Final Capstone Starter Project

Spring Boot

Note: Spring Boot has been configured to run on port `9000` for this project. During our class sessions we used port `8080`. It's changed so as not to avoid any posible conflicts with other things that you may be running concurrently.

Database

You will need to create a database called ***final_capstone*** in pgAdmin.

Once the database is created open a `Query Tool` window for the ***final_capstone*** database so you can run the **setup.sql** to created and populate the User table and populate it with several users.

The **setup.sql** can be found in in the ***/final_capstone/backend-java-server/database*** folder.

Navigate to the ***/final_capstone/backend-java-server/database*** folder in the *Query Tool* window then open and run the **setup.sql** file you should see in that folder. There should be no errors.

Several application users have been populated into the **users** table you can use to login to the application:

Username	Password	Role
user	password	ROLE_USER
admin	password	ROLE_ADMIN
buddy	password	ROLE_USER
frank	password	ROLE_USER

Database users

Database users define who can do what in the database. We have been using the superuser `postgres` for our class work.

The database superuser—meaning `postgres`—must only be used for database administration. It must not be used by applications. As such, two database users are created for the capstone application to use as described here:

Username	Description
`final_capstone_owner`	This user is the schema owner. It has full access—meaning granted all privileges—to all database objects within the `capstone` schema and also has privileges to create new schema objects. This user can be used to connect to the database from PGAdmin for administrative purposes.

Username	Description
`final_capstone_appuser`	The application uses this user to make connections to the database. This user is granted `SELECT`, `INSERT`, `UPDATE`, and `DELETE` privileges for all database tables and can `SELECT` from all sequences. The application datasource has been configured to connect using this user.

Datasource

A Datasource has been configured for you in `/src/resources/application.properties`. It connects to a database called ***final_capstone*** using the `final_capstone_appuser` database user. If you chose a different database name be sure to change it in the `/src/resources/application.properties`

```
# datasource connection properties
spring.datasource.url=jdbc:postgresql://localhost:5432/final_capstone
spring.datasource.name=final_capstone
spring.datasource.username=final_capstone_appuser
spring.datasource.password=finalcapstone
```

JdbcTemplate

We will again be using Spring Dependency Injection in this application.

An example of how to use Spring Dependency Injection to get an instance of `JdbcTemplate` in your DAOs can be found in the `/src/main/java/com/techelevator/dao/JdbcUserDao.java` code. If also have examples in your **module-2 casptone** DAOs.

If you declare a field of type `JdbcTemplate` and add it as an argument to the constructor, Spring automatically injects an instance for you:

```
@Service
public class JdbcUserDao implements UserDao {

    private JdbcTemplate jdbcTemplate;

    public JdbcUserDao(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }
}
```

CORS

Any controller that'll be accessed from a client like the Vue Starter application needs the `@CrossOrigin` annotation. This tells the browser that you're allowing the client application to access this resource. Here is an example from the `AuthenticationController` provided:

```
@RestController
@CrossOrigin
public class AuthenticationController {
    // ...
}
```

Security

Most of the functionality related to Security is located in the `/src/main/java/com/techelevator/security` package. You shouldn't have to modify anything here, but feel free to go through the code if you want to see how things work.

Authentication Controller

There is a single controller provided in the `com.techelevator.controller` package called `AuthenticationController.java`.

This controller contains the `/login` and `/register` routes and works with the React starter as is. If you need to modify the user registration form, start here.

The authentication controller uses the `JdbcUserDao` to read and write data from the users table.

Sample Application

A sample application called `sample-application-tenmo' with a React front-end client and Java back-end server has been provided. It also includes a Java front-end cliend as well as a Vue framework front-end client.

Use this for examples on how to accomplish features you want to include in your final capstone app.

Example code that works is much more reliable than anything your can find in a Google search

Testing

DAO integration tests

`com.techelevator.dao.BaseDaoTests` has been provided for you to use as a base class for any DAO integration test. It initializes a Datasource for testing and manages rollback of database changes between tests.

`com.techelevator.dao.JdbUserDaoTests` has been provided for you as an example for writing your own DAO integration tests.

Remember that when testing, you're using a copy of the real database. The schema for the test database is defined in the same schema script for the real database, `database/schema.sql`. The data for the test database is defined separately within `/src/test/resources/test-data.sql`.