Team Members
Adina Kruijssen, kruijsse@usc.edu
Guillermo Leal Gamboa, lealgamb@usc.edu
James Meyer, jamesmey@usc.edu
Tran Ngo, tranngo@usc.edu
Sajeev Saluja, sajeevsa@usc.edu

Deployment Document

Steps:

1. Download the project folder as a `.zip` and expand it.

2. Navigate to the directory where `pom.xml` is located (either `/` or `/sycamore-maven`)

    a. Run ``mvn package″ in the console
    b. This will generate a target directory with a `SycamoreScheduler.war` file and deployable directory (called `SycamoreScheduler`)

3. Run the `fullDB.sql` script found in the database directory script in MySQL

    a. Ensure that the `CONNECTION_PATH` variable in `src/driver/JDBCDriver.java` is correct (Line #21 at the top of the `JDBCDriver` class)

    b. Start the MySQL server

4. Run the application on the Apache Tomcat Server version 9.0 either through the `SycamoreScheduler.war` file or through the `SycamoreScheduler` folder (both are in the `target` folder under `sycamore-maven`)

5. Navigate to `http://localhost:8080/SycamoreScheduler/`

High-Level Requirements

We want to create a web application that simplifies course planning for USC students.

When entering the website, students will have the option to continue as a guest, sign into an existing profile, or create a new profile. If the user creates a new profile, they must enter their degree program(s), current year, and classes taken. Guests will not be able to input any of this information, and can simply build a course plan using the courses available on the site, and export it for offline access only.

At this point the user will be directed to the main page of the website, which shows a list of courses for that user's degree program. The user can click on any course and view more information about it. If they'd like to take that course, they can add it to their course plan. When the user is finished shopping for courses, they can choose to view their schedule, which will direct them to a new page.

When editing their schedules, users should be able to complete basic actions. These actions include adding courses, viewing courses, requesting to see and add other users' course schedules, adding users to cooperate with, blocking users from viewing or adding their schedule, and other similar actions that a user may require.

Registered users should also be able to cooperate with other users. Multiple users can edit a course schedule simultaneously if they share a major and/or minor. For example, the two users can select professors and courses they wish to take together. Based on these criterias set by the user, the web app should generate a new set of courses from which the users can choose. This should update on both of the user's screens in real-time. There will also be a chatbox on the side of the page so that the users can communicate while they create their schedules.

When users are editing their course schedule, they should have access to additional resources to guide them in the schedule-creating process, like an official course plan containing the requirements for the user's academic program. An example of this would be having the Viterbi handbook for engineering majors. Access to these resources should still be within the web interface like a small minimizable window at the bottom right of the screen.

If a user saves their schedule, the schedule will be saved in our database. Users can then access the database by logging into their account and finding their previously saved schedule on the web interface.

When generating the schedule for the user, our web app will use data previously garnered from the USC Courses website to determine which courses are mandatory (e.g. prerequisites, corequisites, etc.) for their respective academic program. Our web app will also have access to the courses that are available/offered for the upcoming terms, which will assist the user in determining which classes they might want to take in the future.

Overall, this web interface should make course scheduling for students simpler and provide them with a visual of what their schedule will look like for the upcoming academic terms. The user should be able to export this schedule to have offline access to it and add the schedule to their Google Calendar.

Technical Specifications

Web interface (8 hours)

- Start with a login page with three buttons: Login, Sign up, Enter as Guest
    - If the user clicks Sign Up, they will be redirected to a Sign Up page. The page will have a form asking for name, email, current major(s), and current minor(s). When they select their major(s)/minor(s), more fields will display asking which courses from those degree program(s) they have taken.
        - After clicking the submit page (and the information has been successfully submitted to our database), the user will be redirected to a page that asks for the courses that he or she has taken.
    - If the user clicks Login, the user will be prompted a form that asks for username and password. If the user clicks on submit, the user will either be redirected to their homepage (their credentials were valid) or the user will stay on the same page and be prompted to re-enter their username and password (the user entered incorrect credentials).
    - If the user clicks Guest, they will be prompted for their major and then taken to the normal homepage, where they can look at the classes needed and build a schedule, but will not be able to save classes taken or the schedule they build to an account. However, the save button will be omitted from the page.
- Main page: The main page will have a toolbar at the top with a "Classes" tab, "Schedule" tab, and "Profile tab". By default, the Classes tab will display.
- Classes: This page will show a list of courses for the user's degree program(s). Next to the list, a tree of classes for their program(s) will be displayed. The tree div will be tabbed, so if a user has multiple programs, there will be a different tab for each program.

