TASK WEBAPP

# Technology Stack

I have developed a Spring Boot web application to fulfil the requirements of the assignment. The other technologies used were:

* Spring Boot
* Maven
* Thymeleaf
* Spring MVC
* HTML
* CSS
* Bootstrap
* Hibernate
* HSQL
* Spring Security
* Tomcat 8.5
* Git

# User Actions

Basic users (USER role) can perform the following actions:

* Login
* Logout
* View tasks associated with that user
* Add a new task for that user
* Update an existing task for that user (including checking/un-checking task)
* Remove a task associated with that user.

Admin users (ADMIN role) can also maintain the list of users using the application:

* View existing users and their associated role permissions
* Add new user
* Edit an existing user

# Front & Back End

A Model-View-Controller (MVC) pattern was implemented to build this app using Spring’s MVC framework.

The View layer was developed using a combination of **Thymeleaf,** **HTML** and **Bootstrap**. The templates are written in HTML and styled using Bootstrap. The Thymeleaf library is used extensively to apply the model to these templates.

The Controller layer exposes a series of endpoints which handle the user HTTP requests and direct them to the appropriate view components.

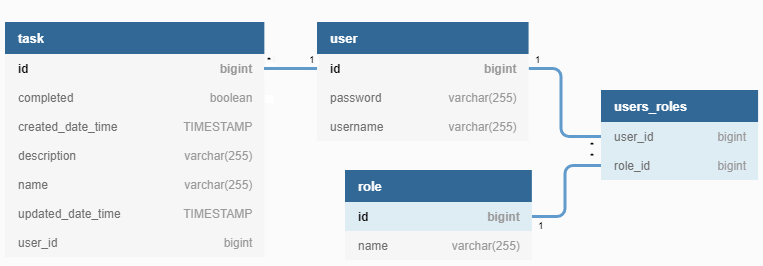
If I had more time I would have liked to try using a more modern web framework such as React or Angular and segregating the app out into separate microservices.

# Persistence

### Database

HQSL was used for the backend persistence technology. An **in-memory HSQL database instance** is spun up during application bootstrapping and some initial data is loaded into the underlying tables to facilitate testing. The tables are auto-generated by Hibernate using the relevant entity definitions.

### Schema Design



With regards to the database schema design, four tables have been created to represent the object models required for the app.

* **Task Table**: Represents the entity model for a task.
* **User Table**: Represents the entity model for a user.
* **Role Table**: Represents the entity model for a user role.
* **Users\_roles Table**: Captures the many-to-many relationship between a user and role.

There is a one-to-many relationship between a user and task (a user can have many tasks, a task can only have one user), and a many-to-many relationship between a user and role (a user can have many roles, and a role can have many users) as illustrated above.

### Data Access Layer

The data access layer is **implemented with Hibernate** and **uses a repository pattern** for data retrieval. The data store is abstracted from the underlying app logic via service components which are injected into the controller layer. These service components are **transactional** and create a new transaction for each call to the database.

# Security

All pages are **fully authenticated using Spring Security**, meaning none of the pages served by the app can be accessed without being logged in as a user with the necessary roles. Upon initial access to any page, the user is re-directed to a login screen which requires a username and password.

Once provided, the login credentials are then validated against a **bespoke security model**. For the sake of testing, two user roles have been set up:

* **USER Role**: Provides the homepage and view/add/update/remove tasks for that given user.
* **ADMIN Role**: The same access the USER role but it also provides access to the Users page which gives the ability to view/add/update users of the app.

If a user attempts to access a page with insufficient privileges (eg: a user with USER privileges tries to access the Admin page), they will be **redirected to an error page** explaining that they do not have the required permissions to access that page.

All **passwords are encrypted** using BCrypt to ensure that no sensitive information is stored in plain text in the DB. The decryption and authentication process is handled by Spring Security.

# Testing

Unit and integration tests have been implemented using the Spring Boot test suite, **JUnit** and **Mockito**. All controller view endpoints are tested for both anonymous and authenticated access to ensure the security configurations are behaving as expected.

If I had more time I would more vigorously test my application, making use of the JaCoCo Maven Plugin to ensure a sufficient level of test coverage during the build. I would also consider using some other web testing frameworks such as Selenium and perhaps a BDD test suite such has Cucumber.

# Performance

Due to time constraints I have not included any major performance tuning. Some potential performance improvements would be to introduce asynchronous processing on the service layer components, or introduce some sort of client caching.

# Building & Logging in

The WAR file can be built using the following Maven command:

* mvn clean package

The homepage URL is:

* <http://localhost:8080/task-service/app/home>

Login credentials are listed below:

|  |  |  |
| --- | --- | --- |
| **Username** | **Password** | **Roles** |
| basic\_user | pass | USER |
| admin\_user | admin | USER, ADMIN |

I tested using the application using Tomcat v8.5.