JIRA Task Management System

Tiruchunapalli Venkata Abhishek

Software Engineering Systems, Northeastern University

tiruchunapalli.a@northeastern.edu

***Abstract*—**

**The aim of the project was to build a task management system that could aid in developing a large-scale project. The software allows you to build new tasks, assign tasks to users, set due dates on tasks, and track the state of tasks from beginning to end. The users also have the ability to add comments to each task, The project also has an activity management for each task which records all the activities performed on a task and maintains them independently. The software and hardware needed are readily available and simple to use. The Home page of the application gives you a good understanding of what task each customer is assigned to and the state in which these tasks are. This would aid in knowing if you would want to assign more tasks to this user and also the progress he has made so far in the tasks the user is assigned.**

**This application was developed using Spring Tool Suite with Java 11, Spring Boot, and Hibernate JPA. MVC Framework was used throughout the project wherein each action is performed with our model carrying the data, Controller holding the business logic, and view as our front end pages.**

***Keywords— SpringBoot, MVC Framework, JAVA, Hibernate***

# **I. Problem Description**

Creating a Task Management System for users wherein each task is assigned to one user, and multiple tasks can be assigned to one user. The task starts in an ‘open’ state and keeps changing hands with each step being recorded in the activity of our task and finally reaches a ‘closed’ state. All updates on the task can be added to the comment section which mentions which user was responsible for each comment. The description can also be updated by the users, the only thing immutable is the creation date of these tasks.

The main objective here is to have simple, user-friendly task management that could track tasks and keep updating them based on user requirements.

# **II. Analysis (Related Work)**

This project was heavily inspired by JIRA a user-based task management system used in several multi-national technology firms in task handling and task management. JIRA is a lot more profound with options to clone existing tasks, create a dynamic link of other tasks by adding it in comments, and tagging users in comments with ‘@’. Several of which I plan on adding to my existing project with time. This project was developed by Atlassian and is a very successful task management tool.

# **III. System Design**

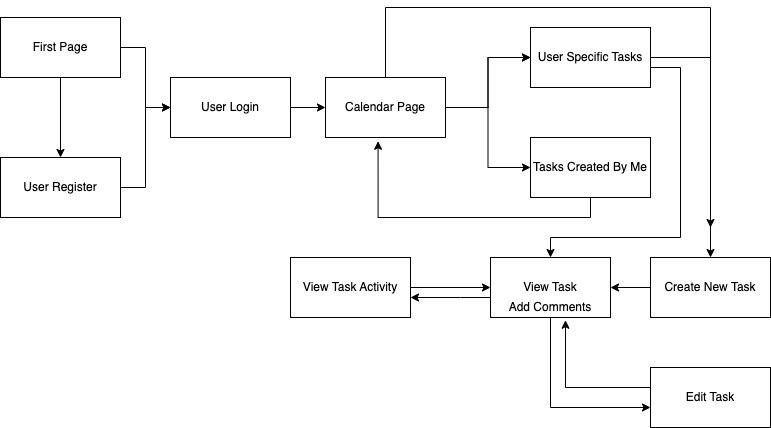


Figure 1. System Architecture

# **IV. Implementation**

*Brief Overview*

Model View Controller or MVC is a software design pattern for developing web applications.

A Model View Controller pattern is made up of the following three parts:-

Model- The lowest level of the pattern which is responsible for maintaining data.

View- This is responsible for displaying all or a portion of the data to the user.

Controller- Software   Code that holds the business logic of the project.

MVC is popular as it isolates the application logic from the user in the interface layer and supports the separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the view. The View then uses the data prepared by the Controller to generate a final presentable response.

The Business Logic:

*A. Registration/Login*

The login and registration buttons are located on the application's landing page. New users must first register, and on submit, the button on the registration form prompts the add-user.htm on Login Controller which calls the UserDAO object and adds a new row in the User table in our MySQL However, users once registered can directly access the login page, this runs a validate function on the UserDAO class to verify the user email and password match the ones in the database. Both the registration and login pages have a form validator attached to them. The registration validator has an email validator that matches the email with a regular expression and only allows emails that pass this test to be added as Users to the database.

The login page verifies you do not log in without entering any data in the form columns using the Login validator. In the login function, we pass email\_id and password as parameters to the HQL query and not as strings to prevent hackers from using tricks that would break the application. We add an object of type User with the name ‘current-user’ to the session and use it to verify if a user is logged in, on each of our JSP pages.