Database (2 hours)

- There will be nine tables in the database: **User**, **Class**, **UserClasses**, **DegreeProgram**, **DegreeClasses**, **Department**, **Prerequisite**, **Corequisite**, and **Instructor**.
- The **User** table will be used for authentication and keeping track of what educational track they are on (namely, their major(s) and minor(s)). It will contain a user ID, the name of the user, a hashed password, the ID of their major degree(s), the ID of their minor degree(s), and their email.
- The **Class** table will be used to keep a list of classes that USC has to offer for the upcoming semester. The columns it contains are the class ID, the department, the class number, section, session, type, start time, end time, days, number of registered, maximum number of registered, instructor, location, syllabus (if present), other information (if present)
- The **UserClasses** table will be used to keep a track of any given user's classes. It will contain a user's class ID that is unique to that relationship, as well as a user ID, a class ID

- The **DegreeProgram** table allows us to keep track of the list of all the degrees possible at USC, so the only columns it contains are the degree ID and the name of the degree. This will include minors.
- The **DegreeClasses** table allows us to correlate any given class with a degree program. There may be more than one degree program per class. The columns it contains are a degree's class ID, the degree ID, and the class ID.
- The **Department** table contains a list of all the departments USC has (for example CSCI or BISC or PHIL). For that reason, the only fields it contains are the department ID and the name of the department.
- The **Prerequisite** table contains a list of prerequisites for various classes, so the fields are the prerequisite ID, the class ID, and the ID of prerequisite for the class given by the class ID.
- The **Prerequisite** table contains a list of corequisites for various classes, so the fields are the corequisite ID, the class ID, and the ID of corequisite for the class given by the class ID.
- The **Instructor** table contains a list of instructors that teach at USC. The columns it contains are the instructor ID and the link to their RateMyProfessor profile if it exists.

Backend Database (4 hours)

- Create a JDBC Driver class
- This is similar to the lab solution
- Class should be static and support:
    - Connecting
    - Closing
    - Adding users
    - Updating users
    - Updating user schedules
    - Any add, delete, and update operations deemed necessary along the way

Web Scraping (8 hours)

- Various USC websites will be scraped and their information imported into the database.
- From USC's catalogue website, catalogue.usc.edu/, the following information can be provided:
    - Degree programs and the classes required for completion of the degree.
- From USC's classes website, classes.usc.edu/, the following information can be provided:
    - Class information: name, section, session, type, start time, end time, days, number of registered, maximum number of registered, instructor, location, syllabus (if present), other information (if present), prerequisites (if present), corequisites (if present).

Backend Servlets (6 hours)

- Create a Controller package. Have a servlet for each page.
    - GET should redirect to its corresponding JSP
    - POST should be used to submit data to the server
- RegisterServlet:
    - Verify user input is valid.
        - No empty fields, minimum length met, special characters, etc.
        - No duplicate usernames and emails
    - If data is invalid:
        - Send error message
        - Do not redirect to <Home>
    - If all data is valid:
        - Create a new User
            - Add new User to the database
            - Add user to session
            - Redirect to Classes page
- LoginServlet:
    - Verify user input is valid.
        - No empty fields, minimum length met, special characters, etc.
        - Authenticate username and password with database entries
    - If user is authenticated:
        - Add user to session
        - Redirect to Classes page
- LogoutServlet:
    - If user has unsaved changes, give option to save
        - If yes, update the database
- ClassesServlet:
    - GET should get a list of all the classes for the user's program and redirect to the classes JSP
- ScheduleServlet:
    - GET should get the user's saved schedule from the database and redirect to the schedule JSP
    - POST should update the user's schedule and the changes should be reflected in the database
        - This can include adding or removing a class
- ProfileServlet:
    - GET should get the user's first name, last name, schedule (if public) and redirect to the profile JSP

        - POST should update an personal changes the user has made and the changes should be reflected in the database
            - This can include changing the user's email, program, etc.
    - On login page, we need to be able to pass in user-inputted username and password to a servlet in order to verify credentials.

Multithreading (8 hours):

- We need to have a minimizable window that serves as a guide for the user when choosing their courses (e.g. official course plans).
    - We can create a new thread for this window. The window will essentially be a mini-browser that runs at the same as the main program.
    - The user should be able to scroll through this window and click on links that redirect them to relevant course resources.

