Overview

This project is a basic template that provides the following capabilities out of the box:

- JWT creating and handling
- Authentication
- Authorization

JWT creating and handling

This project is designed to handle authentication via stateless JSON Web Tokens (JWTs). A user can log in and obtain a web token via the URL `/login` with a POST that includes the user's username and password. The service will then return the JWT, which a front-end or mobile application will be able to use to authorize each new request.

JWTs will be taken from each request and verified and the user added to the session object so that the rest of the application will have access to the users from there. That also means that any verification that the developer wants to do will work the same as the existing authentication application used for Spring MVC.

Unauthenticated paths

If you have endpoints in your api that you want someone to be able to access without being authenticated, you can add those paths to the `springmvc-servlet.xml` file in the section for the `mvc:interceptors` section:

```
<mvc:interceptors>
  <mvc:interceptor>
    <mvc:mapping path="/**"/>
    <bean class="com.techelevator.authentication.JwtAuthInterceptor">
      <property name="excludedUrls">
        <list>
          <!-- Every url in the app must be authenticated
            except /login -->
          <value>/login</value>
        </list>
      </property>
    </bean>
  </mvc:interceptor>
</mvc:interceptors>
```

Just add more `<value>` tags under the `<list>` for `excludedUrls`.

Authentication

The authentication provider provided allows you to develop code in Spring MVC without having to develop your own authentication/authorization framework. The `AuthProvider` provided defines a number of methods that are capable of being used from various parts of your application. This includes but is not limited to:

- Get current logged in user
- Create new user
- Log in as user

See `AuthProvider.java` for a full description of how the methods are intended to be used in your application.

Set Up

A `SessionAuthProvider` class is included with this project to implement the `AuthProvider` interface. As such the following items need to be configured.

Database

To setup the database you will need to run the script `backend/database/create_db.sh`. This will run a series of SQL scripts that setup the permissions, schema and initial data. You will notice in the script that there is a database name `userdb`. You should rename this to reflect your final capstone project.

```
#!/bin/bash
BASEDIR=$(dirname $0)
DATABASE=userdb
psql -U postgres -f "$BASEDIR/dropdb.sql" &&
createdb -U postgres $DATABASE &&
psql -U postgres -d $DATABASE -f "$BASEDIR/schema.sql" &&
psql -U postgres -d $DATABASE -f "$BASEDIR/user.sql" &&
psql -U postgres -d $DATABASE -f "$BASEDIR/data.sql"
```

Once you rename the database and run the script you will need to update the database url in `/backend/src/main/webapp/WEB-INF/springmvc-servlet.xml`

```
<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource"
destroy-method="close">
  <property name="driverClassName" value="org.postgresql.Driver" />
  <property name="url" value="jdbc:postgresql://localhost:5432/userdb" />
  <property name="username" value="postgres" />
  <property name="password" value="postgres1" />
</bean>
```

Usage

You can access the `AuthProvider` by allowing it to be injected into your controllers.

```
@Autowired
private AuthProvider auth;
```

Once you have an instance of the `AuthProvider` you can invoke methods on it.

- `getCurrentUser()` - will return the current logged in user (null if they are not)
- `changePassword(String existingPassword, String newPassword)` - will validate the user's existing credentials and change their password
- `register(String username, String password, String role)` - will create a new user with the provided credentials and role

If you want to restrict access to a specific controller or controller action, you can call the method `userHasRole(String[] roles)` to see if the currently logged in user has any of the roles defined. If not, it will return a false and you can define what to do at that point.

```
if( ! auth.userHasRole(new String {"admin", "editor"}) { // If user
doesn't have the admin or editor role
    throw new UnauthorizedException();
}
```