



CSCI 201 Final Project: Patricia's Polls!

University of Southern California

Fall 2020

Team Members:

Jay Lin (linjay@usc.edu)

Jerry Li (cli49003@usc.edu)

Anna Hong (annahong@usc.edu)

Harshitha Padiyar (padiyar@usc.edu)

Uma Durairaj (uduraira@usc.edu)

Vincente Mai (vmmai@usc.edu)

Table of Contents

| | |
|---|-----------|
| Table of Contents | 2 |
| Project Proposal (Week 1) | 4 |
| Technical Specifications and Full Design (Weeks 2&3) | 5 |
| Introduction | 5 |
| Purpose | 5 |
| Project Scope | 5 |
| SRS | 5 |
| Product Perspective | 5 |
| Authentication System | 5 |
| Description | 5 |
| Requirements | 6 |
| Technical Specifications | 6 |
| Class Enrollment | 6 |
| Description | 6 |
| Requirements and Implementation | 6 |
| Technical Specifications | 7 |
| Polling System | 7 |
| Description | 7 |
| Requirements and Implementation | 7 |
| Technical Specifications | 8 |
| Display of Results | 8 |
| Description | 8 |
| Requirements and Implementation | 9 |
| Technical Specifications | 9 |
| Office Hours Queue | 9 |
| Description | 9 |
| Requirements and Implementation | 10 |
| Technical Specifications | 11 |
| Chat | 11 |
| Description | 11 |
| Requirements and Implementation | 11 |
| Technical Specifications | 12 |
| Database: EER Diagram | 12 |
| User Interface and Front End | 12 |
| Classes and Data Structures | 13 |

| | |
|---------------------------------|-----------|
| Testing (Week 4) | 20 |
| Polling System Features | 20 |
| Display of Results | 21 |
| Authentication | 22 |
| Enrollment | 23 |
| Office Hours Queue | 24 |
| Chat | 27 |
| JUnit Testing - Chat | 28 |
| Deployment Documentation | 28 |
| Required Software and Tools | 28 |
| Getting Started | 29 |

Project Proposal (Week 1)

Our project will be a polling app meant to help professors better understand their students' progress in their courses. In this new age of online learning, having one consolidated platform where students and professors can interact about their classes would be incredibly helpful. Professors need to be able to get a better idea of how their students are handling the workload, understanding the material, participating in the class, and getting the help they need. This app is intended for use in parallel with online classes, but could be modified for use with in-person learning. Some features of our app include polls, chats, enrollment in multiple classes, and office hours queues, as well as different permissions and levels of access for students, instructors, and guests. Polls can function as a quick quiz method to gauge understanding or just as a way to check in with students and their progress. Chats can help people connect and can be anonymous, between students in the same class, or between students and professors. Registered users will be able to join multiple classes, respond to polls, and access the resources available. The office hours queues will act as our additional multithreading and networking functionality and allow instructors to optimize their teaching environment.

Our app will allow guest users or members but they will have differing levels of access, using the Google Sign In API for user authentication. All users will be able to view public polls. Authenticated student users with accounts will have additional functionality of voting on polls for classes they are registered in, chatting directly with their professors, and joining the office hours queue for their registered professors. Authenticated professor users with accounts will be able to create classes, create polls, view comprehensive polling results, and set constraints on their office hours queue. Poll responses from registered users will be stored in the local MySQL database.

Technical Specifications and Full Design (Weeks 2&3)

Introduction

Purpose

This SRS outlines the functional requirements for the web app. This document is intended for use by the team building it.

Project Scope

This project will be a web-based polling application that will allow instructors to measure students' progress through class or university-wide polls, and connect with students through office hours.

SRS

Product Perspective

This product will allow instructors to create classes, which students can enroll in using a class code. Instructors can create private class polls or public polls to measure students' progress. Students can vote in any type of poll, while guest users may only vote in public polls. Instructors may view the results of private class polls and public polls. Additionally, instructors can create an office hours meeting and set certain constraints on it. Students may join or wait to join the office hours meeting for classes they are enrolled in, while guest users are blocked from entering office hours. Students will be able to chat with their instructors or guest users.

Authentication System

Description

An instructor can create an instructor level account with full functionalities, and a student can create a student level account with some limited functionalities. Creating either type of account requires signing in with a Google account. A guest can also view the main page with public polls, and will experience restricted functionalities with no access to other individual class pages.

Requirements

- **Instructor Specific Requirements**
 - An instructor can create an account by signing in through the Google API with their Gmail address.
- **Student Specific Requirements**
 - A student can create an account by signing in through the Google API with their Gmail address.
- **Guest Specific Requirements**
 - A guest user may access the app anonymously, without creating an account.

Technical Specifications

Implementation:

- Estimated: 3 hours
- Google Sign-In and OAuth 2.0 are used to authenticate users and allow them to create accounts. Users can specify when they log in whether they want to sign in as a student or as an instructor.
- Users will be redirected to sign in with Google, and information will be retrieved from their profiles and stored in the UserInfo table of the database (first name, last name, email address, and account level).

Class Enrollment

Description

Instructors will be able to create a class with a class code, and students will be able to enroll in different classes using those codes. Enrollment in a class gives students access to private class polls and office hour meetings.

Requirements and Implementation

- **Instructor Specific Requirements**
 - An instructor can create a class which will have a unique class code. An instructor may create multiple classes.
 - An instructor may see a list of classes that they have created and the list of students enrolled in each class.
- **Student Specific Requirements**

- A student may enroll in a class using the unique class code. A student may enroll in multiple classes.
- A student can see the list of classes that they are enrolled in and the list of other students enrolled in the same class.
- A student enrolled in a class has access to specific features including the private class polls and office hours meetings (and chatting with the professor).
- **Guest Specific Requirements**
 - A guest is unable to enroll in any class.

Technical Specifications

Implementation:

- Estimated: 5 hours
- Class enrollment is implemented through the database. A Class table will store all class names with a foreign key of Instructor ID, and a Class Member table will store class codes with student IDs to track membership in courses.
- The class code will be randomized and alphanumeric. The program will check that the code doesn't already exist, and will then assign the code to a newly created class.

