Red Hat Process Automation Manager 7.8

Running and modifying the employee rostering starter application for Red Hat Business Optimizer using an IDE

Last Updated: 2021-03-03

Red Hat Process Automation Manager 7.8 Running and modifying the employee rostering starter application for Red Hat Business Optimizer using an IDE

Red Hat Customer Content Services
brms-docs@redhat.com

Legal Notice

Abstract

This document describes how to run and modify the Employee Rostering starter application included as a reference implementation in Red Hat Process Automation Manager 7.8.

Table of Contents

PREFACE

As a business rules developer, you can use an IDE to build, run, and modify the **optaweb-employee-rostering** starter application that uses the Red Hat Business Optimizer functionality.

Prerequisites

- You use an integrated development environment, such as Red Hat CodeReady Studio or IntelliJ IDEA.
- You have an understanding of the Java language.
- You have an understanding of React and TypeScript. This requirement is necessary to develop the OptaWeb UI.

CHAPTER 1. OVERVIEW OF THE EMPLOYEE ROSTERING STARTER APPLICATION

The employee rostering starter application assigns employees to shifts on various positions in an organization. For example, you can use the application to distribute shifts in a hospital between nurses, guard duty shifts across a number of locations, or shifts on an assembly line between workers.

Optimal employee rostering must take a number of variables into account. For example, different skills can be required for shifts in different positions. Also, some employees might be unavailable for some time slots or might prefer a particular time slot. Moreover, an employee can have a contract that limits the number of hours that the employee can work in a single time period.

The Red Hat Business Optimizer rules for this starter application use both hard and soft constraints. During an optimization, the planning engine may not violate hard constraints, for example, if an employee is unavailable (out sick), or that an employee cannot work two spots in a single shift. The planning engine tries to adhere to soft constraints, such as an employee's preference to not work a specific shift, but can violate them if the optimal solution requires it.

CHAPTER 2. BUILDING AND RUNNING THE EMPLOYEE ROSTERING STARTER APPLICATION

You can build the employee rostering starter application from the source code and run it as a JAR file.

Alternatively, you can use your IDE, for example, Eclipse (including Red Hat CodeReady Studio), to build and run the application.

2.1. PREPARING DEPLOYMENT FILES

You must download and prepare the deployment files before building and deploying the application.

Procedure

1. Download the **rhpam-7.8.0-reference-implementation.zip** file from the Software Downloads page for Red Hat Process Automation Manager 7.8.
2. Unzip the downloaded archive.
3. Copy the contents of the **jboss-rhba-7.8.0.GA-maven-repository/maven-repository** subdirectory into the **~/.m2/repository** directory.
4. Expand the **rhpam-7.8.0-optaweb-employee-rostering.zip** file that is extracted from the reference implementation archive.
   The **optaweb-employee-rostering-distribution-7.39.0.Final-redhat-00005** folder is created. This folder is the base folder in subsequent parts of this document.

NOTE

File and folder names might have higher version numbers than specifically noted in this document.

2.2. RUNNING THE EMPLOYEE ROSTERING STARTER APPLICATION JAR FILE

You can run the Employee Rostering starter application from a JAR file included in the reference implementation download.

Prerequisites

- You have downloaded and extracted the **rhpam-7.8.0-reference-implementation.zip** file as described in Section 2.1, “Preparing deployment files”.
- A Java Development Kit is installed.
- Maven is installed.
- The host has access to the Internet. The build process uses the Internet for downloading Maven packages from external repositories.

Procedure

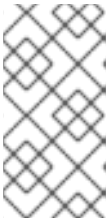
1. In a command terminal, change to the **sources** directory.

2. Enter the following command:

```
mvn clean install -DskipTests
```

3. Wait for the build process to complete.
4. Navigate to the **optaweb-employee-rostering-distribution-7.39.0.Final-redhat-00005/sources/optaweb-employee-rostering-standalone/target** directory.
5. Enter the following command to run the Employee Rostering JAR file:

```
java -jar optaweb-employee-rostering-standalone-*-exec.jar
```



NOTE

This command starts the employee rostering application with a non-production database. To start the employee rostering application with a production database, add the **--spring.profiles.active=production** argument to the preceding command.

6. To access the application, enter **http://localhost:8080/** in a web browser.

2.3. BUILDING AND RUNNING THE EMPLOYEE ROSTERING STARTER APPLICATION USING MAVEN

You can use the command line to build and run the employee rostering starter application.