*The multi-threading requirement is now going to be fulfilled by adding the feature of users being able to follow another user's schedule and being able to view and edit it together

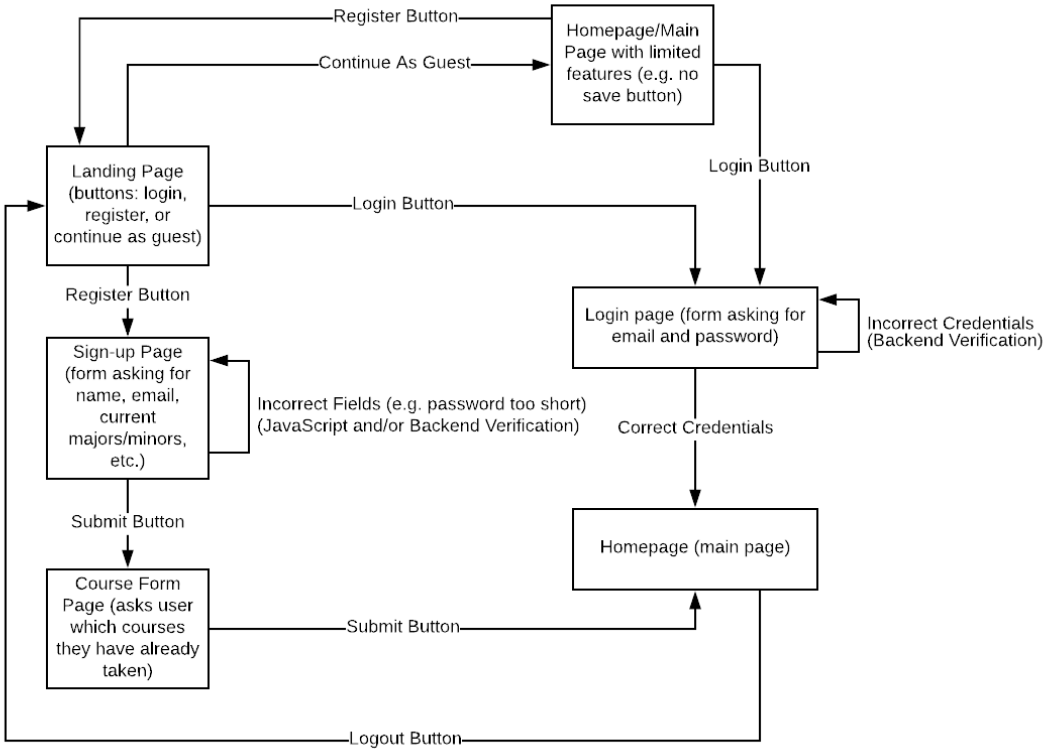
Networking (8 hours)

- GoogleCalendarServlet:
    - GET should check if changes have been made to the schedule through Google Calendar
        - If there have been changes, update the database accordingly
    - POST should communicate with the user's Google calendar
        - Should handle adding, updating, and deleting events
        - These changes should be reflected in the database as well
- The web app will work with Google Calendar so that the user can sync their schedules.
- We will also implement Google's Sign-in API to make this process easier.

Detailed Design

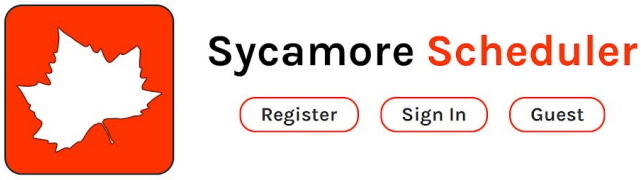
All of the specifications in the Technical Specifications document are designed out

Site Map:

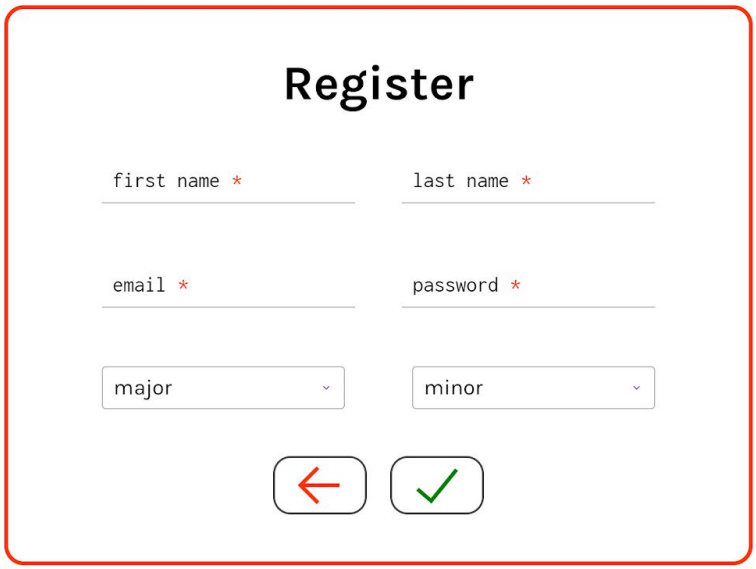


Note: After reaching the homepage/main page, buttons will no longer redirect the user to another page (i.e. the homepage will be one dynamic webpage). The only page redirection from the homepage is the logout button which will redirect the user back to the registration/login page.