Polling System

Description

The polling system will allow instructors to create private class polls or public polls consisting of a multiple choice question. Only students enrolled in the instructor's class may vote in private class polls. Any student and guest users may vote in public polls. Private class poll results and public polls will be stored for a set amount of time.

Requirements and Implementation

- **Instructor Specific Requirements**
 - Instructors can create a new poll and will be prompted to enter a question with up to four response options.
 - Instructors can create either private class polls or public polls.
 - Polls will have a unique question ID and answer choices will each have a unique answer ID.
 - Polls contain a single multiple choice question.

- Polls may have time constraints set by the instructor. Students will only be able to vote on a poll during a certain period of time.
- **Student Specific Requirements**
 - Students may submit their vote in public polls or private class polls.
 - Student responses will be stored in the program.
- **Guest Specific Requirements**
 - Guests may submit their vote in public polls.
 - Guest responses will be counted and in the program.

Technical Specifications

Implementation:

- Estimated: 4-7 hours
- Polls will be tracked through storing them in the database.
 - “Poll” table
 - Tracks all polls created. The question, class code (if applicable), instructor ID, response list ID, and whether or not the poll is public will be attributes in the table.
 - “Response” table
 - Tracks each response, number of votes received, and a foreign key for the question ID of the poll associated with each response.
 - “UserResponse” table
 - Tracks student votes by storing student ID, response ID, and question ID. Can be used to avoid duplicate voting, and for the instructor to monitor who in a class has responded.

Display of Results

Description

At any time, instructors can view the results of the polls they sent out. Students can view the results of private polls after they have answered them.

Requirements and Implementation

- **Instructor Specific Requirements**
 - An instructor can select whether the poll is public or private.
 - An instructor can view the results of their polls at any time.

- An instructor can view poll results which have been limited to be only available to the instructor.
- **Student Specific Requirements**
 - A student can only view the results after they have answered the poll. The page should display the question, answer choices, and the number of votes for each choice.

Technical Specifications

Implementation:

- Estimated: 2-4 hours
- Poll results will be obtained through database queries in a Poll Database Handler file.
- The program must track the list of students who responded with each answer choice
- Any Modification to the response pool is a signal to update the display of the result.

Office Hours Queue

Description

Instructors are able to create an office hours meeting queue with a set limit on the number of students in the meeting room and the amount of time they can be in the meeting room. Students in the instructor's class may join the office hours queue. If the room is not full, they will enter it; otherwise, they will wait in the waiting list until someone else leaves the room, and will then be invited to join the meeting. If the student accepts the invitation, then the server will send them a link to the meeting room.

Requirements and Implementation

- **Instructor Specific Requirements**
 - An instructor can create a new office hours meeting
 - Office hours meeting rooms have size constraints set by the instructor. Only a certain number of students may join the office hours meeting room, and others in the waiting list must wait for people to leave the meeting in order to join it.
 - Office hours meeting rooms have a time slot length for students to be in the meeting room before their turn in the meeting is considered over and the next student waiting is invited to the meeting. A student is invited to

the meeting and must accept in order for the server to send them the link to the meeting.

- Office hours meeting queues have enrollment constraints. Only students enrolled in a class may join the office hours meeting queue for that class.
- Office hours meeting queues have time constraints set by the instructor. Students will only be able to try to join the office hours meeting queue during a certain period of time when the meeting is open.

- **Student Specific Requirements**

- A student may join office hours if constraints are satisfied. They must be enrolled in the instructor's class and office hours must be open at the time. They must enter the class code to join.
 - If these constraints are not satisfied, the student is unable to join the office hours queue.
- A student will be added into the waiting list if they try to join the office hours meeting and the meeting is full.
- A student will be invited to join the meeting when there is space in the meeting. They must accept the invitation and the server will then send them the meeting Zoom link. They are given 30 seconds to accept their invitation, or else their spot is lost and the next waiting student is invited. Once the invitation is accepted, the student's turn in the meeting begins and lasts for as long as the time slot set by the Instructor.

- **Guest Specific Requirements**

- A guest will not have the option to join the office hours queue.

Technical Specifications

Implementation:

- Estimated: 7-8 hours
- The office hours queue will be implemented through multithreading using semaphores.
- Networking using sockets will be used to send the link to students that are invited to join the meeting
- Office hours connects to the database to ensure proper enrollment for Students and Instructors.
- ** NOTE: Unable to implement using Servlets and JSP. Must be run separately from a Java application. Still integrated with the local database.

Chat

Description

A user will be able to create a chat with another user. Instructors will be able to create chats with or receive chats from students enrolled in one of their classes. Guests will not be able to send or receive chats.

