment” button.	User clicks on “Group Students” button
Events	• Text box entry for your office location • Button to submit	• Button for help next student is pressed • the next student/group of students who have selected you as a valid TA will be moved off of the queue.	• Button for “Complete appointment” is pressed	• Checkboxes appear to the left of the queue of students • The user (instructor) selects which students they would like to group.
Exit Condition	String is valid (more than one character) when submit button is pressed.	There must be a student/group of students to remove from the queue.	The instructor must be helping a student.	The instructor selects a “complete group” option.
Post-Conditions	The user returns to the instructor view page.	The next student (or group of students) is moved to the currently helping slot on the instructor view page. The student receives an email and pop up notification that help is available and reminds them of the instructor’s name and office location.	The student that is currently being helped is removed from the “currently helping” slot for the instructor	The students are grouped into one group in the queue. The place in the queue will be equal to the soonest possible for all combined students.
Acceptance Test	The instructor office location is updated	The student/group of students helped is no longer in the queue	The student/group of students helped is no longer in the queue and is no longer in the getting helped slot.	The group of students now appears in the queue.

Name	**Instructor view: Remove No Show student**	**Student view: Check in**	**Student view: Leave queue**
Importance	Must-Have	Must-Have	Useful
Actor	Users as Instructors	Users as Students	Users as Students
Trigger	User clicks on “Remove no show student” button	User clicks on “Check in” button	User clicks on “Leave queue” button
Events	• The student/group of students that you are currently helping will be removed from this instructor's “currently helping” slot and will be notified via email that they must reschedule.	• The student chooses which instructors they are willing to meet with (checkboxes for available instructors), explains what they need help with (via paragraph textbox), and chooses an estimate of how long it will take (drop-down options of 1,2,3,5,10,15,20).	• The student chooses to leave the queue.
Exit Condition	N/A	User presses “submit”	User presses “Leave Queue”
Post-Conditions	The students are removed from the instructor's “currently helping” slot.	The student is added to the queue. The student receives an email confirmation.	The student is cleared from the queue.
Acceptance Test	The student/group of students is no longer in the queue.	The student is added to the queue.	The student is no longer in the queue.

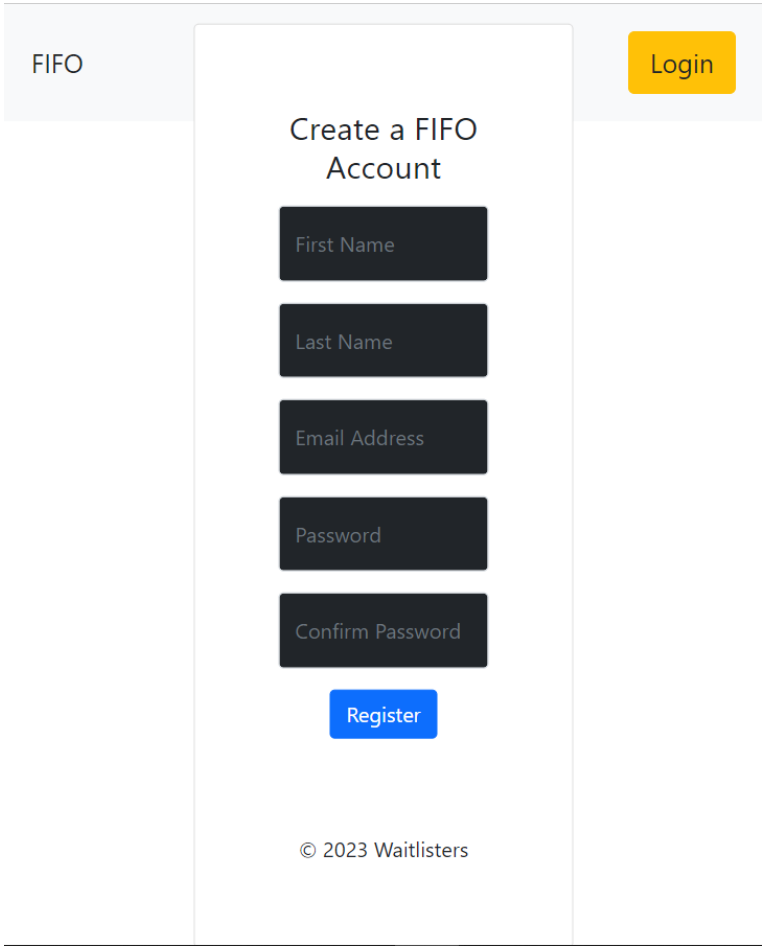
User Interface Requirements

Below are user interface diagrams (FIGMA) based on user use cases and user requirements. They are basic user interfaces to get fundamental functionality working. Crucially, these interface diagrams showcase how a user might create an account, sign in, add and create new queues and courses, and manage the respective queues that are associated with their account.

