Capstone Starter Project

Database

Inside the `<project-root>/database/` directory, you'll find an executable Bash script (`.sh` file) and several SQL scripts (`.sql` files). These can be used to build and rebuild a PostgreSQL database for the capstone project.

From a terminal session, execute the following commands:

```
cd <project-root>/database/
./create.sh
```

This Bash script drops the existing database, if necessary, creates a new database named `final_capstone`, and runs the various SQL scripts in the correct order. You don't need to modify the Bash script unless you want to change the database name.

Each SQL script has a specific purpose as described below:

File Name	Description
`data.sql`	This script populates the database with any static setup data or test/demo data. The project team should modify this script.
`dropdb.sql`	This script destroys the database so that it can be recreated. It drops the database and associated users. The project team shouldn't have to modify this script.
`schema.sql`	This script creates all of the database objects, such as tables and sequences. The project team should modify this script.
`user.sql`	This script creates the database application users and grants them the appropriate privileges. The project team shouldn't have to modify this script. See the next section for more information on these users.

Database users

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 below:

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.

Spring Boot

Datasource

A Datasource has been configured for you in `/src/resources/application.properties`. It connects to the database using the `capstone_appuser` database user. You can change the name of this database if you want, but remember to change it here and in the `create.sh` script in the database folder:

```
# 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

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());
    }

}
```