Requirements and Implementation

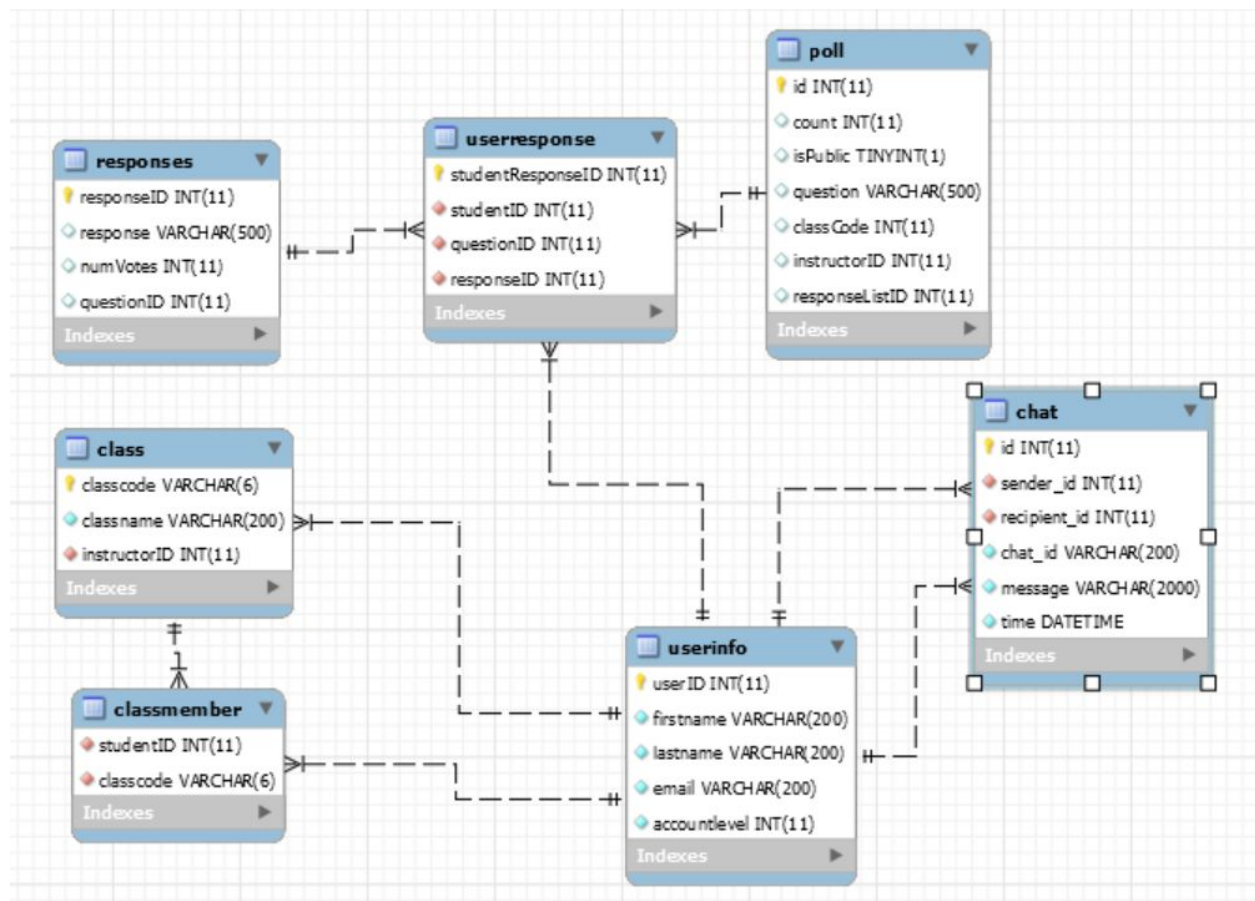
- **Instructor Specific Requirements**
 - An instructor will be able to message one-on-one students that are currently enrolled in their classes.
 - An instructor can either start chats with students or receive them
 - An instructor will be restricted from receiving and sending messages from guests or students not in their class.
- **Student Specific Requirements**
 - A student will be able to start messages with or receive messages from instructors of classes they are enrolled in and other students in classes they are enrolled in.
 - A student will be unable to send messages to instructors of classes they are not enrolled in.
- **Guest Specific Requirements**
 - A guest cannot send messages or receive messages.

Technical Specifications

Implementation:

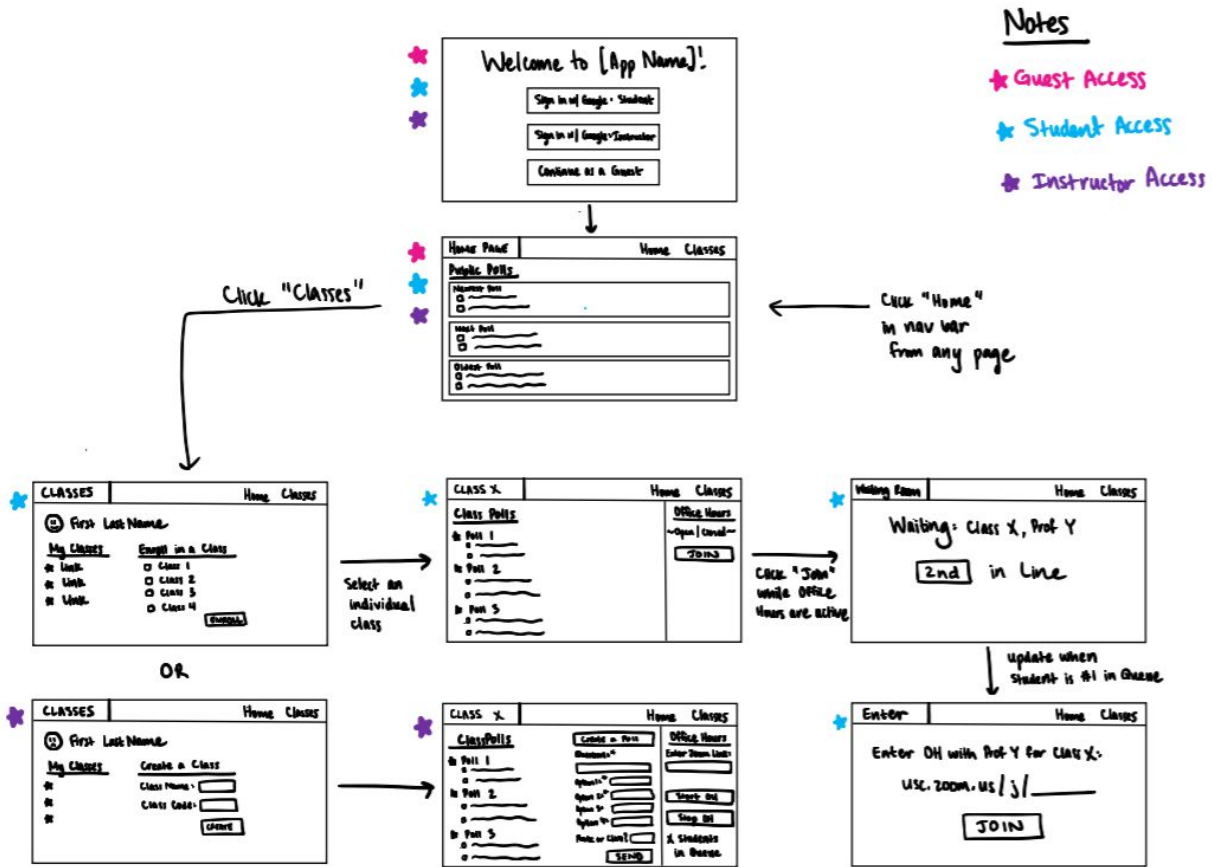
- Estimated: 5-10 hours
- The chat feature will be implemented through networking to store and retrieve messages in a sequential manner. This allows a chatroom to only be created when the first message is sent.
- Utilization of JDBC and Servlets to efficiently query and update the database.
- Messages will be stored in the database, in a “Chat” table that includes sender and recipient IDs, chat ID, message contents, and time sent.
- chat IDs will be created by adding two unique userIDs for quicker queries.