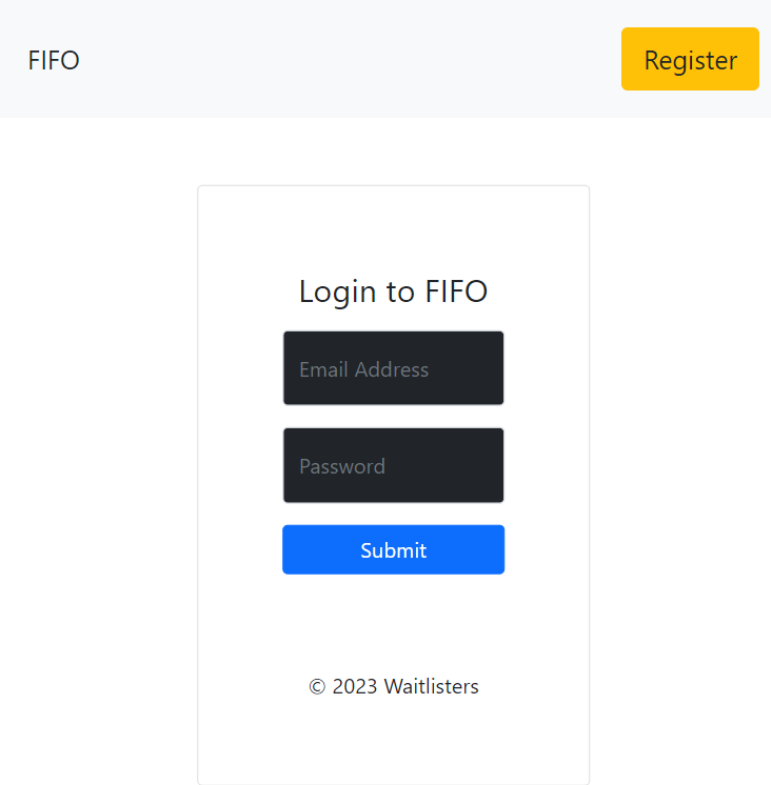
1. Home Page: Users will either sign in or register for an account on our application. It is extremely minimal and intuitive to use as there are two large and clear buttons dictating the purpose of this page - namely, the sign-in and register buttons, which take them to the respective pages. Moreover, all pages have a persistent footer that provides contact details in case the user encounters a problem.



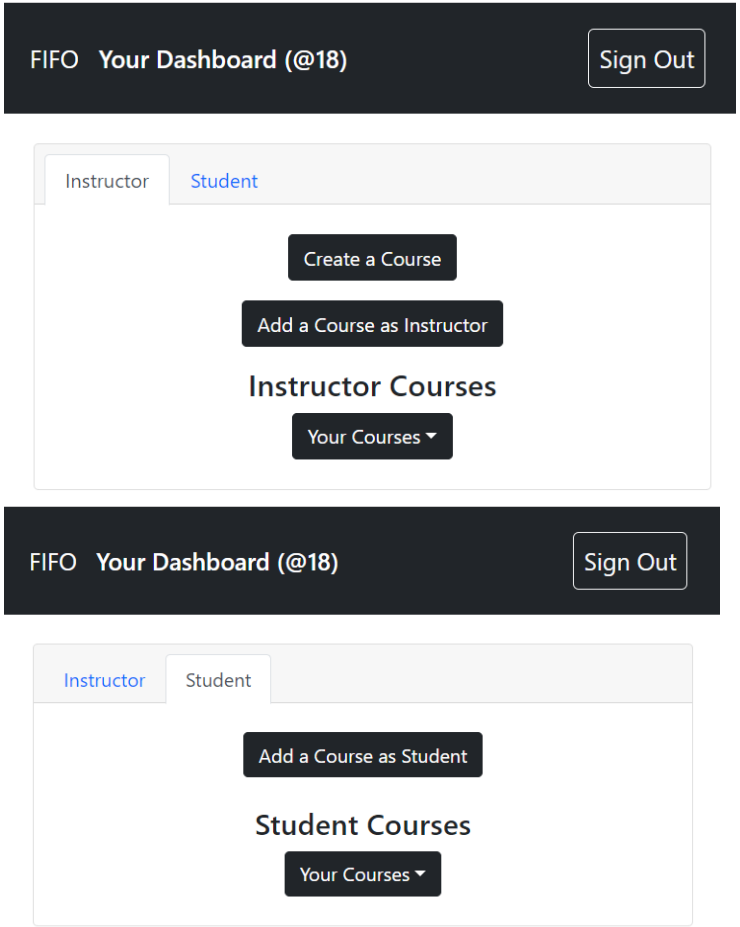
2. Register Page: Users can create their accounts on this page after clicking the Register button on the homepage. Again, this page is minimalistic only serving the purpose it is designed to serve - allowing a user to enter a username and password without any distractions.



