BookStore

Generating the project

We will use the Kodnito MicroProfile Archetype to generate our project. Open your terminal and type in the following command to generate our project.

```
mvn archetype:generate -DarchetypeGroupId=com.kodnito -DarchetypeArtifactId=kodnito
-microprofile-archetype -DarchetypeVersion=1.0.1 -DgroupId=com.kodnito.bookstore.rest
-DartifactId=book-store -Dversion=1.0-SNAPSHOT
```

Type Enter and you will have your new project generated. Now go to the project directory and type the following command for downloading the dependencies and when it’s done, open the project in your favourite IDE. Open the pom.xml and add the following:

```
    <dependency>
    <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.4.196</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.eclipse.persistence</groupId>
        <artifactId>eclipselink</artifactId>
        <version>2.7.4</version>
    </dependency>
    <dependency>
        <groupId>javax.transaction</groupId>
        <artifactId>javax.transaction-api</artifactId>
        <version>1.3</version>
    </dependency>

    <plugins>
        <plugin>
            <groupId>fish.payara.maven.plugins</groupId>
            <artifactId>payara-micro-maven-plugin</artifactId>
            <version>1.0.1</version>
            <configuration>
                <payaraVersion>${version.payara.micro}</payaraVersion>
                <deployWar>true</deployWar>
                <commandLineOptions>
                    <option>
                        <key>--autoBindHttp</key>
                    </option>
                </commandLineOptions>
            </configuration>
        </plugin>
    </plugins>
```

Your pom.xml should look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.kodnito.bookstore.rest</groupId>
    <artifactId>book-store</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>
    <dependencies>
        <dependency>
            <groupId>org.eclipse.microprofile</groupId>
```

```
            <artifactId>microprofile</artifactId>
            <version>2.0.1</version>
            <type>pom</type>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <version>1.4.196</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>eclipselink</artifactId>
            <version>2.7.4</version>
        </dependency>
        <dependency>
            <groupId>javax.transaction</groupId>
            <artifactId>javax.transaction-api</artifactId>
            <version>1.3</version>
        </dependency>
    </dependencies>
    <build>
        <finalName>restapi</finalName>
        <plugins>
         <plugin>
                <groupId>fish.payara.maven.plugins</groupId>
                <artifactId>payara-micro-maven-plugin</artifactId>
                <version>1.0.1</version>
                <configuration>
                    <payaraVersion>${version.payara.micro}</payaraVersion>
                    <deployWar>true</deployWar>
                    <commandLineOptions>
                        <option>
                            <key>--autoBindHttp</key>
                        </option>
                    </commandLineOptions>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <failOnMissingWebXml>false</failOnMissingWebXml>
    <version.payara.micro>5.183</version.payara.micro>
    </properties>
</project>
```

We added dependencies for H2 database, JPA , Payara Micro Maven runtime and javax transaction

API. Now open the terminal and navigate to the project directory and type the following command to download the dependencies :

```
mvn clean install
```

Payara Micro Config

Create a new directory called `WEB-INF` inside `src/main/webapp` and inside `WEB-INF` directory create the `glassfish-resources.xml` file and add the following to configure DataSource:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1
Resource Definitions//EN" "http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
    <jdbc-resource
            pool-name="H2Pool"
            jndi-name="java:app/jdbc/restapi"/>
    <jdbc-connection-pool
            name="H2Pool"
            res-type="javax.sql.DataSource"
            datasource-classname="org.h2.jdbcx.JdbcDataSource">

        <property name="user" value="sa"/>
        <property name="password" value=""/>
        <property name="url" value="jdbc:h2:mem:restapiDB"/>
    </jdbc-connection-pool>
</resources>
```

We use the open source H2 database, which can be embedded in Java applications or run in the client mode. It's really easy to get started with H2 database, but I don't think it's a good idea to use it in production. This config will create an in memory based database called `restapiDB`. Now that we have our PayaraMicro DataSource configured it's time to create our `persistence.xml` file. Inside `src/main/resources` create the `persistence.xml` file and add the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
             xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
    <persistence-unit name="restapi_PU" transaction-type="JTA">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <jta-data-source>java:app/jdbc/restapi</jta-data-source>
        <properties>
            <property name="eclipselink.ddl-generation" value="create-tables"/>
            <property name="eclipselink.ddl-generation.output-mode" value="database"/>
        </properties>
    </persistence-unit>
</persistence>
```

`persistence.xml` is the standard configuration file for JPA and it has to be included in the `META-INF` directory. The `persistence.xml` file defines what provider to be used, the name of the persistence unit, how classes should be mapped to database tables. `eclipselink.ddl-generation` will create the database and tables.

Now that we have everything configured, it's time to start working on our API.

Entity

An Entity is a Java class that is marked with annotations that represent objects in a database. Create a new file called Book.java inside com.kodnito.bookstore.entity and make it look like this:

```
Unresolved directive in bookstore.adoc -
include::{bookstore_java_dir}/com/kodnito/bookstore/entity/Book.java[]
```

- `@Entity` annotation indicates that it is a JPA entity
- `@Table` annotation is used to name the table in the database
- `@NamedQueries` annotation is used to add multiple queries
- `@NamedQuery` annotation defines query with a name
- `@Id` annotation is used to define the primary key and the Id property is also annotated with `@GeneratedValue` to indicate that the Id should be generated automatically.

Business Logic

It's time to concentrate on the business logic code. It's always best to separate the code that each class does its own job. We will now create the BookService.java file for interacting with the database. Now create the BookService.java file inside com.kodnito.bookstore.service package and make it look like:

```
Unresolved directive in bookstore.adoc -
include::{bookstore_java_dir}/com/kodnito/bookstore/service/BookService.java[]
```

What does everything mean in this file, we start at the beginning of the file with the `@ApplicationScoped` annotation. When an object is annotated with the `@ApplicationScoped` annotation, it is created once for the duration of the application. `@PersistenceContext` annotation injects the EntityManager to be used at runtime. We have created five methods to interact with the database. getAll method will get all the objects from the books table, when we want a single object we will use the findById method with an id. Update method like it says will update an existing object, create method will create a new Book object and delete will delete an existing Book object from the database. The `@Transactional` annotation provides the application the ability to control the transaction boundaries.

Summary

In this chapter, we created our application from maven archetype, added the dependencies we need for our application, configured our application, created entities classes and created our business logic code for interacting with the database.