Database: EER Diagram




User Interface and Front End

The following is our Week 3 plan for the GUI, included in that week's full design document.




The below are screenshots of the final user interface:

Sign in as a Student:


Sign in

Sign in as an instructor:


Sign in

Sign in as guest

Instructor Home Page

Welcome

[Create class](#)

[Create poll](#)

[View Results](#)

[Start Chatting](#)

[Start Office Hours](#)

Student Home Page

[Add a new class](#)

[View class polls](#)

[View public polls](#)

[View previous polls](#)

[Start Chatting](#)

[Join Office Hours](#)

Instructor Start Office Hours

annahong312@gmail.com

Class Code
Meeting Limit
Time Slot Length
Zoom Link
Enter length of office hours (in minutes)

Start

Create a Poll

Class Code

Poll Visibility

Private ☐ Public ☐

Question:

Design your answers:

Answer A

Answer B

Answer C

Answer D

Submit

Available Polls

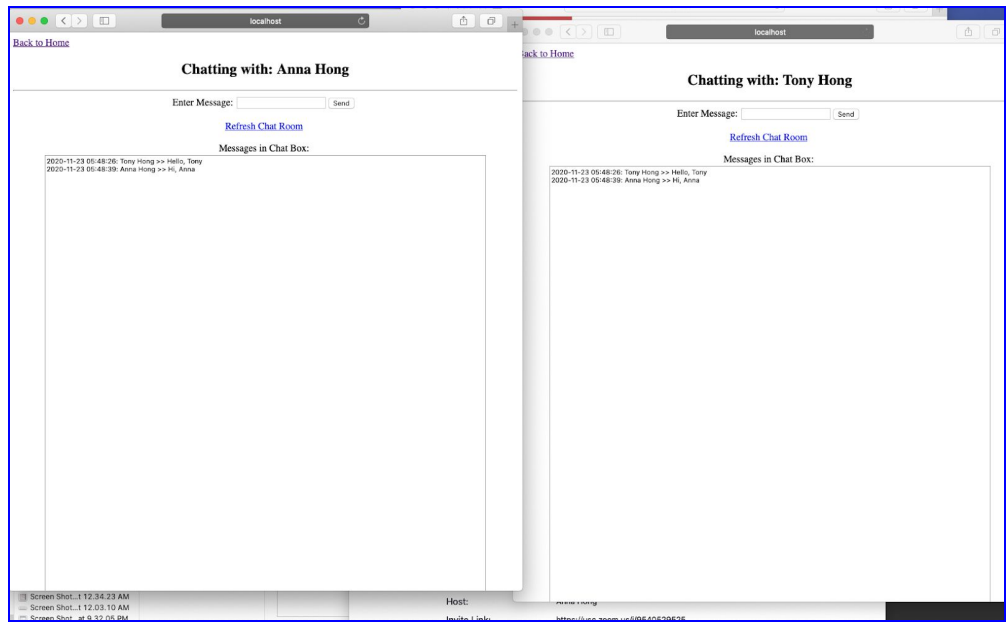
- **Question:** First private poll

- b
- c
- a

- **Question:** Private poll 2

- p
- l
- l
- o

[Home](#)