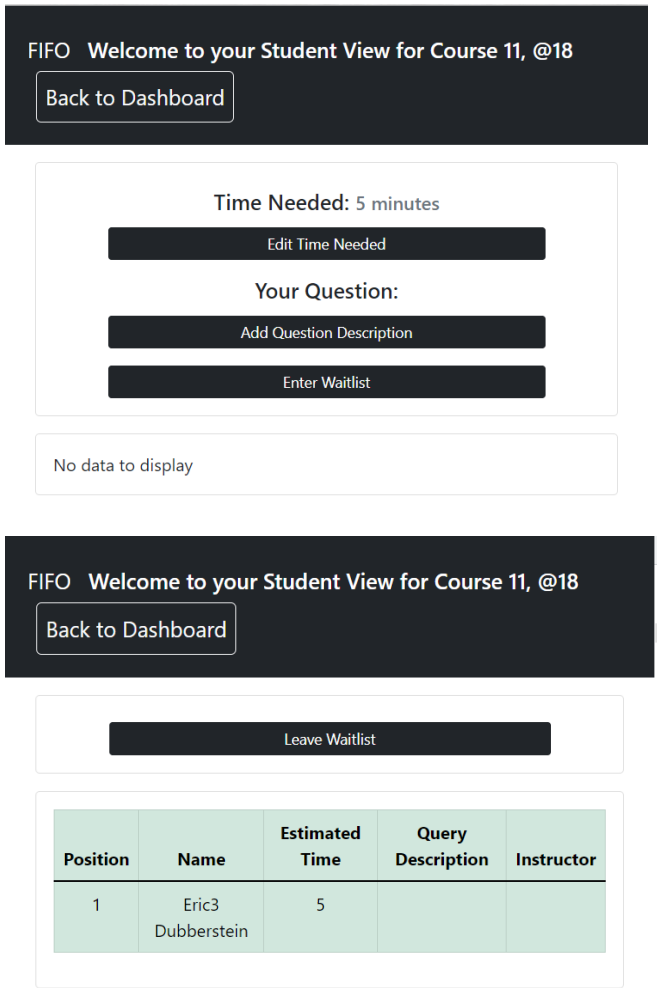
3. Sign-in Page: Allows a user to sign in using their account credentials after clicking the sign in button on the homepage. Initial design doesn't take into account that the user might have forgotten their password.