*B. User Home/ Post Login*

Post login in you are shown your current user information with the option to either add new tasks or view all the tasks i.e (Show Calendar). On clicking ‘Show Calendar’ a get request is sent to ‘showcalendar.htm’ this results in building a User list that would store all user information in a list. Using a linked hashmap we iterate through each user and map each User to the list of tasks they are assigned to. The tasks are retrieved in the TaskDAO class by passing the user email as a parameter.

The LinkedHashMap always returns data on the page the way they way were added to the User table. i.e Legacy users are shown first and new users go down the list. The LinkedHashMap of type <User, List<Task>> and the JSTL forEach tags help in displaying this information on our calendar page.

The Show calendar page has several hyperlinks a group of which point to several users that we have registered and also a ‘Created By Me’ link that points to the tasks created by the logged-in users. On Clicking the hyperlink of each username we would be redirected to the page only showing tasks the user we clicked on is assigned to. On clicking the same hyperlink we are redirected to the home page where the tasks of all customers are visible again.

The user is only able to view/edit or view the activity of the task only after he is logged in. The user can see the home page and tasks briefly but clicking on the hyperlink of individual tasks would prompt an error page with an option to redirect to the user login page. Each task is color coded on the state line which shows the status of the task from light-yellow to dark-red signifying how far from completing the task is. If the user is logged in he has all the permissions necessary to view/edit the task even though he is not assigned to it.

*C. Add new tasks*

Adding a new task to the list is a very simple process. On clicking the add tasks hyperlink placed in multiple locations in the application you are prompted to a form that has 4 string inputs two inputs to choose from and another is the due date. The create date of the task is inherently assigned with the current date value. All the inputs shown here are not static, on calling the ‘add-tasks.htm’ all the users are added to the request scope and shown as a list to choose from on our JSP page. This page is also associated with a validator which will keep reloading unless each row is filled with appropriate data.

D. *View/edit tasks*

The added task redirects you to the view task page which is attached with an Id that signifies the primary key with which the task is associated. We need to use an annotation called @PathVariable every time we would like to use this variable in the given context in each of our pages where it is required.

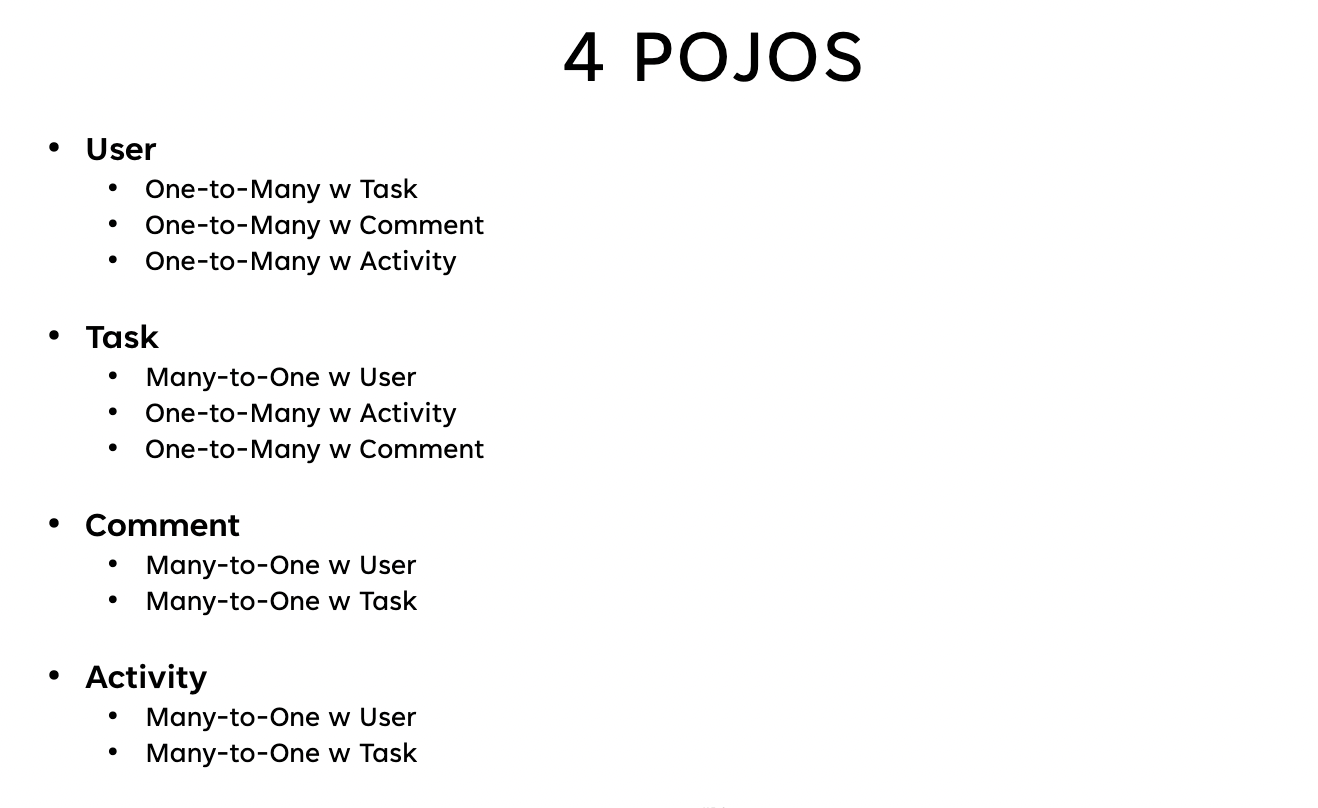
Here you have the option to edit your existing tasks in two ways. One to only update the task assignment, or two to click on the link and make changes to each row of the task. The update task assignment can happen directly on the “viewtask/id” page whereas other changes are done with the help of a form validator on our update task page. The user also has the ability to add comments to the page, every time comments are added they are appended by a ‘By:’ that adds the name of the user who has added the comment to the task.