If you use this procedure, the data is stored in memory and is lost when the server is stopped. To build and run the application with a database server for persistent storage, see Section 2.4, "Building and running the employee rostering starter application with persistent data storage from the command line".

Prerequisites

- You prepared the deployment files as described in Section 2.1, "Preparing deployment files".
- A Java Development Kit is installed.
- Maven is installed.
- The host has access to the Internet. The build process uses the Internet for downloading Maven packages from external repositories.

Procedure

1. Navigate to the **optaweb-employee-rostering-backend** directory.
2. Enter the following command:

```
mvn spring-boot:run
```

3. Navigate to the **optaweb-employee-rostering-frontend** directory.
4. Enter the following command:

```
npm start
```

NOTE

If you use **npm** to start the server, **npm** monitors code changes.

5. To access the application, enter **http://localhost:3000/** in a web browser.

2.4. BUILDING AND RUNNING THE EMPLOYEE ROSTERING STARTER APPLICATION WITH PERSISTENT DATA STORAGE FROM THE COMMAND LINE

If you use the command line to build the employee rostering starter application and run it, you can provide a database server for persistent data storage.

Prerequisites

- You prepared the deployment files as described in Section 2.1, “Preparing deployment files”.
- A Java Development Kit is installed.
- Maven is installed.
- The host has access to the Internet. The build process uses the Internet for downloading Maven packages from external repositories.
- You have a deployed MySQL or PostrgeSQL database server.

Procedure

1. In a command terminal, navigate to the **optaweb-employee-rostering-standalone/target** directory.
2. Enter the following command to run the Employee Rostering JAR file:

```
java -jar optaweb-employee-rostering-standalone-*-exec.jar --
spring.profiles.active=production
spring.datasource.url=<DATABASE_URL> --spring.datasource.username=
<DATABASE_USER> --spring.datasource.password=<DATABASE_PASSWORD>
```

In this example, replace the following placeholders:

- **<DATABASE_URL>**: URL to connect to the database, for example **jdbc:postgresql://postgresql:5432/MY_DATABASE**
- **<DATABASE_USER>**: The user to connect to the database
- **<DATABASE_PASSWORD>**: The password for **<DATABASE_USER>**

2.5. BUILDING AND RUNNING THE EMPLOYEE ROSTERING STARTER APPLICATION USING INTELLIJ IDEA

You can use IntelliJ IDEA to build and run the employee rostering starter application.

Prerequisites

- You have downloaded the Employee Rostering source code, available from the Employee Rostering GitHub page.
- IntelliJ IDEA, Maven, and Node.js are installed.
- The host has access to the Internet. The build process uses the Internet for downloading Maven packages from external repositories.

Procedure

1. Start IntelliJ IDEA.
2. From the IntelliJ IDEA main menu, select **File → Open**.
3. Select the root directory of the application source and click **OK**.
4. From the main menu, select **Run → Edit Configurations**.
5. In the window that appears, expand **Templates** and select **Maven**. The Maven sidebar appears.
6. In the Maven sidebar, select **optaweb-employee-rostering-backend** from the **Working Directory** menu.
7. In **Command Line**, enter **spring-boot:run**.
8. To start the back end, click **OK** .
9. In a command terminal, navigate to the **optaweb-employee-rostering-frontend** directory.
10. Enter the following command to start the front end:

```
npm start
```

11. To access the application, enter **http://localhost:3000/** in a web browser.

CHAPTER 3. OVERVIEW OF THE SOURCE CODE OF THE EMPLOYEE ROSTERING STARTER APPLICATION

The employee rostering starter application consists of the following principal components:

- A **backend** that implements the rostering logic using Red Hat Business Optimizer and provides a REST API
- A **frontend** module that implements a user interface using React and interacts with the **backend** module through the REST API

You can build and use these components independently. In particular, you can implement a different user interface and use the REST API to call the server.

In addition to the two main components, the employee rostering template contains a generator of random source data (useful for demonstration and testing purposes) and a benchmarking application.

Modules and key classes

The Java source code of the employee rostering template contains several Maven modules. Each of these modules includes a separate Maven project file (**pom.xml**), but they are intended for building in a common project.

The modules contain a number of files, including Java classes. This document lists all the modules, as well as the classes and other files that contain the key information for the employee rostering calculations.