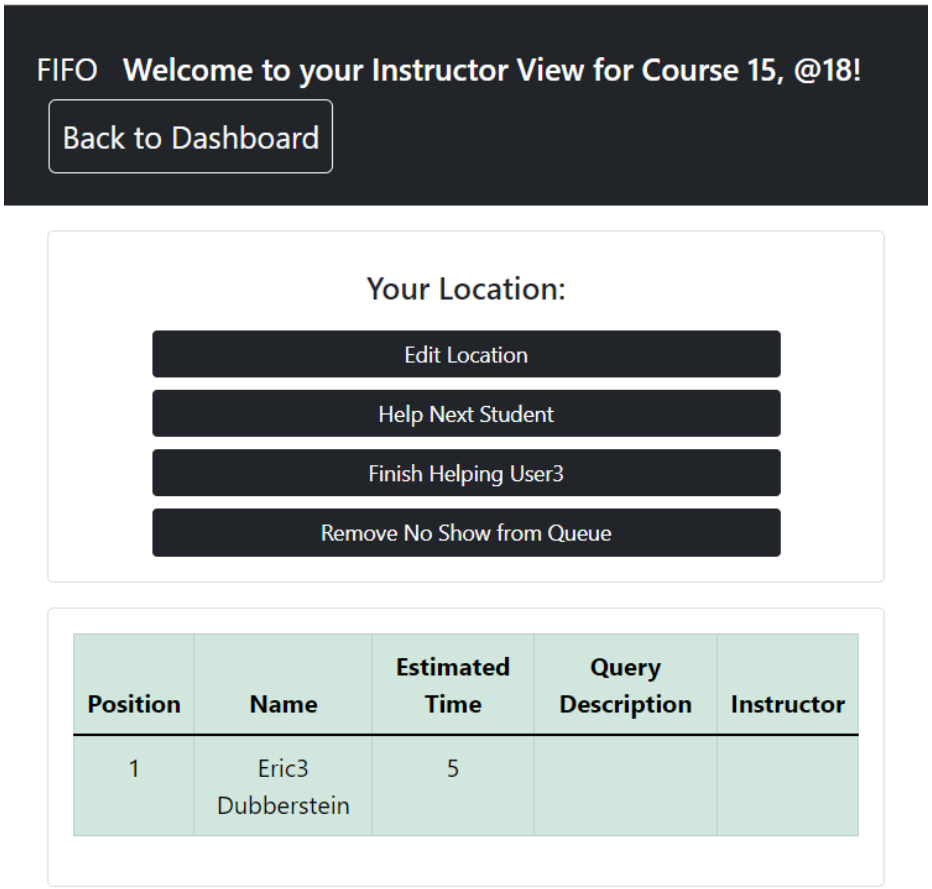
4. Dashboard Page: Users can enter the application in two views - student and instructor. Student mode allows users to add courses they are taking and to see those course queues. Instructor mode allows users with appropriate access to create or join courses, and to see the management page of a course view.



5. Student Page: Join a queue as a student based on instructor preference and estimated wait times for each instructor.



6. Instructor Page: An instructor can manage the queues for the courses they have added/created in their account. Instructors can group students together based on student query classification. The first student in the queue must be picked no matter what. Notifications are sent to each student selected via email.



Security Requirements

For our project, we have to make sure instructors and students have separate sets of options to explore. We have a couple safeguards in place to take care of this security aspect. After registering, instructors get an option to create a course and generate a code for it which students can use to get enrolled in that course in order to book the office hours of the TAs in that course. This will prevent the students from randomly enrolling for any course and disturb the TAs.

After booking office hours successfully, we are making sure students get notified via email properly. The email will contain information like the instructor with whom the student has booked the hours and also the details like timings and the location.

Additionally, it is worth acknowledging that this site would be vulnerable to Denial-of-service attacks. The likely fix to this will be to just shut down the website until the attack stops, which given the non-essential nature of the project is sufficient. Moreover, we could add Captcha checks in the account creation page to stop bots from persistently creating a new account. Protecting against SQL injection is a more pressing security issue given the persistence of the database for the website's functionality.

Lastly, the use of an account based website will ensure that each user is able to only access their own course queues and not anyone else's.

System Requirements

The application will be a web application. The system requirements will be minimal. Any modern HTML-5 browser with Javascript enabled will be able to use the service.

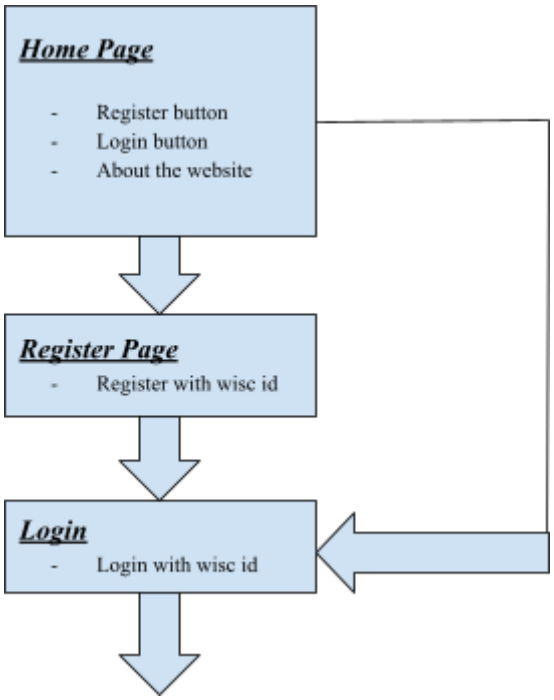
We anticipate using services like Mailchimp to send out emails to notify users about the availability of an instructor. We are planning to use Docker so we can readily develop and deploy on every major operating system such as MacOS, Windows, and Linux. As for the web server requirements, our team is using Express Node.js