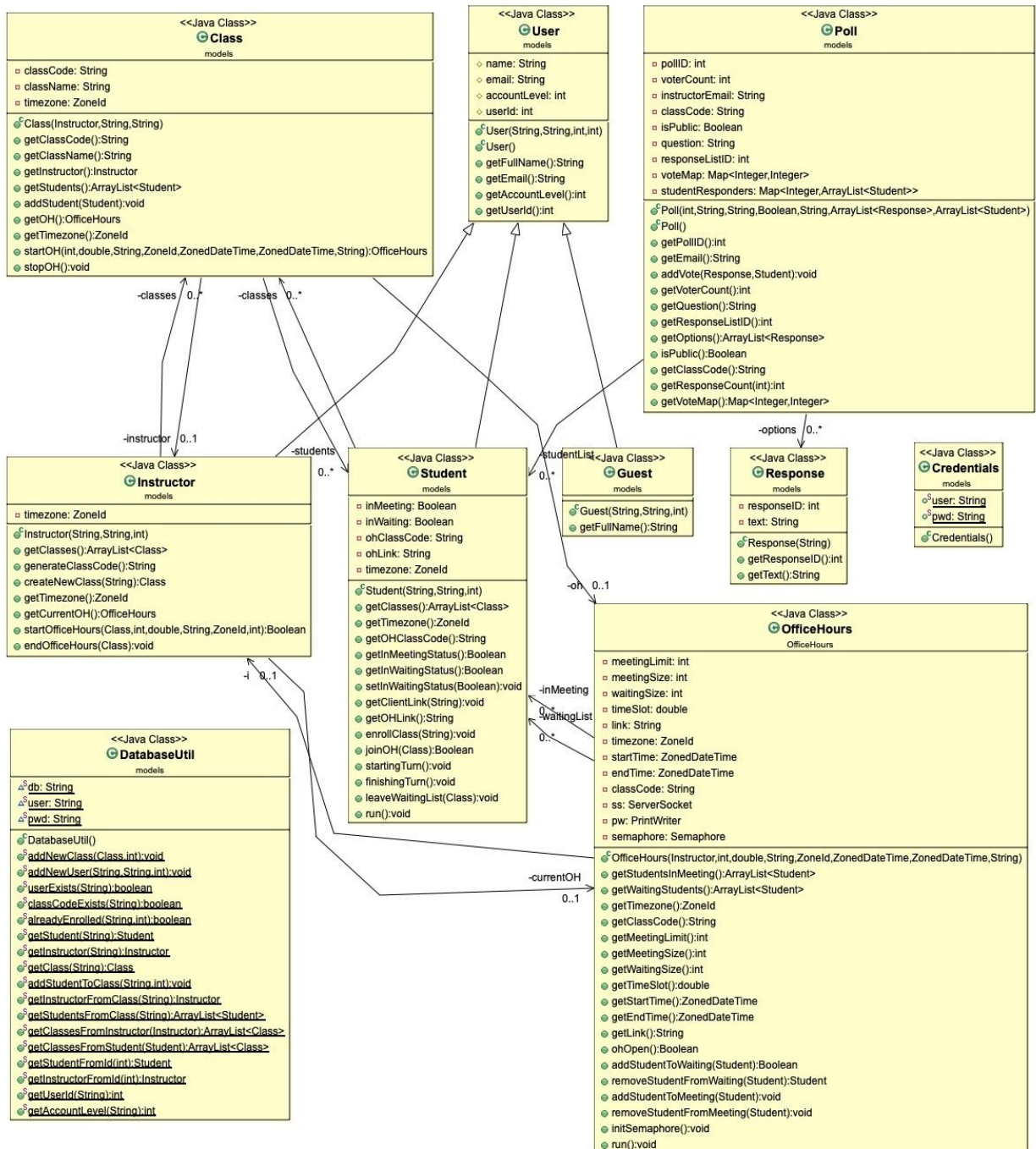


Student Join Office Hours

Class Code

Classes and Data Structures

Class Diagram



- Class Diagram Breakdown

- User (extends Thread)

- Data members:

- ArrayList<Poll> publicPolls

- Functions:

- getPublicPolls()

- Returns ArrayList<Poll> of public polls

- `getFullName()`
 - Returns String of full name
 - Note: for Guest users, this defaults to Anonymous
- **Subclasses:**
 - Student
 - Data members:
 - `ArrayList<Class> classes`
 - `Boolean inMeeting;`
 - `Boolean inWaiting;`
 - `String ohClassCode`
 - `String ohLink`
 - `ZoneId timezone`
 - Functions
 - `getEmail()`
 - Returns String of email
 - `getClasses()`
 - Returns `ArrayList<String>` of classes they enrolled in
 - `getPrivatePolls(String classCode)`
 - Returns `ArrayList<Poll>` of private polls in a given class
 - `getTimezone()`
 - Returns the ZoneID system default of the user, meaning their timezone
 - `getOHClassCode()`
 - Returns the class code of the OfficeHours that a student has most recently joined
 - `getInMeetingStatus()`
 - Returns a Boolean representing whether the Student is in the meeting or not
 - `getInWaitingStatus()`
 - Returns a Boolean representing whether the Student is in the waiting list or not
 - `setInWaitingStatus (Boolean)`
 - Sets the waiting status of the Student
 - `enrollClass (String)`
 - Enrolls a Student in a class based on the code
 - `joinOH (Class)`

- Returns true if a Student successfully joins office hours for a class
- startingTurn()
 - Signals the start of a Student's turn in the office hours meeting
- finishingTurn()
 - Signals the end of a Student's turn in the office hours meeting
- leaveWaitingList(Class)
 - Allows a Student to leave office hours if they are in the waiting list
- run()
 - Runs the Student thread used for office hours meeting turn
- Instructor
 - Data members:
 - ArrayList<Class> classes
 - ArrayList<Poll> privatePolls
 - ZoneId timezone
 - OfficeHours currentOH
 - Functions:
 - getClasses()
 - Returns ArrayList<String> of classes they created
 - createPrivatePoll(String classCode)
 - Creates a private poll in the given class
 - createPublicPoll()
 - Creates a public poll
 - getPrivatePolls(String classCode)
 - Returns ArrayList<Poll> of private polls in a given class
 - getCurrentOH()
 - Returns the OfficeHours that the Instructor currently has open
 - startOfficeHours(Class, int, double, String, ZoneId, int)
 - Returns true if a class' new OfficeHours object is successfully created and started by the Instructor
 - endOfficeHours(Class)

- Ends the Instructor's currently open OfficeHours
 - Guest
 - Data members:
 - String name
 - Default: "Anonymous"
 - Functions:
 - N/A
- Class
 - Data members:
 - String classCode
 - String className
 - Instructor instructor
 - ArrayList<Student> students
 - ArrayList<Poll> privatePolls
 - OfficeHours oh
 - ZoneId timezone
 - Functions:
 - getClassCode()
 - Returns the class code
 - getClassName()
 - Returns the class name
 - getInstructor()
 - Returns the Instructor that created the class
 - getStudents()
 - Returns an ArrayList<Student> of students that are enrolled in the class
 - getPrivatePolls()
 - Returns an ArrayList<Poll> of private polls
 - getOfficeHours()
 - Returns an ArrayList<OfficeHours> of active OH queues
 - addStudent(Student)
 - Adds a Student to students
 - addPrivatePoll(Poll)
 - Adds a Poll to privatePolls
 - getOH()
 - Returns the current OfficeHours for the class
 - getTimezone()
 - Returns the class' timezone (same as the Instructor of the class' timezone)