- **optaweb-employee-rostering-benchmark** module: Contains an additional application that generates random data and benchmarks the solution.
- **optaweb-employee-rostering-distribution** module: Contains README files.
- **optaweb-employee-rostering-docs** module: Contains documentation files.
- **optaweb-employee-rostering-frontend** module: Contains the client application with the user interface, developed in React.
- **optaweb-employee-rostering-backend** module: Contains the server application that uses Red Hat Business Optimizer to perform the rostering calculation.
  - **src/main/java/org.optaweb.employeerostering.service.roster/rosterGenerator.java**: Generates random input data for demonstration and testing purposes. If you change the required input data, change the generator accordingly.
  - **src/main/java/org.optaweb.employeerostering.domain.employee/EmployeeAvailability.java**: Defines availability information for an employee. For every time slot an employee can be unavailable, available, or it can be designated a preferred time slot for the employee.
  - **src/main/java/org.optaweb.employeerostering.domain.employee/Employee.java**: Defines an employee. An employee has a name, a list of skills, and works under a contract. Skills are represented by skill objects.
  - **src/main/java/org.optaweb.employeerostering.domain.roster/Roster.java**: Defines the calculated rostering information.
  - **src/main/java/org.optaweb.employeerostering.domain.shift/Shift.java**: Defines a shift to which an employee can be assigned. A shift is defined by a time slot and a spot. For

example, in a diner there could be a shift in the **Kitchen** spot for the February 20 8AM-4PM time slot. Multiple shifts can be defined for a given spot and time slot. In this case, multiple employees are required for this spot and time slot.

- **src/main/java/org.optaweb.employeerostering.domain.skill/Skill.java**: Defines a skill that an employee can have.
- **src/main/java/org.optaweb.employeerostering.domain.spot/Spot.java**: Defines a spot where employees can be placed. For example, a **Kitchen** can be a spot.
- **src/main/java/org.optaweb.employeerostering.domain.contract/Contract.java**: Defines a contract that sets limits on work time for an employee in various time periods.
- **src/main/java/org.optaweb.employeerostering.domain.tenant/Tenant.java**: Defines a tenant. Each tenant represents an independent set of data. Changes in the data for one tenant do not affect any other tenants.
- ***View.java**: Classes related to domain objects that define value sets that are calculated from other information; the client application can read these values through the REST API, but not write them.
- ***Service.java**: Interfaces located in the service package that define the REST API. Both the server and the client application separately define implementations of these interfaces.

CHAPTER 4. MODIFYING THE EMPLOYEE ROSTERING STARTER APPLICATION

To modify the employee rostering starter application to suit your needs, you must change the rules that govern the optimization process. You must also ensure that the data structures include the required data and provide the required calculations for the rules. If the required data is not present in the user interface, you must also modify the user interface.

The following procedure outlines the general approach to modifying the employee rostering starter application.

Prerequisites

- You have a build environment that successfully builds the application.
- You can read and modify Java code.

Procedure

1. Plan the required changes. Answer the following questions:
   - What are the additional scenarios that *must* be avoided? These scenarios are *hard constraints*.
   - What are the additional scenarios that the optimizer must *try to avoid* when possible? These scenarios are *soft constraints*.
   - What data is required to calculate if each scenario is happening in a potential solution?
   - Which of the data can be derived from the information that the user enters in the existing version?
   - Which of the data can be hardcoded?
   - Which of the data must be entered by the user and is not entered in the current version?
2. If any required data can be calculated from the current data or can be hardcoded, add the calculations or hardcoding to existing view or utility classes. If the data must be calculated on the server side, add REST API endpoints to read it.
3. If any required data must be entered by the user, add the data to the classes representing the data entities (for example, the **Employee** class), add REST API endpoints to read and write the data, and modify the user interface to enter the data.
4. When all the data is available, modify the rules. For most modifications, you must add a new rule. The rules are located in the **src/main/resources/org/optaweb/employeerostering/service/solver/employeeRosteringScoreRules.drl** file of the **optaweb-employee-rostering-backend** module.
   Use the Drools language for the rules. For reference information about the Drools rule language, see *Designing a decision service using DRL rules* . Classes defined in the **optaweb-employee-rostering-backend** modules are available to the decision engine.
5. After modifying the application, build and run it.

APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Wednesday, March 03, 2021.