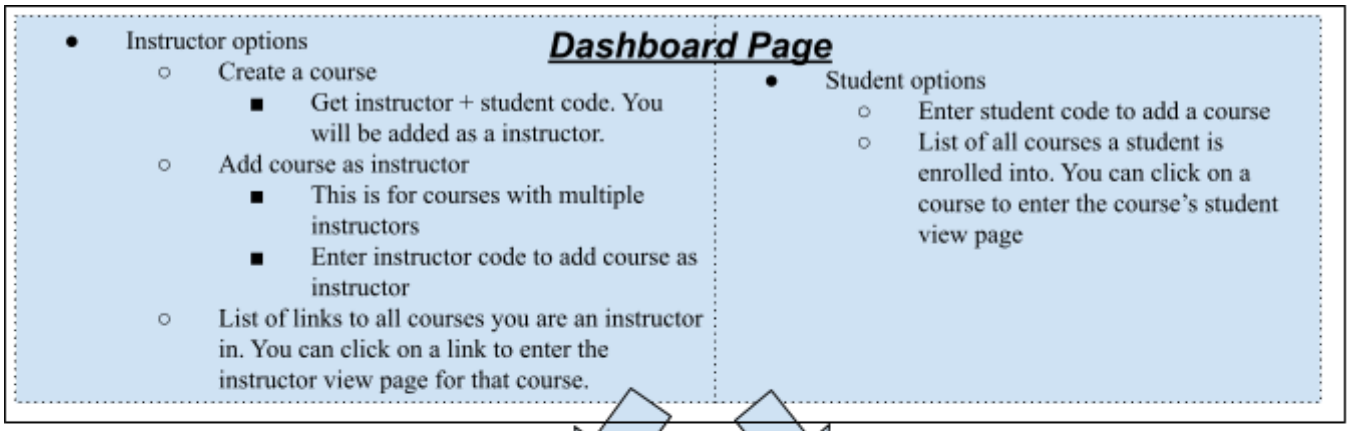
Overall, our application is relatively independent of external services and entities as the data required for our application to operate comes from the user. Moreover, this data is processed directly from our backend and the user interface is updated in an appropriate manner.