GUI Mockup (refer to site map above for details on how to get from one page to another):
UPDATE: The views that are complete have been updated. The rest are almost complete and will be updated with final documentation

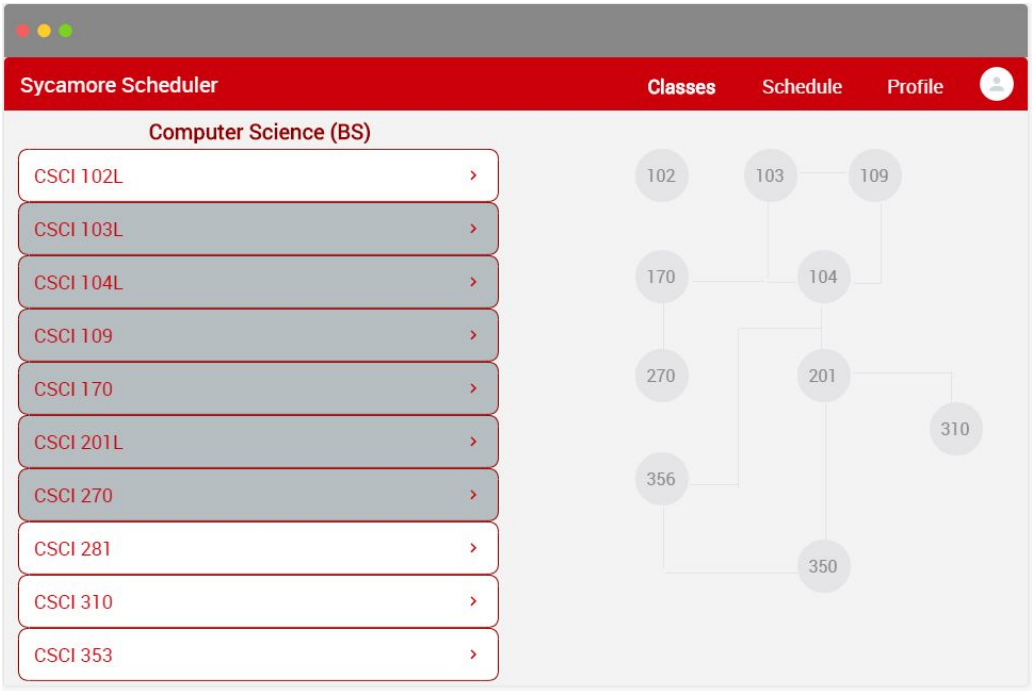


Landing page

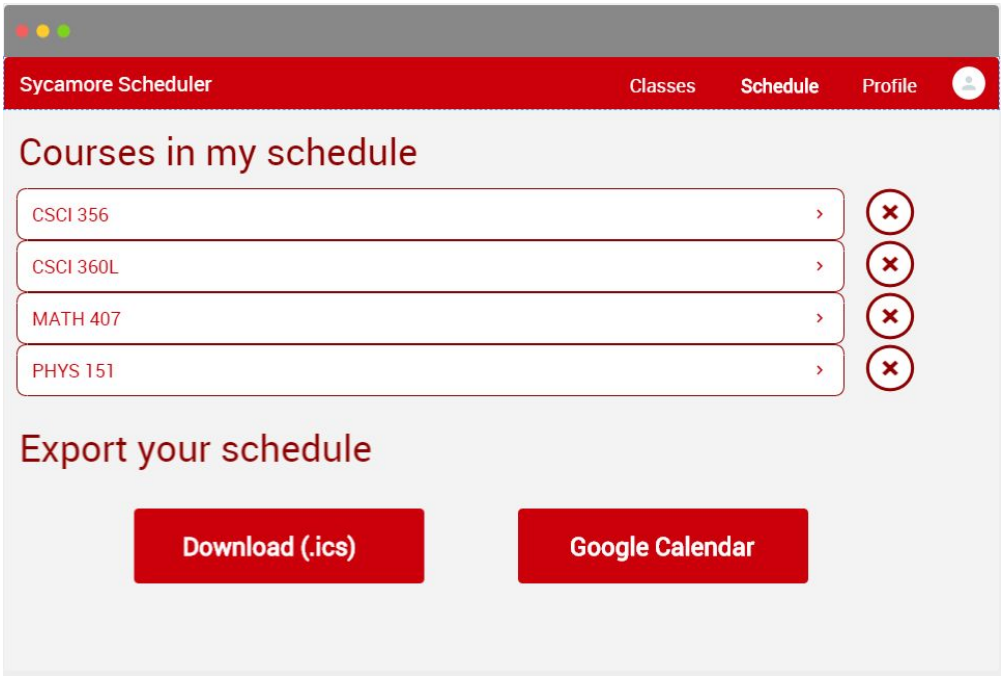


Sign up page

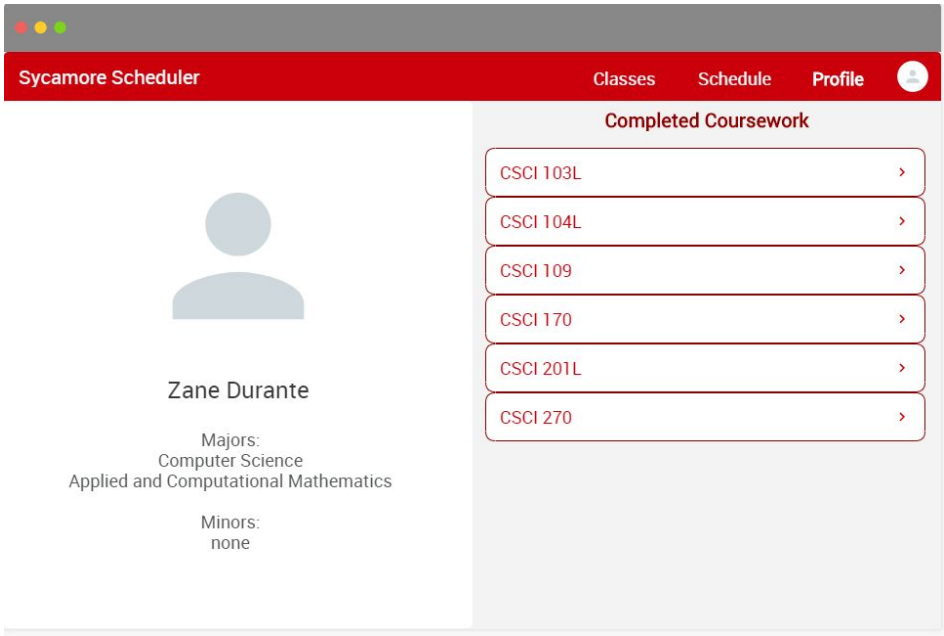
Login page



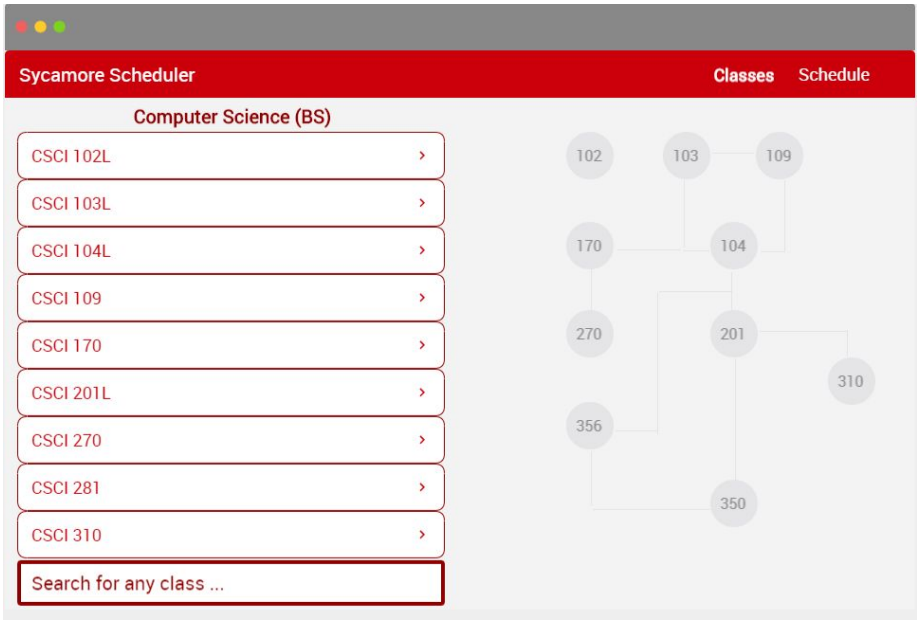
Home page (Classes tab) for signed in users