E. *View Activity*

While working on tasks it is often common to make changes that you would regret later or want to revert back to or to check the changes made as a reference for a similar task that has come up. For all these tasks activity plays a really good role. The activity bar of the task shows all the actions and edits performed along with the user who made the change and also the time at which the changes were made. This acts as a tool for accountability and also preserves task history from its creation.

F. *POJO Classes*

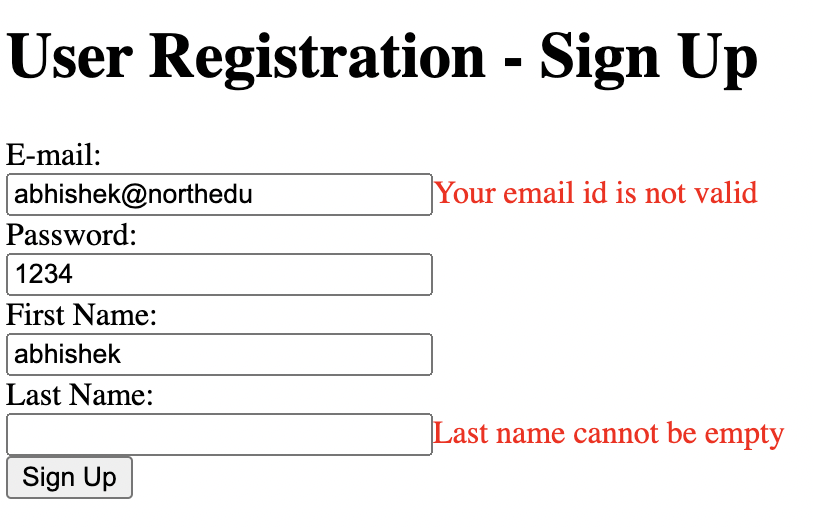
The page uses four POJO classes to make sure this implementation is achieved. The four classes are represented below:



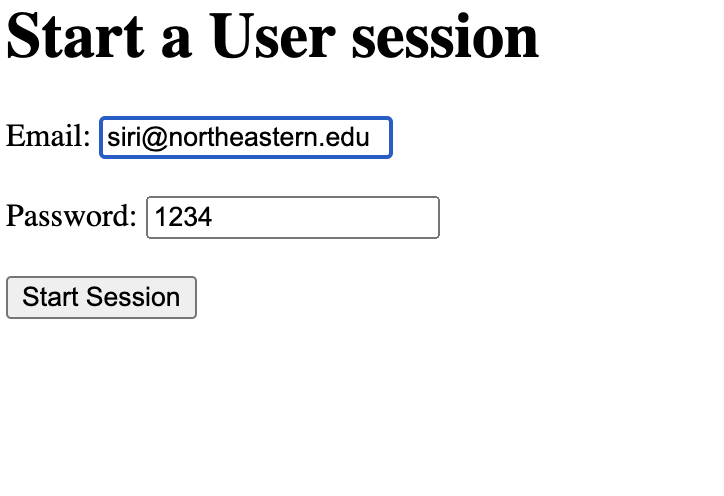
*E. Connection between tables*

The User table does not have any of the foreign keys in its own table as it acts as the owning entity for each of these relationships. The task table has one column ‘user\_id’ which helps in mapping each task to its appropriate user. The comment table has two associations with User and Task. It has two foreign keys one identifying the user who added the comment and the other the task to which the comment was added. A combination of (1,4) would mean a comment added by the user(One) to task(Four). The activity table has a very similar association with the User and Task tables. A combination of (1,4) would mean the activity was done by User(One) to the task(Four).

