CSCI 5308 - Advanced Topics in Software Development:
Final Project Documentation – Group 20

Code Repository: https://git.cs.dal.ca/courses/2022-winter/csci-5308/group20

BUILD STEPS

Prerequisites: Staff Scheduler requires Git for checking out the code from the repository, Java version 1.8 or above, and Maven version 3 or above for compiling and building the source code.

Procedure for obtaining the code & set-up for build: The source code for the Staff Scheduler application resides in a repository in GitLab (the link for which is provided above), which can be checked out using the 'git checkout command. After the code has been successfully checked out, please import the project in the IDE as a Maven Project using the pom.xml available at the root level.

Build Steps: Once the project has been loaded as a Maven project, we can build the project using the Maven artifact goals.

- Compile: mvn clean compile
- Build: mvn clean package | mvn clean install
- Test: mvn test
- Run: mvn spring-boot: run

The output of the build will be in the 'target' folder in the content directory, which will have the generated class files and the executable jar file.

DEPLOYMENT

The Staff Scheduler application is a Spring Boot Application that requires Java 1.8 to be installed on the machine on which the application is being built or deployed. It is also a maven-based project and requires Maven 3.0 or above for executing maven goals such as Compile, Build, Test, and Deploy.

The web application runtime environments are deployed on Heroku Servers, which also support Java version 1.8.

There are three application environments:

- Develop
- Test
- Production

Each one of these environments has its own respective database with identical database and table schemas.

The source code repository for the Staff Scheduler application is hosted on GitLab, where it is also integrated with CI/CD Pipelines. The pipeline has Build, Test, Code-Quality, and Deployment stages for the main branches associated with each environment. Below are the respective branches for each environment:

- The Production environment picks up the source code available on the 'main' branch.
- The Test environment picks up the source code available on the 'test-app' branch.
- The Develop environment picks up the source code available on the 'develop' branch.

The command used in 'gitalb-ci.yml' for the deployment is

"- dpl --provider=heroku --app=app-staff-scheduler --api-key=$HEROKU_API_KEY".

Here the provider is "Heroku", the name of the application in Heroku is "app-staff-scheduler" and the api-key parameter is generated by the application owner's Heroku account.

DEPENDENCIES

The application has the following dependencies which are loaded by the pom.xml during the build process. These are all maven dependencies that download the resources during the build process into the application context and allow the functionalities in the code.

No.	Name	Description
1	spring-boot-starter-data-jpa	Spring Data JPA handles most of the complexity of JDBC-based database access and ORM (Object Relational Mapping). It reduces the boilerplate code required by JPA. It makes the implementation of your persistence layer easier and faster.
2	spring-boot-starter-web	Starter of Spring web uses Spring MVC, REST, and Tomcat as a default embedded server. The single spring-boot-starter-web dependency transitively pulls in all dependencies related to web development. It also reduces the build dependency count
3	mysql-connector-java	A JDBC driver is a set of Java classes that implement the JDBC interfaces, targeting a specific database. The JDBC interfaces come with standard Java, but the implementation of these interfaces is specific to the database you need to connect to. Such an implementation is called a JDBC driver.
4	spring-boot-starter-test	By using the spring-boot-starter-test 'Starter' which imports both Spring Boot test modules as well as JUnit, AssertJ, Hamcrest, and several other useful libraries.
5	Lombok	Project Lombok is a java library that automatically plugs into your editor and builds tools, spicing up your java. We Never have to write another getter or equals method again, with one annotation our class has a fully-featured builder, Automate our logging variables, and much more.

6	junit	JUnit 5 is the next generation of JUnit. The goal is to create an up-to-date foundation for developer-side testing on the JVM. This includes focusing on Java 8 and above and enabling many different styles of testing.
7	java.ws.rs-api	High-level interfaces and annotations are used to create RESTful service resources.
8	springdoc-openapi-ui	java library helps to automate the generation of API documentation using spring boot projects. springdoc-openapi works by examining an application at runtime to infer API semantics based on Spring configurations, class structure, and various annotations.
9	springdoc-openapi-data-rest	At first, we might expect SpringDoc to add a page, size, and sort query parameters to the generated documentation. However, by default, SpringDoc does not meet this expectation. To unlock this feature, we should add the springdoc-openapi-data-rest dependency:
10	java.mail-api	The JavaMail API provides a platform-independent and protocol-independent framework to build mail and messaging applications. The JavaMail API is available as an optional package for use with the Java SE platform and is also included in the Java EE platform
11	java. mail	The JavaMail API provides classes that model a mail system.
12	commons-text	Apache Commons Text is a library focused on algorithms working on strings.
13	bootstrap	This is a front-end dependency utilized for UI styling and beautification of pages.
14	fullcalendar	This is a JavaScript-based library that helps with the rendering of a calendar in the UI.