Capstone Starter Project

Database

Inside the `<project-root>/database/` directory you will find an executable Bash script (`.sh` file) and several SQL scripts (`.sql` files). These can be used to (re)build a PostgreSQL database for the capstone project. From a terminal session, simply execute the following commands:

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

This Bash script will drop the existing database (if necessary), create a new databsae named `capstone`, and run the various SQL scripts in the correct order. You do not need to modify the Bash script.

Each of the SQL scripts has a specific purpose as described below:

File Name	Description
`data.sql`	This script is used to populate the database with any static setup data or test/demo data. This script is intended to be modified by the project team.
`dropdb.sql`	This script is used to destroy the database so that it can be recreated. It drops the database and associated users. This script is **not** intended to be modified by the project team.
`schema.sql`	This script is used to create all of the database objects (e.g. tables and sequences). This script is intended to be modified by the project team
`user.sql`	This script is used to create the database application users and grant them the appropriate privileges. See below for more information on these users. This script is **not** intended to be modified by the project team.

Database Users

The database superuser (i.e. `postgres`) should only be used for database administration and should not be used by applications. As such, two database users are created for use by the capstone application as described below:

Username	Description
`capstone_owner`	This user is the schema owner. It has full access (i.e. 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
`capstone_appuser`	This user is intended to be used by the application to make connections to the database. This user is granted `SELECT`, `INSERT`, `UPDATE`, `DELETE` privileges for all database tables and can `SELECT` from all sequences. The application datasource has been configured to connect using this user.

Spring MVC Configuration

Datasource

A Datasource has been configured that can be injected into your DAO objects. It connects to the database using the `capstone_appuser` database user. You can change the name of this database if you wish, but remember to change it here and in the `create-capstone-db.sh` script in the database folder.

Database Transactions

The Datasource has been configured to disable autocommit behavior. Instead, database transactions can be managed by using the `@Transactional` annotation on Controllers that make database modifications.

JSP

Spring has been configured to look for JSP files in the `<project-root>/src/main/webapp/WEB-INF/jsp/` directory.

Web Resources

The following directories have been created for static web resource files:

Directory	Description
`<project-root>/src/main/webapp/css/`	`.css` files go here
`<project-root>/src/main/webapp/img/`	image files (e.g. `.png`, `.jpg`, `.gif`) go here
`<project-root>/src/main/webapp/js/`	`.js` files go here

JQuery and Bootstrap

Minified versions of the JQuery Core and Validation libraries (including the "Additional Methods" library) have been included in the `<project-root>/src/main/webapp/js/` directory.

Minified versions of the Bootstrap CSS and Javascript files have been included in the `<project-root>/src/main/webapp/css/` and `<project-root>/src/main/webapp/js/` directories respectively.

Testing

DAO Integration Tests

`com.techelevator.DAOIntegrationTest` has been provided for 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;

import org.junit.Before;
import javax.sql.DataSource;

public class MyJdbcDaoIntegrationTest extends DAOIntegrationTest {

    private MyJdbcDao dao;

    @Before
    public void setup() {
        DataSource dataSource = this.getDataSource();
        dao = new MyJdbcDao(dataSource);
    }

    @Test
    public void do_that_thing() {
        // use the dao here to perform some kind of test
    }
}
```

Deploying

The project is already set up and ready to be deployed to Heroku. You will need to create a new Heroku application using these commands at the root directory:

```
heroku create
heroku config:set SPRING_PROFILES_ACTIVE=heroku
```

And then use this command to push your application:

```
git push heroku master
```

Once that's complete, you will want to set up the database on Heroku by sending your SQL files up to it:

```
heroku psql < database/schema.sql
heroku psql < database/data.sql
```