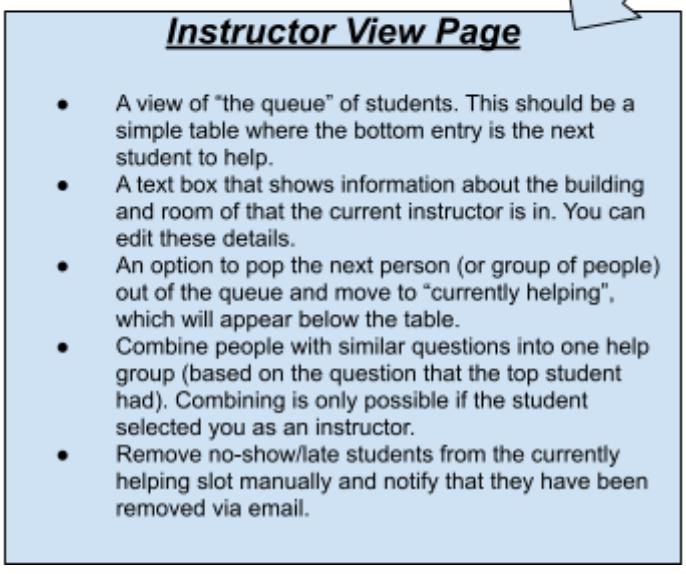
Specification

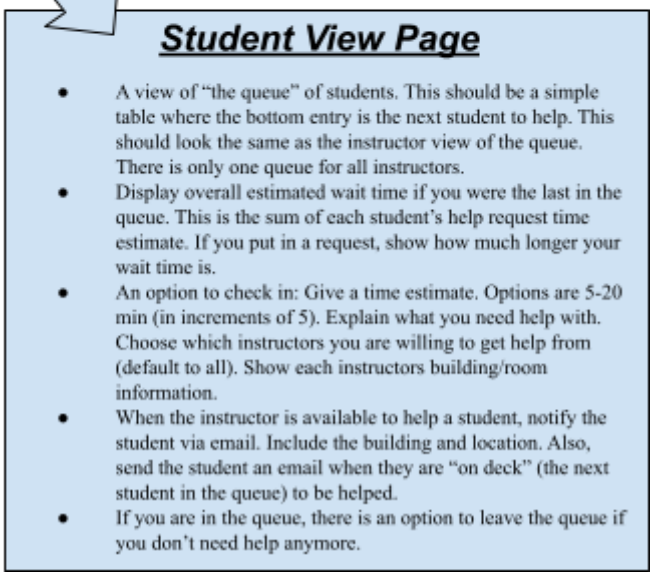
Section 1: Page Layout Diagram

Below is an overview of the structure and skeleton of our website. The diagram showcases a potential flow of events that a potential user can undertake while navigating through the website. There are two main modes/views in our website - instructor and student.









Section 2: MySQL Database Schema (subject to change relative to updates in Section 1).

Queues	
queue_id	int, not null, primary key
course_id	int, not null, foreign key
user_id	int, not null, foreign key
queue_estimated_time	int, not null,
queue_request_status	enum("DONE", "IN_PROGRESS", "WAITING", "CANCELED") not null.
queue_topic_description	varchar(255)
queue_timestamp	timestamp, not null
queue_instructor_user_id	int, foreign key (the instructor that is helping). Can be null

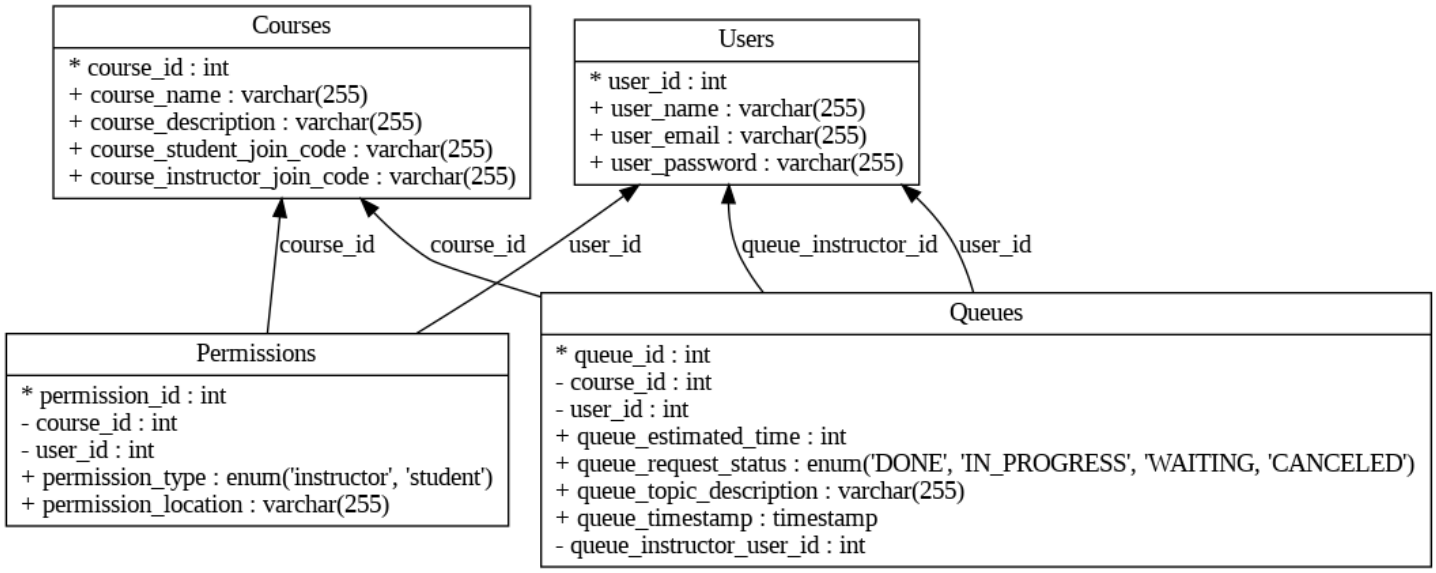
Courses	
course_id	int, primary key, not null
course_name	varchar(255)
course_description	varchar(255)
course_student_join_code	varchar(255)
course_instructor_join_code	varchar(255)