- startOH(int, double, String, ZoneId, ZonedDateTime, ZonedDateTime, String)
 - Returns the OfficeHours object that it creates for the class
 - stopOH()
 - Ends the OfficeHours for a class
- Poll
 - Data members:
 - String pollID
 - Instructor creator
 - Class myClass
 - Boolean resultVisibility
 - ArrayList<Response> options - List of Response Objects
 - ArrayList<Student> studentList - List of students who responded
 - Map<ResponseID, numVotes> voteMap
 - Number of votes for each response
 - Map<ResponseID, ArrayList<Student> >
 - List of responders for each response
 - Functions:
 - String pollID()
 - Returns pollID
 - Instructor getCreator()
 - returns creator
 - void addVote(Response r, Student s)
 - Adding Response r to the responseList
 - Add s to StudentList in Map
 - ArrayList<Response> getResponders()
 - returns an ArrayList of student object that have responded to the poll
 - boolean getResultVisibility()
 - returns resultVisibility
 - void setResultVisibility(boolean v)
 - If the poll is public, visibility is always true
 - If the poll is private, visibility is true only when className matches one of the String in user's ArrayList<String> classes. If User type is "Guest", then visibility is always false.
- Poll Results
 - Response class - Handling a single response

- Data members:
 - Poll poll - Associated Question
 - String responseID
 - String text

- Functions:
 - getResponseCount()
 - returns number of votes
 - getStudentList()
 - returns list of students associated with the response
 - getPercent()
 - Used for display purposes
 - Returns the percentage of votes that went to a single response

- Office Hours
 - Data members:
 - ConcurrentLinkedQueue<Student> inMeeting
 - Vector<Student> waitingList
 - int meetingLimit
 - int meetingSize
 - int waitingSize
 - Double timeSlot
 - String link
 - ZoneId timezone
 - ZonedDateTime startTime
 - ZonedDateTime endTime
 - String classCode
 - ServerSocket ss
 - PrintWriter pw
 - Semaphore semaphore

 - Functions:
 - getStudentsInMeeting()
 - Returns an ArrayList<Student> of students in the meeting
 - getWaitingStudents()
 - Returns an ArrayList<Student> of students in the waiting list
 - getTimezone()

- Returns the timezone of the OfficeHours (same as the timezone of the Class and Instructor that created it)
- getClassCode()
 - Returns the class code of the Class that has the OfficeHours
- getMeetingLimit()
 - Returns the capacity of the meeting set by the Instructor
- getMeetingSize()
 - Returns the current size of the meeting
- getWaitingSize()
 - Returns the current size of the waiting list
- getTimeSlot()
 - Returns the length of a time slot in the OfficeHours meeting
- getStartTime()
 - Returns the ZonedDateTime start time of the meeting
- getEndTime()
 - Returns the ZonedDateTime end time of the meeting
- getLink()
 - Returns the link associated with the OfficeHours
- ohOpen()
 - Returns a Boolean representing whether the time constraints are met at the current moment, meaning whether OfficeHours are open or not
- addStudentToWaiting (Student)
 - Adds a Student to the waiting list
- removeStudentFromWaiting (Student)
 - Removes a Student from the waiting list
- addStudentToMeeting(Student)
 - Invites a Student to join the meeting, sends them the link to the meeting over a socket if accepted, and transfers them from the waiting list to the meeting
 - If invite to meeting not accepted in 30 seconds, the meeting invite is expired and the Student cannot join
- removeStudentFromMeeting(Student)
 - Remove a Student from the meeting list
- initSemaphore()
 - Initialize the semaphore used to keep the meeting size under the Instructor's set maximum limit
- run()

- Run the OfficeHours thread while within the time constraints and handle Student threads in the waiting list and meeting
- Chat (Stretch Goal)
 - Data members:
 - senderID
 - recipientID
 - Functions:
 - void startChat()
 - Creates a table in the database between two users
 - Queries for available chats that can be started
 - boolean validateUsers(User recipient)
 - Validates if the message can be sent or received based on account levels
 - void sendMessage(Message content)
 - Physically adds the table to the database with new chatID and Message content
 - Void displayMessages(senderID, recipientID)
 - Returns the most recent 5 messages from the database
 - toString() pretty print message content
 - void reloadChat()
 - When called, essentially updates the displayed messages by searching database for new messages sent or received

Testing (Week 4)

Polling System Features

- Name: CreatePoll
 - Purpose: For poll creation, check that the program properly accepts one question and two-four responses
 - Expected Input:
 - Question: “How would you rate your semester?”
 - Responses: “1”, “2”, “3”, “4”, or any combination of those answers
 - Expected Output:
 - Program should prompt professor for a question, at least two responses, and result visibility
 - Possible expected errors:

- Professor only gives one possible response - in this case, the program prompts for another response
 - Professor does not fill in the “Question” box (empty string detected) - in this case, prompt them for a question.
 - Professor does not specify result visibility - must prompt professor for visibility
- Name: CheckPollData
 - Purpose: To ensure that the poll data is properly inserted into the database
 - Expected Input: Same as above
 - Question: “How would you rate your semester?”
 - Responses: “1”, “2”, “3”, “4”, or any combination of those
 - Expected Output:
 - In database, the Poll table should have a row for the new question
 - In database, the Response table should have four additional rows - one for each response option. Each response will have a unique responseId
 - New database information should have the correct professorId, QuestionId, and ResponseId as well as correct Question/Response information. Check this with a query in each table.
 - Edge Cases - n/a
 - Possible expected errors
 - Associated QuestionId and ResponseId do not match the corresponding question / responses
- Name: SingleStudentResponse
 - Purpose: To make sure that each student response is properly recorded in the database
 - Expected Input: One student chooses option “1” from the poll above
 - Expected Output:
 - Use a database query to ensure that there is a new entry under UserResponse.
 - The new entry should have a unique studentResponseID and have the correct studentId, questionId, and responseId associated with it
 - Edge Cases - n/a
 - Possible expected errors
 - Associated QuestionId, ResponseId, and StudentId do not match

Authentication

- Note: These tests were conducted by running signin.jsp on the server, signing in through various accounts (as a Student, Instructor, and Guest) and checking the database for

appropriate changes, as well as checking that each type of user could view the appropriate pages.

