Capstone Starter Project

Database

Inside the `<project-root>/database/` directory, you'll find an SQL scripts (`.sql` file). These can be used to build and rebuild the authentication tables and users in a PostgreSQL database for the capstone project.

From a Windows terminal session (NOT GitBase), execute the following command to create `final-capstone` database:

```
createdb -U postgres final-capstone
```

When prompted for a password, enter `postgres1`

You should create a database connection in dbVisualizer for the `final-capstone` database.

Any SQL associated with your project, including the SQL mentioned above to set up the authentication tables should be run in dbVisualizer.

Spring Boot

Datasource

A Datasource has been configured for you in `/src/resources/application.properties`. It connects to a database named `final-capstone`. You can use a different database name if you want, but remember to change it here so the server can find the database:

```
# datasource connection properties
spring.datasource.url=jdbc:postgresql://localhost:5432/final_capstone
spring.datasource.name=final_capstone
spring.datasource.username=postgres
spring.datasource.password=postgres1
```

Put all your application in the `final-capstone` data base.

```
DO NOT MAKE ANY CHANGES TO THE AUTHENTICATION TABLES PROVIDED!

DOING SO MAY RENDER THE AUTHENTICATION CODE GIVEN UNUSABLE.

IF YOU NEED TO STORE USER INFORMATION FOR YOUR APPLICATION CONSIDER CREATING A NEW
TABLE AND LINKING IT TO THE AUTHENTICATION USERS TABLE IF YOU NEED THE USER NAME.
```

JdbcTemplate

If you look in `/src/main/java/com/techelevator/dao`, you'll see `UserSqlDAO`. This is an example of how to get an instance of `JdbcTemplate` in your 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 UserSqlDAO implements UserDAO {

    private JdbcTemplate jdbcTemplate;

    public UserSqlDAO(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:

```
@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 in the `com.techelevator.controller` package called `AuthenticationController.java`.

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

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

Testing

DAO integration tests

`com.techelevator.dao.DAOIntegrationTest` 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.

The following is an example of extending this class for writing your own DAO integration tests:

```
package com.techelevator.dao;

import com.techelevator.model.User;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import org.springframework.jdbc.core.JdbcTemplate;

import javax.sql.DataSource;

public class UserSqlDaoIntegrationTest extends DAOIntegrationTest {

    private UserSqlDAO userSqlDAO;

    @Before
    public void setup() {
        DataSource dataSource = this.getDataSource();
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
        userSqlDAO = new UserSqlDAO(jdbcTemplate);
    }

    @Test
    public void createNewUser() {
        boolean userCreated =
userSqlDAO.create("TEST_USER","test_password","user");
        Assert.assertTrue(userCreated);
        User user = userSqlDAO.findByUsername("TEST_USER");
        Assert.assertEquals("TEST_USER", user.getUsername());
    }

}
```