Users	
user_id	int, primary key, not null
user_name	varchar(255). E.g. “Eric Dubberstein”
user_email	varchar(255)
user_password	varchar(255)

Permissions	
permission_id	int, primary key
course_id	int, foreign key
user_id	int, foreign key

permission_type	enum("INSTRUCTOR","STUDENT")
permission_location	varchar(255), can be null. This stores the location for where an instructor is holding office hours for that class.

Relational Diagram:



API Calls:

Student Route:

POST: student/enqueue:

Parameters:

user_id, course_id, queue_estimated_time, queue_topic_description

Backend/DB actions:

- Check the permissions table with the user_id and course_id to make sure that this user is a student in this course.
- Add user to queue with new queue_id (new row in queue table), where queue_request_status will be set to “WAITING”, and information from API call is saved in the entry (user_id, course_id, queue_estimated_time, queue_topic_description)

POST: student/exitQueue:

Parameters: user_id, course_id

Backend/DB actions:

Check the permissions table with the user_id and course_id to make sure that this user is a student in this course. For the entry (row) in the Queue table with the matching user_id and course_id, set the queue_request_status to “CANCELED”

GET: student/getQueueStatus:

Parameters: user_id, course_id

Backend/DB actions:

Check the permissions table with the user_id and course_id to make sure that this

user is a student in this course. If so, return the entries in the Queue table that have a matching course_id and have a queue_request_status of “WAITING”. Sort the entries by the queue_timestamp field. Join with the student table to receive the student name based on the user_id. The final result should have user_name, queue_estimated_time, and queue_topic_description for each user in the queue.

Instructor Route:

GET: instructor/getQueueStatus:

Parameters: user_id, course_id

Backend/DB actions:

Check the permissions table with the user_id and course_id to make sure that this user is a instructor in this course. If so, return the entries in the Queue table that have a matching course_id and have a queue_request_status of “WAITING”. Sort the entries by the queue_timestamp field. Join with the student table to receive the student name based on the user_id. The final result should have queue_id, user_name, queue_estimated_time, and queue_topic_description for each user in the queue.

POST: instructor/takeNextStudent:

Parameters: user_id, course_id

Backend/DB actions:

Check the permissions table with the user_id and course_id to make sure that this user is a instructor in this course. Then, find the next person on the queue (select top 1 from queue table with matching course_id filter by least recent helped where queue_request_status is “Waiting”). Update this entry to the queue so that queue_request_status is “IN_PROGRESS” and the queue_instructor_user_id is set to the user_id of the instructor that made this request.

POST: instructor/finishHelpingStudent:

Parameters: user_id, course_id

Backend/DB actions:

Check the permissions table with the user_id and course_id to make sure that this user is a instructor in this course. Find the user that this instructor is currently helping by searching the Queue table for an entry where queue_request_status is “IN_PROGRESS” and the queue_instructor_user_id is the user_id of the instructor making the request. Update this entry to the queue so that queue_request_status is “DONE”.

POST: instructor/removeNoShowStudent:

Parameters: user_id, course_id

Backend/DB actions:

Check the permissions table with the user_id and course_id to make sure that this user is an instructor in this course. Find the user that this instructor is currently helping by searching the Queue table for an entry where queue_request_status is “IN_PROGRESS” and the queue_instructor_user_id is the user_id of the instructor making the request. Update this entry to the queue so that queue_request_status is “CANCELED”.

POST: instructor/setRoomInfo:

Parameters: user_id, course_id, permission_location

Backend/DB actions:

Check the permissions table with the user_id and course_id to make sure that this user is an instructor in this course. Update this entry (row) of the permissions table, setting permission_location field to the value passed into the API call.

GET: instructor/getRoomInfo:

Parameters: user_id, course_id

Backend/DB actions:

Check the permissions table with the user_id and course_id to make sure that this user is an instructor in this course. Return the permission_location field from the permissions table for the matching entry with the appropriate user_id and course_id.