- Name: InstructorLogin
 - Purpose: To ensure that instructors are able to create an account.
 - Expected Input: Instructor logs in using their Gmail account - inputs email address and password
 - Expected Output: Instructor's user ID is generated and their info is added as a record in the UserInfo table
 - How to check output is correct: Instructor has the correct level of access to the app; if a returning user, the correct relevant information is shown; the instructor is able to create polls; the UserInfo table in the database contains the correct details (first name, last name, email address)
 - Possible expected errors?
 - Instructor inputs incorrect login information - should be handled by the Google API, it will prompt the user to input correct Gmail login credentials.
- Name: StudentLogin
 - Purpose: To ensure that students can create an account.
 - Expected Input: Student logs in with their Gmail account - inputs email address and password
 - Expected Output: Student's info is added into the UserInfo table
 - How to check output is correct: Student has appropriate level of access to the app; if a returning user, the relevant classes are shown; the UserInfo table contains the correct details.
- Name: GuestLogin
 - Purpose: To ensure that guests can access the app without authentication.
 - Expected Input: A user clicks the "continue as guest" button on the landing page
 - Expected Output: Guest can access the public polls page of the app and vote in them, but nothing else
 - How to check output is correct: Check that the guest has the appropriate level of access and can't enroll in classes or access class pages.

Enrollment

- Name: SingleStudentEnrollment
 - Purpose: To ensure that student users are correctly enrolled in the classes they choose.
 - Expected Input: Student's user ID and class code
 - Expected Output: Student added to class

- How to check output is correct: Query the ClassMember table in the database and check that a record with the correct student user ID and class code has been created
 - Edge Cases
 - Wrong class code/class doesn't exist- error message: class DNE
 - Inputting a class code for a class that the student is already enrolled in- nothing should happen, table should only contain the student once
- Name: SingleInstructorEnrollment
 - Purpose: To ensure that instructor users are correctly creating classes.
 - Expected Input: Instructor's user ID and class name
 - Expected Output: An unused, random class code is generated and the class code corresponds to a new class with the class name
 - How to check output is correct: Query the Class table in the database and check that a record with the correct instructor user ID, class code, and class name has been created
 - Check to ensure that the created class code has not yet been used in the Class table
 - Edge Cases
 - Class name already exists- class name does not need to be unique, but class code has to be
- Name: MultipleStudentEnrollment
 - Purpose: To ensure that multiple student users can correctly enroll in the classes they choose.
 - Expected Input: For loop of students' user ID and class code
 - Expected Output: Students added to class
 - How to check output is correct: Query the ClassMember table in the database and check that all the students added are correct
 - Edge Cases
 - Large numbers of students
- Name: MultipleInstructorEnrollment
 - Purpose: To ensure that multiple instructor users are correctly creating classes
 - Expected Input: For loop of instructors' user ID and class names
 - Expected Output: All unused, random class codes are generated and the class codes correspond to a new class with the class name
 - How to check output is correct: Query the Class table in the database and check that all records with the correct instructor user ID, class code, and class name have been created
 - Check to ensure that the created class codes have not yet been used in the Class table
 - Edge Cases

- Multiple instructors create a class with the same name
 - One instructor creates multiple classes with the same name
- NOTE: The GUI will not have an option for instructors to enroll in a class, nor will it have an option for students to create a class. Guests will not be able to do either.

Office Hours Queue

- **Important Notes:**
 - Using a testing file titled “OHTest.java” since JUnit is not ideal for testing multithreading
 - ***Make sure to run each test one by one (comment out the rest) or else threads from different tests may overlap and cause errors. Tests may take time to run so be patient! **
 - If repeating any tests, make sure to clear the database so as not to duplicate them
 - If you choose to edit the length of office hours set by instructors, ensure that the Thread sleeps in that test for that amount of time so that office hours will close within the test.
 - Instructor inputs office hours constraints as follows: Class, meeting limit, time slot, link, start time, meeting length (in minutes)
- Name: InstructorCreateOH
 - Purpose: Check that the Instructor can create OH for one of their classes
 - Input: Instructor, Class
 - Expected Output: returns true
 - Office hours opens and closes, and the difference in time between the two is equivalent to the length of the meeting (set by instructor)
- Name: InstructorCreateOHWrong
 - Purpose: Check that the Instructor cannot create OH for a class that is not theirs
 - Input: Instructor, Class
 - Expected Output: returns false
 - Office hours does not open
- Name: AddStudentBeforeStarted
 - Purpose: Check that a Student cannot join OH if it has not been started by the Instructor yet
 - Input: Instructor, Class, Student
 - Expected Output: returns false
 - Office hours does not open
 - Student does enter meeting or start turn
- Name: AddStudentAfterEnded
 - Purpose: Check that a Student cannot join OH if it has already ended
 - Input: Instructor, Class, Student

- Expected Output: returns false
 - Office hours opens and closes, and the difference in time between the two is equivalent to the length of the meeting (set by instructor)
 - Student does not enter meeting or start turn
- Name: AddStudentWrongClass
 - Purpose: Check that a Student cannot join OH if they are not enrolled in the class
 - Input: Instructor, two Classes, Student
 - Expected Output: returns false
 - Office hours opens and closes, and the difference in time between the two is equivalent to the length of the meeting (set by instructor)
 - Student does not enter meeting or start turn
- Name: AddStudentToOH
 - Purpose: Check that a Student correctly joins office hours within the constraints
 - Input: Instructor, Class, Student
 - Expected Output: returns true
 - Office hours opens and closes, and the difference in time between the two is equivalent to the length of the meeting (set by instructor)
 - Student joins office hours queue, then is in waiting list, and then is invited to join the meeting
 - If tester runs “ClientThread.java” when prompted, the Student will receive the link over the socket and start their turn in the meeting
 - Student ends their turn after the length of the time slot has elapsed
- Name: AddStudentToFullOH
 - Purpose: Check that a Student correctly joins the waitlist when OH is full, and is then invited to meeting when another Student leaves
 - Input: Instructor, Class, multiple Students
 - Expected Output: returns true
 - Office hours opens and closes, and the difference in time between the two is equivalent to the length of the meeting (set by instructor)
 - Student joins office hours queue, then is in waiting list, and then is invited to join the meeting if there is space
 - Order of arrival of the students is maintained.
 - If tester runs “ClientThread.java” when prompted, the Student will receive the link over the socket and start their turn in the meeting
 - If “ClientThread.java” is not run, then after 30 seconds, the meeting invite will expire and the next waiting student is offered an invite.
- Name: AddMultipleStudentsOH
 - Purpose: Check that multiple students joining office hours at the same time is handled properly

