CSCI 201 (Spring 2022) Final Project Submission:

Aadesh Bajaj, Nick Chu, Slater Gable, Aditya Hariharan, Paul Kim, Matthew Quan
aadeshba@usc.edu, nhchu@usc.edu, smgable@usc.edu, adityaha@usc.edu, pbkim@usc.edu, maquan@usc.edu

Chore Board Project Proposal

We are aiming to create a user interface in which roommates will be able to coordinate the completion of chores. This paper is an outline of the many features which we plan to include.

The login system will be crucial for the Chore Board in order to share tasks with roommates. Using the website without logging in will function like an offline calendar, allowing you to continue as a guest and add tasks. After logging in, we intend to have a task view of projects for the group members to see. This will display all the information regarding the task (like who assigned it, who it is assigned to, the name of it, the description, and the difficulty level as well). These tasks can be updated, viewed, and deleted pretty easily from the task view so that there is easy access to the chores necessary.

Furthermore, we want our project to also update in real time with each user input. Once they have joined, they should have full functionality available to a group.

In conclusion, our project's purpose is to serve as a task list so that users/students can be able to put their names down, and indicate what tasks/chores they need to complete. We want to visibly indicate preferences/choices of tasks in a nice, simple, and clean manner.

Technical Specifications

HomePage.jsx

The main purpose for HomePage.jsx is to serve as the hub for all of our program's functionality. It shows the task view containing all of the group's tasks through api endpoints to the backend which makes sql calls to the database through JPARepository methods. Users are able to edit/create/delete tasks through the "Edit Task" button, interacting through a dispatcher (TaskDispatcher) to modify the SQL database. Finally, links that can logout, redirect back to home, or go to the global chat feature will be displayed up above in the Navbar.

Create/View/Edit/Delete tasks in Home.jsx linked to ListTaskComponent.jsx and CreateTaskComponent.jsx:

There will be a button on each individual task in HomePage.jsx that will allow the user to edit, view, or delete a task. Deleting a task will remove the task from the table and only show the remaining tasks, but editing or viewing a task will bring the user to a new page, CreateTaskComponent.jsx. Similarly, HomePage will also contain a create new task button that brings the user to an empty Task.jsp document as well. On this page there will be details about each task that the user can input (outlined in a visualization on “Create/View/Edit/Delete Task, Task.jsp PDF”). These inputs will be stored in an SQL database unique to an individual task. The save button will upload the contents to the database and the cancel button will cancel the query.

New Tech Stacks (6+):

1. React
2. Chat Feature (Multi-Threading, Networking), Firebase
3. SpringBoot (Backend Feature)
    a. Spring Security
    b. JPARepository
    c. Layered model of backend frameworks
    d. Spring framework
4. Bootstrap
5. Figma - Design UI/UX
6. Git - resource management and control
7. Node

New Software (6+):

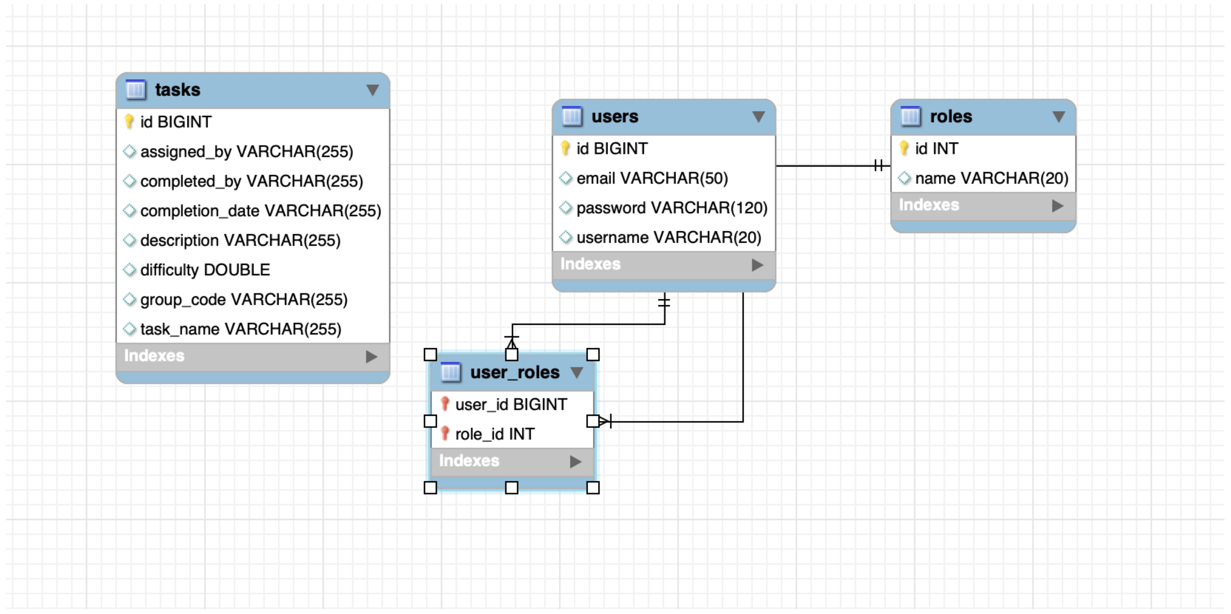
1. Eclipse
2. Visual Studio
3. MySQL
4. React
5. SpringBoot
6. Servlets - used within SpringBoot
7. Javascript
8. Java
9. JDBC - used within SpringBoot
10. Github
11. Firebase
12. Tomcat - used by SpringBoot