GET: instructor/getCurrentlyHelpingStudent:

Parameters: user_id, course_id

Backend/DB actions:

Check the permissions table with the user_id and course_id to make sure that this user is an instructor in this course. Find the user that this instructor is currently helping by searching the Queue table for an entry where queue_request_status is "IN_PROGRESS" and the queue_instructor_user_id is the user_id of the instructor making the request. Join with the user table based on the user_id of the student in this queue slot, and return this student's name. Also return queue_topic_description and queue_estimated_time for this row in the queue.

Account:

POST: AccountCreateAccount:

Parameters: wisc.edu email, password, name

Backend/DB actions:

Check the user table to see if there is already an account for the email address supplied. If there is, return a 409 (front end should then prompt the user to sign-in). Otherwise, create a new row in the user table with the corresponding name, email, and password from the parameters.

GET: AccountSignIn:

Parameters: wisc.edu email, password

Backend/DB actions:

Check the user table to see if there is already an account for the email address and password supplied. If there isn't, return a 409 (front end should then say incorrect password). Finally, return the User_id for this user

Dashboard:

POST: CreateCourse:

Parameters: user_id, course_name, course_description

Backend/DB actions:

Create a new course (row) in the Course table using the course_name and course_description from the parameters. The student and instructor join codes should each be sudo-randomly generated 6 character alpha-numeric codes with the course_id appended to ensure 1-1 mapping from join code to course_id.

Using the course_id for the newly created course, add a new row to the permissions table such that the user that is creating the course is an instructor for that course. In other words, the SQL should look something like:
INSERT INTO permissions (course_id, user_id, permission_type, permission_location) VALUES (course_id_from_newly created_course,
user_id_of_user_who_made_this_request_from_api_parameters, “INSTRUCTOR”,NULL).
Return the course_id for the newly created course to the frontend.

GET: GetCourseJoinCodes:

Parameters: user_id, course_id

Backend/DB actions:

Check the permissions table with the user_id and course_id to make sure that this user is an instructor in the course specified by the course_id. Then run a select query in the Courses database to return the course_student_join_code and course_instructor_join_code.

POST: JoinExistingCourseAsInstructor:

Parameters: user_id, course_name, course_instructor_join_code

Backend/DB actions:

Check if the course_instructor_join_code exists in the course table. If so, pull the course_id from the course table corresponding to this row.

Using the course_id for the course, add a new row to the permissions table such that the user making the request is an instructor for that course. In other words, the SQL should look something like:
INSERT INTO permissions (course_id, user_id, permission_type, permission_location) VALUES (course_id_from_step1, user_id_of_user_who_made_this_request_from_api_parameters, “INSTRUCTOR”,NULL)

POST: JoinExistingCourseAsStudent:

Parameters: user_id, course_student_join_code

Backend/DB actions:

Check if the course_student_join_codeexists in the course table. If so, pull the course_id from the course table corresponding to this row.

Using the course_id for the course, add a new row to the permissions table such that the user making the request is a student for that course. In other words, the SQL should look something like:
INSERT INTO permissions (course_id, user_id, permission_type, permission_location) VALUES (course_id_from_step1, user_id_of_user_who_made_this_request_from_api_parameters, “STUDENT”,NULL)

GET: GetCoursesEnrolledAsStudent

Parameters: user_id

Backend/DB actions:

Check if the user table to see if this user exists. If so, return a select query on the permissions table filtering by matching user_id from parameter list and permission_type = “STUDENT”. Join with the course table on course_id to get the name of each course. The final result returned should be a list of course_id, course_name pairs. E.g. in json

[

{course_id: 1234, course_name: “Algorithms”},

{course_id: 5678, course_name: “Data Structures”}

]

GET: GetCoursesEnrolledAsInstructor

Parameters: user_id

Backend/DB actions:

Check if the user table to see if this user exists. If so, return a select query on the permissions table filtering by matching user_id from parameter list and permission_type = “INSTRUCTOR”. Join with the course table on course_id to get the name of each course. The final result returned should be a list of course_id, course_name pairs. E.g. in json

[

{course_id: 1234, course_name: “Algorithms”},

{course_id: 5678, course_name: “Data Structures”}

]