- Input: Instructor, Class, multiple Students
- Expected Output: returns true
 - Office hours opens and closes, and the difference in time between the two is equivalent to the length of the meeting (set by instructor)
 - Student joins office hours queue, then is in waiting list, and then is invited to join the meeting if there is space
 - Students that join at the same time will be randomly ordered.
 - If tester runs “ClientThread.java” when prompted, the Student will receive the link over the socket and start their turn in the meeting
 - If “ClientThread.java” is not run, then after 30 seconds, the meeting invite will expire and the next waiting student is offered an invite.
- Name: StressTestOHJoins
 - Purpose: Check that office hours works with more students added!
 - Input: Instructor, Class, many Students
 - Expected Output: returns true
 - Office hours opens and closes, and the difference in time between the two is equivalent to the length of the meeting (set by instructor)
 - Student joins office hours queue, then is in waiting list, and then is invited to join the meeting if there is space
 - Order of arrival of the students is maintained. Students that join at the same time will be randomly ordered.
 - If tester runs “ClientThread.java” when prompted, the Student will receive the link over the socket and start their turn in the meeting
 - If “ClientThread.java” is not run, then after 30 seconds, the meeting invite will expire and the next waiting student is offered an invite.

Chat

- Name: startChat
 - Purpose: To ensure that chat room is created between two specified users
 - Expected Input: A user clicks the “Send Message” button on the user and retrieves both userIDs
 - Expected Output: Sender can now send a message to the recipient
 - How to check output is correct: database table will be created between the two users
 - Edge Cases
 - Start chat with user that does not exist
- Name: validateUsers

- Purpose: To ensure that senders/recipients are authenticated to receive/send a message
- Expected Input: Guest, Instructor, and Student
- Expected Output: none
 - How to check output is correct: chatroom should only be created if users meet the requirements to send or receive messages
- Edge Cases
 - userID does not exist and guest accounts
- Name: sendMessage
 - Purpose: to make sure message is sent to the other user
 - Expected Input: userIDs, messageContent
 - Expected Output: none
 - How to check output is correct: messageID created for the new message, and assigned to the right recipient
 - Edge Cases
 - Empty message should not be executed
- Name: displayMessages
 - Purpose: to make sure messages are displayed correctly/inorder
 - Expected Input: userIDs
 - Expected Output: list of expected messages
 - How to check output is correct: the returned list of recent cached messages should be the same as expected list
 - Edge Cases
 - n/a

JUnit Testing - Chat

- After deploying the project, run Java Resources/src/chat/JUnitChat.java as a JUnit Test to verify the above functionality.
- Must be run on an empty SQL database.

Deployment Documentation

Required Software and Tools

1. Eclipse IDE for Java Developers

2. MySQL Workbench
3. Apache Tomcat v9.0
4. Commons-lang3-3.11.jar
5. JUnit 5
6. JRE System Library

Getting Started

To run the web application:

Step 1: Download the provided zip file and unzip.

Step 2: In Eclipse IDE create a new Dynamic Web Project and make sure the Target runtime is Apache Tomcat v9.0.

Step 3: Navigate to the newly created Dynamic Web Project Folder destination.

Step 4: Copy all files from WebContent into your own Web Project/WebContent folder. Do not delete WEB-INF or META-INF from the Web Project/WebContent folder.

Step 5: Copy and replace the src folder with those from the included unzipped file and refresh the Eclipse project explorer to see the newly added files.

Step 6: Copy the commons-lang3-3.11.jar and the mysql-connector-java-8.0.22.jar file to the Dynamic Web Project Root directory.

Step 7: Go to project Build Path > Configure Build Path > Libraries > Classpath > Add JARs > select the jar files and apply for both. (Windows users: Make sure the commons-lang jar file is in WebContent/WEB-INF/lib) (Mac users: Add both jars to /usr/local/Cellar/tomcat/9.0.37/libexec/lib)

Step 8: On classpath select Add Library > JUnit5 > Finish > Apply and Close

Step 9: In the Eclipse project navigate to Java Resources/src/models/Credentials.java.

Step 10: Edit user and pwd to contain your own MySQL login information (defaulted at “root”, “root”).

Step 11: Open MySQL Workbench and run the provided script [FinalProject.sql]

Step 12: Select WebContent/signin.jsp and run it on your local server.

Step 13: If you have signed in using Google sign-on and want to try sign in as a different user with a different account, use a different browser or try incognito on any browser other than Chrome.

To run the Office Hours Queue feature:

Step 1: Make sure your MySQL Workbench is still open and rerun the “FinalProject.sql” script.

Step 2: Open the files “OHTest.java” and “ClientThread.java” in Eclipse. Make sure to read the documentation for Office Hours Queue and the notes at the top of the “OHTest.java” file.

Step 3: Click on your Eclipse “Console”. In the top right corner, click on the “Open Console” button which should open a dropdown menu. Click on “1 New Console View”.

Step 4: In “OHTest.java”, comment out all but 1 test and click “Run As > Java application”.

Step 5: When prompted by messages in the console, run “ClientThread.java” by clicking “Run As > Java application”, while the “OHTest.java” file is running. If the console signals that the meeting invite has expired, running “ClientThread.java” will reference the next waiting student.

Step 6: Click the “Display Selected Console” button to switch one console so it shows the output of ClientThread in order to see what is sent over the socket to the Student. You can pin the ClientThread console by clicking the “Pin Console” button.

Step 7: Repeat steps 4-5 for all tests in “OHTest.java” to see the functionality of the Office Hours Queue feature. (Note: You will need to clear the database if you repeat tests to avoid creating duplicate entries).