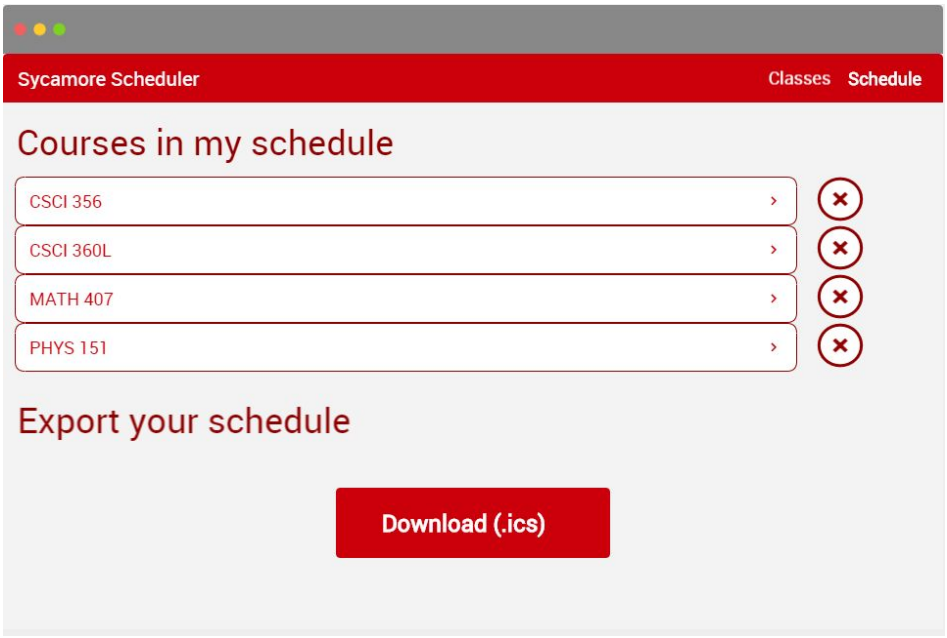
Home page (Schedule tab) for signed in users



Home page (Profile tab) for signed in users

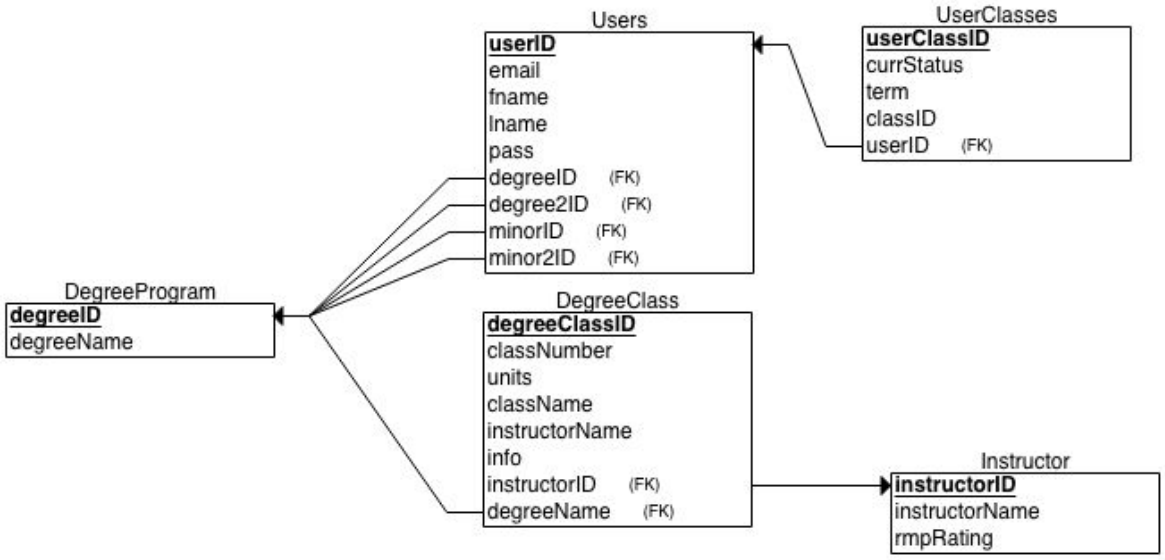


Home page (Classes tab) **for guest users only**



Home page (Schedule tab) **for guest users only**

Database Schema:



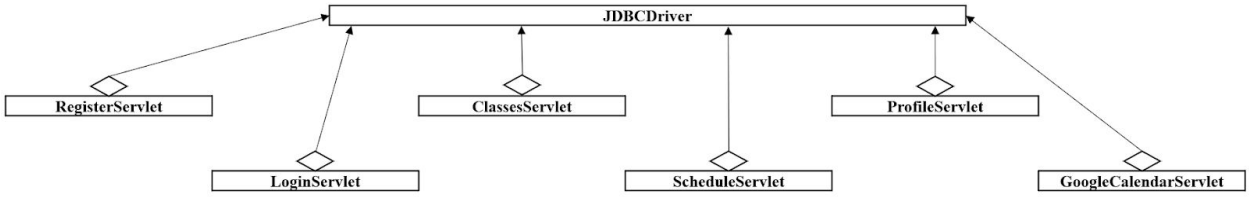
*Add degreeName to UserClasses table. This will simplify the JDBCDriver.getSchedule() method so only classes for a specific degree program are returned.
**Add following table to keep track of which users can access another user's schedule. There will be a a user id as the primary key, and then a column for another user, and a column that is 1 if the other user can access the first user's schedule, 0 if they cannot (this is similar to Assignment 3)

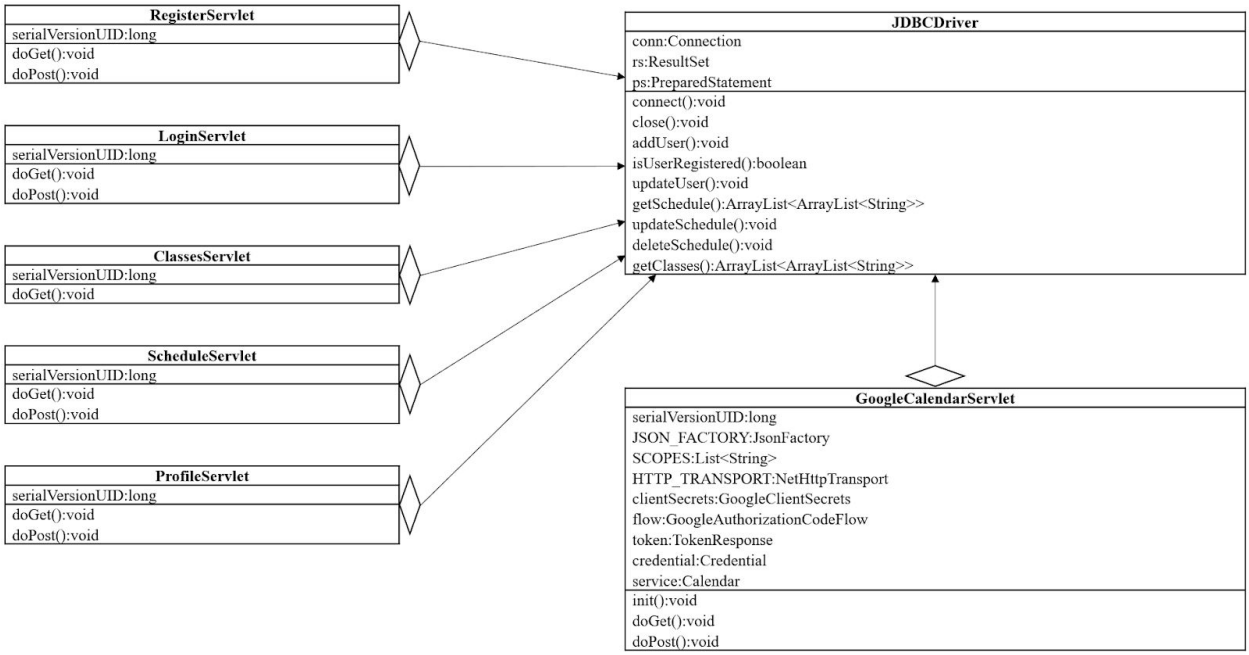
Hardware and Software Requirements:

- Hardware: Nothing Specific
- Tomcat Server
- Java and Java Servlets
- Google Calendar API
- GSON, Boon, and/or Jackson API
- MySQL JDBC Driver 5
- Python (Scraping)
- JavaScript (ReactJS, JQuery)
- HTML, CSS (Bootstrap, possibly Sass)

Class Diagram and/or Inheritance Hierarchy:

- Package: driver
    - JDBCDriver
- Package: controller
    - RegisterServlet
    - LoginServlet
    - LogoutServlet
    - ClassesServlet
    - ScheduleServlet
    - ProfileServlet
    - GoogleCalendarServlet





*The following classes have been updated to return a boolean, true if information updated correctly, and false otherwise: addUser(), updateUser(), updateSchedule(), deleteSchedule(), **The RegisterServlet.doGet() method is not necessary, only the RegisterServlet.doPost() method is necessary for registering requests.The LoginServlet.doGet() method is no longer necessary. Only the LoginServlet.doPost() method is necessary for users logging in.