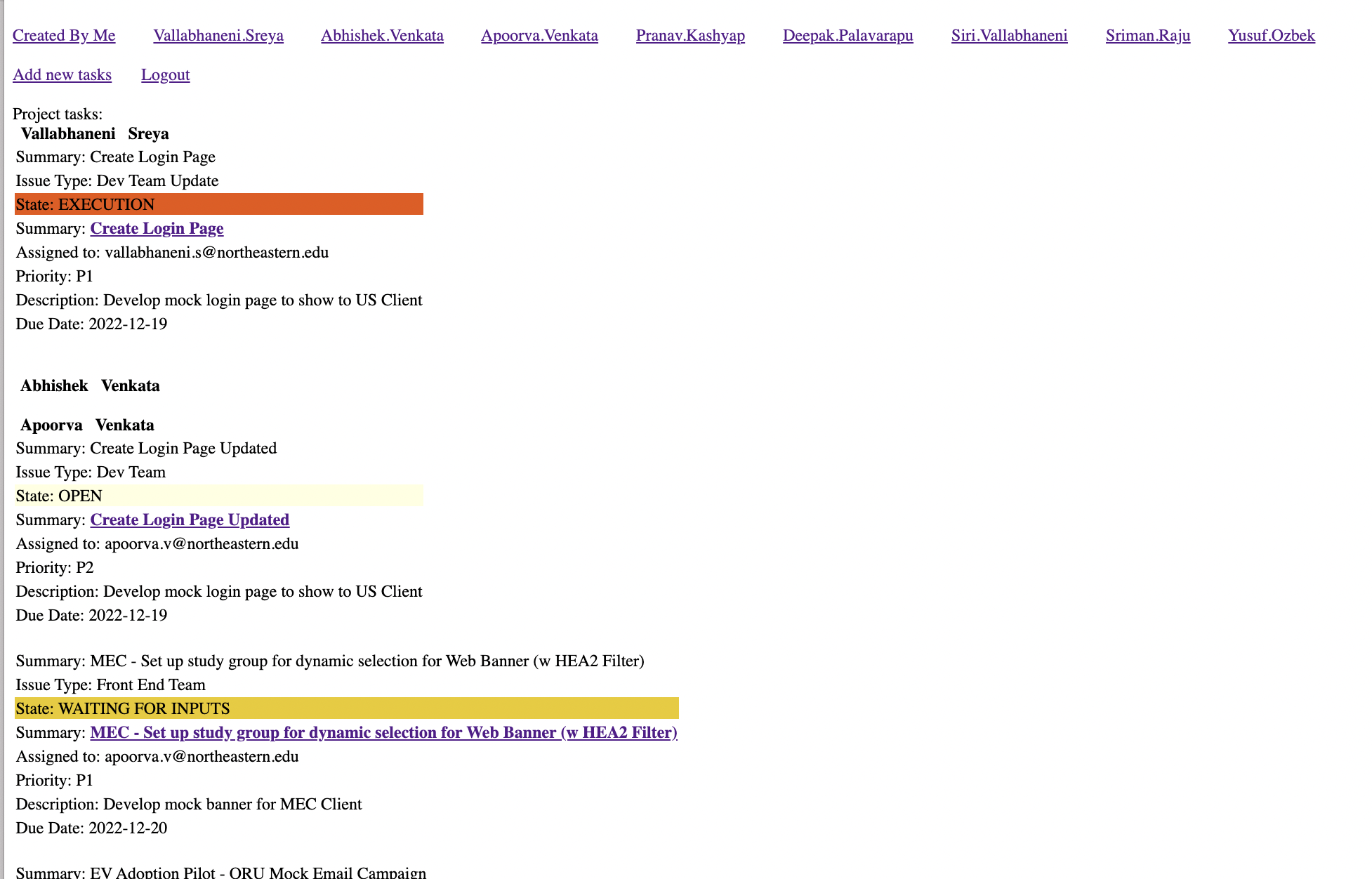
# **V. Evaluation**



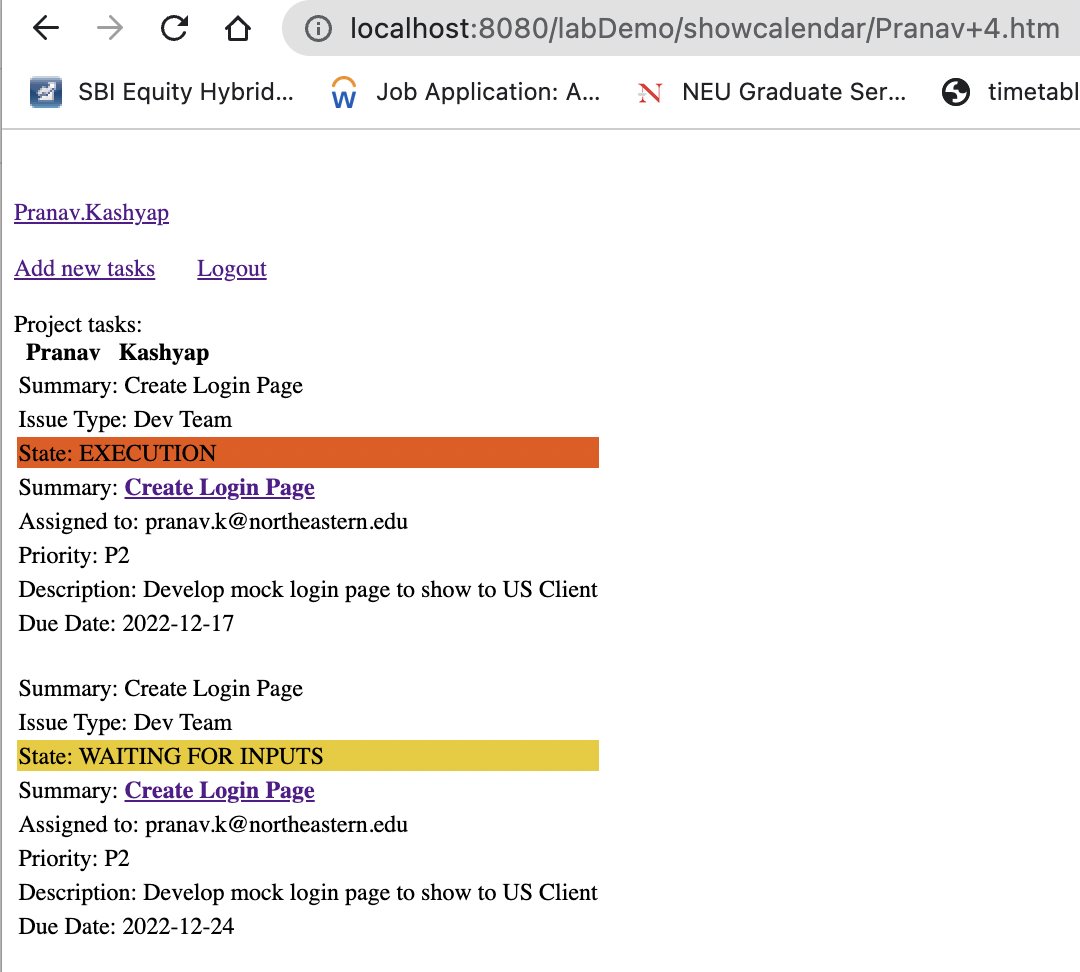
1. Registration page



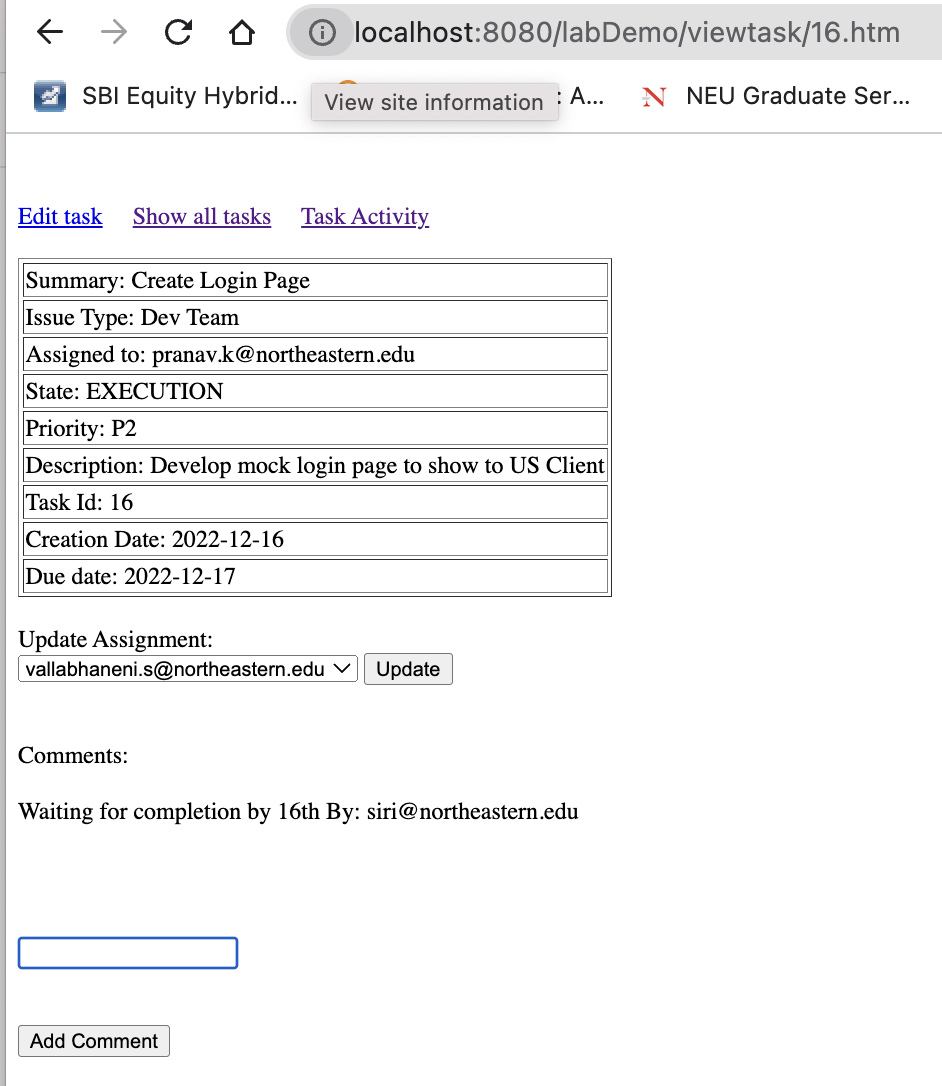
1. Login form



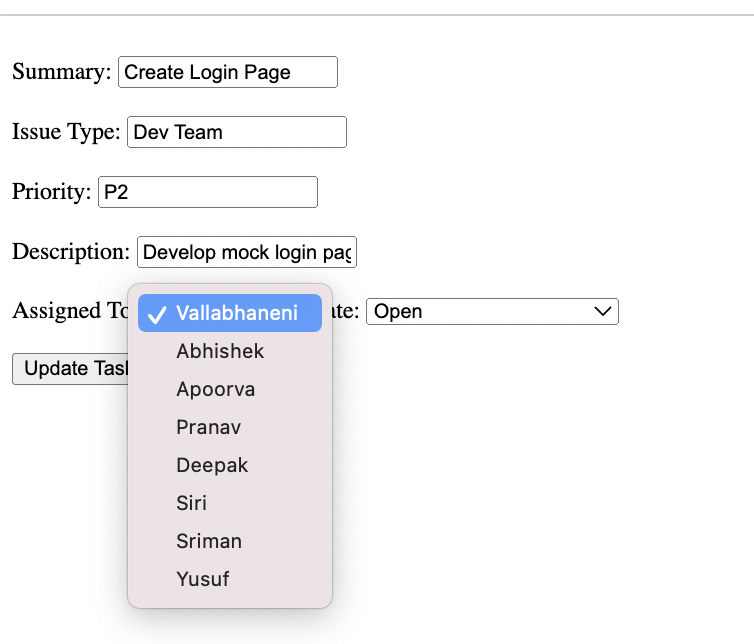
1. Home/Calendar



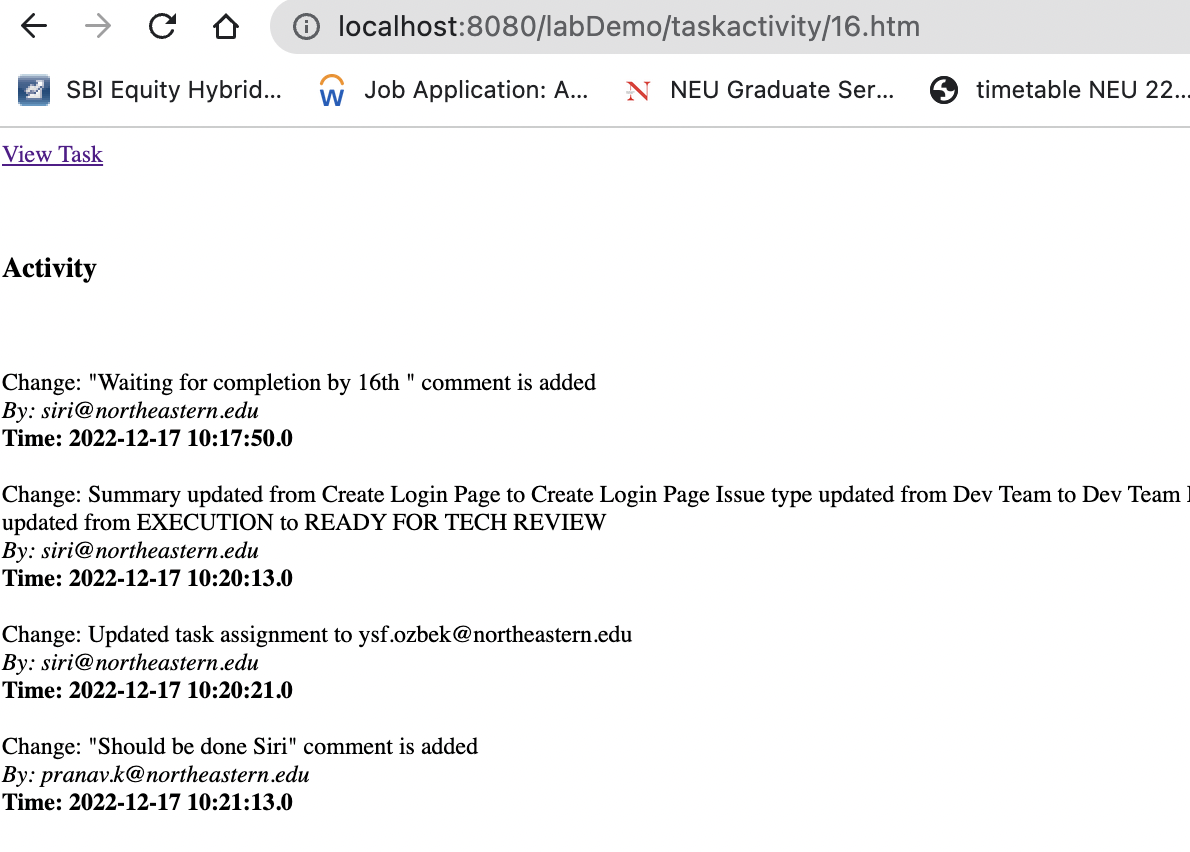
1. User-Specific Home Page



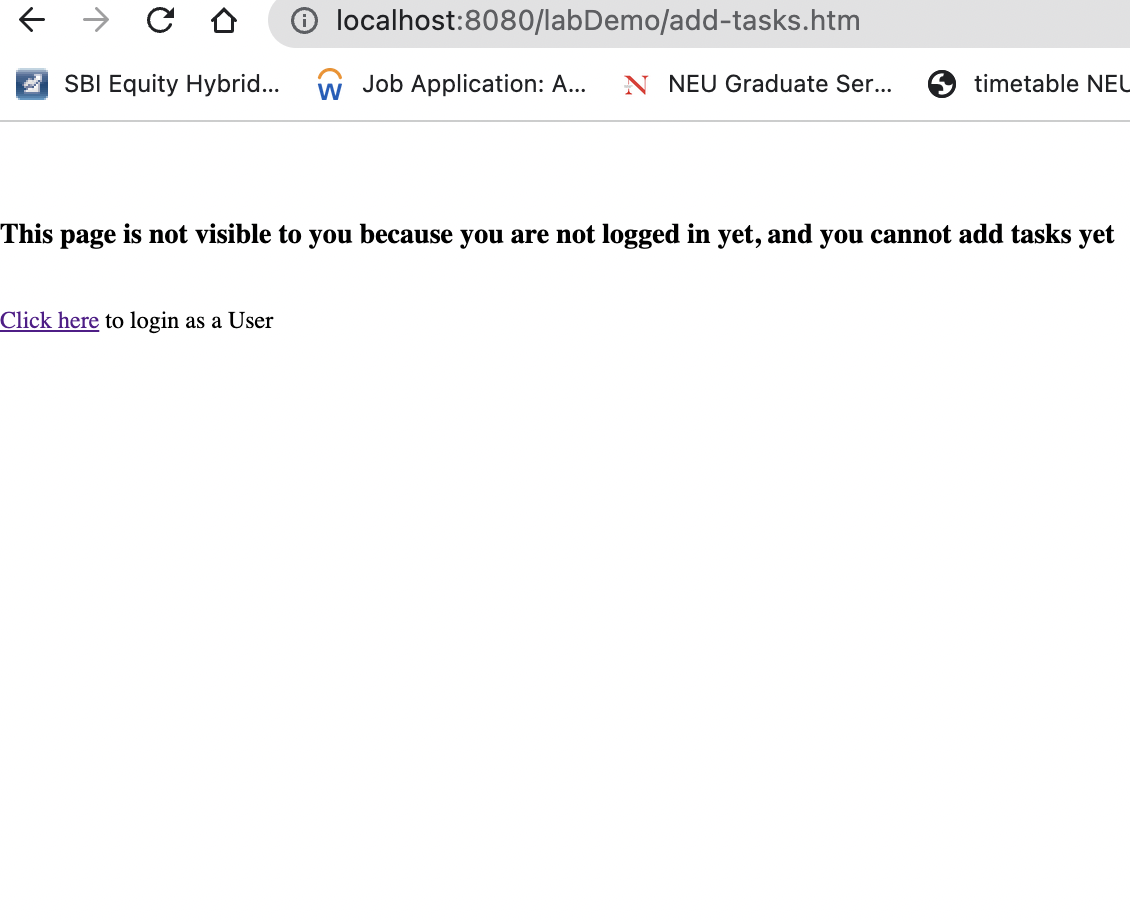
1. View Task/ID



1. Update Task



1. Task Activity



1. Failure Attempt without login

# **VI. Conclusions and Future Work**

Given the scope of this project and the inspiration for the project there is significant scope for improvement both in the design and implementation space of this project.

This project was a reflection of my ability to create, and manipulate multiple POJO classes using Hibernate using the help of controllers and annotations from Springboot and javax.

Here are a few steps I will be implementing or adding to this project going ahead:

i. Ability to filter tasks based on the due date, state, and priority.

ii. Build a project POJO class and add a list of Users to each class. Only members of a project can view the tasks of a project and assign tasks only between them. Some users with higher permissions can make tasks that are viewable and editable by both groups.

iii. Adding specific comments that automatically turn into hyperlinks either pointing to a user or a hyperlink.

iv. A notification tool that alerts users on any new tasks assigned to them or comments made on the tasks as soon as they log in.

Further, UI enhancements and the addition of dynamic material on our JSP pages would make the application lot more interactive and interesting to use, and much faster to adapt too.

# **VII. References**

1. <https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html> - Regular expression used for email validation
2. <https://docs.jboss.org/hibernate/orm/5.6/userguide/html_single/Hibernate_User_Guide.html> - Hibernate transaction management
3. <https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashMap.html> - Datatype iteration and management
4. <https://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/> - JSTL Documentation