Design Document

Data Structures:

The data structures present revolve around Spring Boot’s revolutionary way of populating the database. There is a controller, model, and view framework that is key to implementing any REST API (which is what is used to communicate between the front end and the back end). The models are responsible for creating the structures within the database (User and Task). The controllers create API endpoints and serve as the communication to the front-end (the view part of the framework), while the repositories link to the database and start queries. The Task data structure specifically involves a few data features: the completion date (or due date), who it is assigned to, who it is assigned by, the difficulty, the name, and the description. For each task, we also want to create a task object that can store some of these required attributes that we want to be able to pull when performing SQL queries to populate the calendar from the database. Also, anytime a task is created, we want to populate the database with the information from the created task object. In addition to these structures, we also need to have an object that stores users, most likely in some sort of array list (we would likely have a user class, which will be a model in SpringBoot).

Database Design:



Our backend design focuses on four main tables. The “Users” tables will be the primary table that contains all of the users information including the user’s (username, email, password). This will be used for our Login services. The tasks table will be used to store the data for tasks, which was described above. Essentially, it stores Task objects using typical database formatting.

Finally, the roles table is used to give users specific access to things. For example, the Home page is only accessible if logged in (or attempting to guest login). Clicking the Home button without logging in should redirect the user to the log in page, and this is mandated by giving a logged in user the role "ROLES_USER." The join table allows us to quickly associate roles and users.

6+ Pieces of Software:

The software we intend to use in this project mostly consists of technologies not common to the CSCI201 curriculum, though the curriculum plays a big role in informing our process. The infrastructure of the project will be coded in Java, JavaScript, and React (jsx files, which allow us to code in HTML and CSS with javascript as well). We will develop within the Eclipse integrated development environment. Apache Tomcat v9.0 will be used as a server environment to test and soft-launch our application. In order to store user and application data, we will use MySQL as our database system. To maintain version control of our application, we will use Git through Github. We will also use React to make a more interactive user interface in our frontend. In addition, Spring Boot in our backend framework will be used to easily "run" our web application whenever needed. Spring Security

React:

React is a JavaScript library used for building user interfaces. We intend to use it in order to streamline our front-end process to make an interactive, dynamic, and responsive user interface. In addition, by using React, we will be adding a new tool to our tech stack.

Spring Boot:

Sprint Boot is a platform that makes it easy to create stand-alone, production-grade Applications that you can "Just Run." It has embedded Tomcat, starter dependencies and 3rd party libraries whenever possible. It is a backend framework major in the Java ecosystem, that lets Java developers start building web applications quickly, without fuss. The platform can also be used alongside frontend frameworks like React, which is what we will be using. The platform works by automatically configuring the application based on the dependencies you have added to the project. For example, if our MySQL database is on our classpath, but we have not configured any database connection, then Spring Boot auto-configures an in-memory database. In sum, by using the Spring Boot platform, it takes out all the pains related to dependency management, configuration, bean declaration, etc.

Chat Feature (Firebase):

Another possible consideration for a feature is an internal chat box feature for users in the same group. Clients (users) would be able to post their messages onto the server using the HTTP Post method. We may want to use a sort of timer feature (threading would be perfect for this) where the HTTP Get request for other messages from other clients would be essentially blocked until the server is given something (another message) to return. We could easily style this messenger using Figma or other online sources (or just plain CSS/HTML/React). It would be a very useful feature for users in the same group who may feel it necessary to further their communication on an application that directly relates to the chores they have to do (most likely an apartment group).