LogoutServlet
serialVersionUID:long
doPost():void

Server/Client/Multithreading Classes:

Package: Server

ServerThread extends Thread
ois: ObjectInputStream, oos: ObjectOutputStream, sch: Schedule
sendSelection(Selection sel): void, run(): void

Schedule
serverThreads: Vector<ServerThread>
broadcast(Selection sel, ServerThread st): void, main(String [] args): void

Package: data

Selection implements Serializable
serialVersionUID: long, username: String, selection: String
getUsername(): void, getSelection(): void

Package: Client

Client extends Thread
ois: ObjectInputStream, oos: ObjectOutputStream
run(): void, main(String [] args): void

Package: Socket

ServerSocket
sessionVector: Vector<Session>
open(Session session): void, onSelection(String selection, Session session): void, close(Session session): void, error(Throwable error): void

Key Algorithms:

- The JDBCDriver class will be responsible for accessing a MySQL database. This class will handle a lot of the operations that the program requires
    - Because of this, most classes except for the LogoutServlet class will require access to this class
- The RegisterServlet class will access the Users table in a MySQL database using the JDBCDriver class, which uses the jdbc:mysqldriver
    - The boolean addUser() method will take five parameters:
        - four strings representing the email, first name, last name, and password
        - one ArrayList<String> representing the academic programs the user is enrolled in
        - returns true if a user is successfully added to the database and false otherwise.
- The LoginServlet class will access the Users table in a MySQL database using the JDBCDriver class, which uses the jdbc:mysqldriver
    - The isUserRegistered() method will take two strings as parameters, representing the email and password
        - It will return a boolean value specifying whether the user was authenticated or not against the Users table
- The LogoutServlet class will
    - Sign the user out and redirect to the home page
- The ClassesServlet class will access the Class table in a MySQL database using the JDBCDriver class, which uses the jdbc:mysqldriver
    - The getClasses() method will take one string as a parameter, representing the department name
        - It will return an ArrayList<ArrayList<String>> representing the information for the department's classes
- The ScheduleServlet class will access the UserClasses table in a MySQL database using the JDBCDriver class, which uses the jdbc:mysqldriver
    - The getSchedule() method will take two strings as parameters, representing the email and degree program name
        - It will return an ArrayList<ArrayList<String>> representing the user's schedule
    - The ScheduleServlet will also handle updates to the student's schedule.
        - The front-end will pass an action (a string variable) either being "add" or "remove"
            - If the action is add, we call on the JDBCDriver's addClassToSchedule() method which takes three strings as parameters, representing the email, class name, and degree program name

            - If the action is remove, we call on the JDBCDriver's removeClassFromSchedule() method which takes three strings as parameter, representing the email, class name, and degree program name
- The ProfileServlet class will access the Users and UserClasses table in a MySQL database using the JDBCDriver class, which uses the jdbc:mysqldriver
    - The getSchedule() method will take two strings as parameters, representing the email and degree program name
        - It will return an ArrayList<ArrayList<String>> representing the user's schedule
    - The boolean updateUser() method will take two parameters
        - One string representing the email
        - One Map<String, String> representing the key-value pairs of the information to update
        - returns true if a user's information is successfully updated on the database and false otherwise.
- The GoogleCalendarServlet class access the user's Google Calendar using the Google Calendar API and it will access the UserClasses table in a MySQL database using the JDBCDriver class, which uses the jdbc:mysqldriver
    - The getSchedule() method will take two strings as parameters, representing the email and degree program name
        - It will return an ArrayList<ArrayList<String>> representing the user's schedule
    - The class will then add this information to the user's Google Calendar

In addition to the deployment document, submit your requirements, specifications, detailed design document, testing document/code, and updated code with any changes clearly delineated using a feature such as Track Changes.

**update database tables, servlet classes

- The RegisterServlet.doGet() method is now implemented to retrieve all possible degree programs when a user is trying to register.
- In general, all servlets now follow these guidelines:
    - Set the HttpServletResponse status.
    - Get the PrintWriter and write the success/error message or send the data from the database.
        - This information is sent as a JSON.