Bootstrap:

Bootstrap is a front-end open source toolkit that will allow us to easily build and customize a responsive webapp, and allow it to be mobile/tablet friendly. By using bootstrap, we will be able to layout our website more easily using containers and the responsive grid system that it provides. It will also allow us to create and customize a responsive navbar. Bootstrap is one of the most popular front-end frameworks, and we believe that by using this software it will allow our code to have more consistency, and have more customizable options for our design.

Figma:

Figma is a web-based graphics editor that will allow us to create web prototypes. We will use figma to create the initial protodatabtypes of all pages of the webapp design. It will allow us to have a blueprint of what we want to show in our frontend, and make it easier to code CSS/HTML for each webpage. We believe figma will be the best choice in designing our website due to its UI/UX designs, and ability for multiple members to work on a design at once.

Testing Document

Nav Bar

Test Step	Input	Expected Output
Clicking Logo	none (button)	If logged in, home page displayed. If not logged in, redirects to the login page.
Clicking Home	none (button)	If logged in, home page displayed. If not logged in, redirects to the login page.
Clicking Logout	none (button)	Logs user out
Clicking Login	none (button)	Takes user to login page

Login Page

Test Step	Input	Expected Output
Normal User Login	User Email Matching User PW	Home page displayed with Logout, [username]
Clicking guest login	none	Group’s Home Page is displayed
Create New Account	None	Register Page is displayed
Bad User Login	Entering unregistered credentials	Gives error message back

HomePage

Test Step	Input	Expected Output
Clicking Delete Task	none (button)	Deletes the task from the table
Clicking View Task	none (button)	takes user to Task Edit page
Clicking Update Task	none (button)	takes user to Task Edit Page

Clicking add task	None (button)	Takes user to Add Task page (aka taskCreate)

TaskCreate Testing:

Test Step	**Input**	**Expected Output**
Clicking Add Task on the home page	None (Button)	TaskCreate/Edit page displayed
Click on “save”	Title, Assigned By, Assigned To, Due Date, Description, Stars (Difficulty)	All edits/create on the page will be saved and pushed to the database. Then you will be redirected to the home page, where it will be displayed in table format
Click on “cancel”	None (Button)	All inputs on the task will be canceled. If the task was already created, it will revert to what it was before.

Register Page Testing:

Test Step	**Input**	**Expected Output**
Registering User	Include Username, Email, Password,	Should successfully be able to register a user. Username should be between 3 and 20 characters, password

		between 6 and 40 characters
Email Validation	Include Email	Emails should have an @ symbol and have a period as well. Errors will be displayed if emails are clearly incorrectly formatted
Create Account Button	Click Create Account Button	Should be successfully be able to create account. If any fields are missing or misinputted, error messages will be displayed instead of successful registration

Deployment Document: Task Board

In order to deploy the application, follow the instructions in the video. You can also follow the text instructions below:

1. In a fresh Terminal, navigate to the folder where you want to store the project. Copy the SSH Code from https://github.com/201FinalProject/CSCI201FinalProject and clone the repository. Alternatively, open the ZIP.
2. Open the TaskBoard.sql file in MySQL, located inside the root directory of the project. Run the SQL server, then run the given script. This will create the database for the project.
3. Navigate to a fresh workspace in Eclipse. Select Import Projects > Maven > Existing Maven Projects > Next. For the "Root Directory," pick the folder titled

"spring-boot-spring-security-jwt-authentication-master" inside the repository folder "Full-Stack-Pt.2." Click Finish.

4. Under src/main/java, inside the package "spring.backend," run SpringBootSecurityJwtApplication.java as a Java Application. This launches our back-end framework which we will use to connect our front-end to SQL.
    a. If your username and password are not "root" and "root," enter your MySQL credentials in the application.properties file under src/main/resources.
5. After running the Spring framework in step 4, open up TaskBoard2.sql. Run that SQL script. It is important to execute steps 2-5 in this particular order.
6. Install Node.js from https://nodejs.org/en/download/ . Follow the instructions given by the installer.
7. Now it's time to launch the application itself. Open a fresh Terminal. Navigate to the folder CSCI201FinalProject/Full-Stack/react-jwt-auth-master. Once in the react-jwt-auth-master folder, enter the command "npm install." After the installation is complete, type the command "npm start."
    a. If you want to change the port the application runs on, in the file CSCI201FinalProject/Full-Stack-Pt.2/react-jwt-auth-master/.env, the port can be specified.
    b. Note: the Spring application must be run on 8080. Use npx kill-port if something else is running on this port.