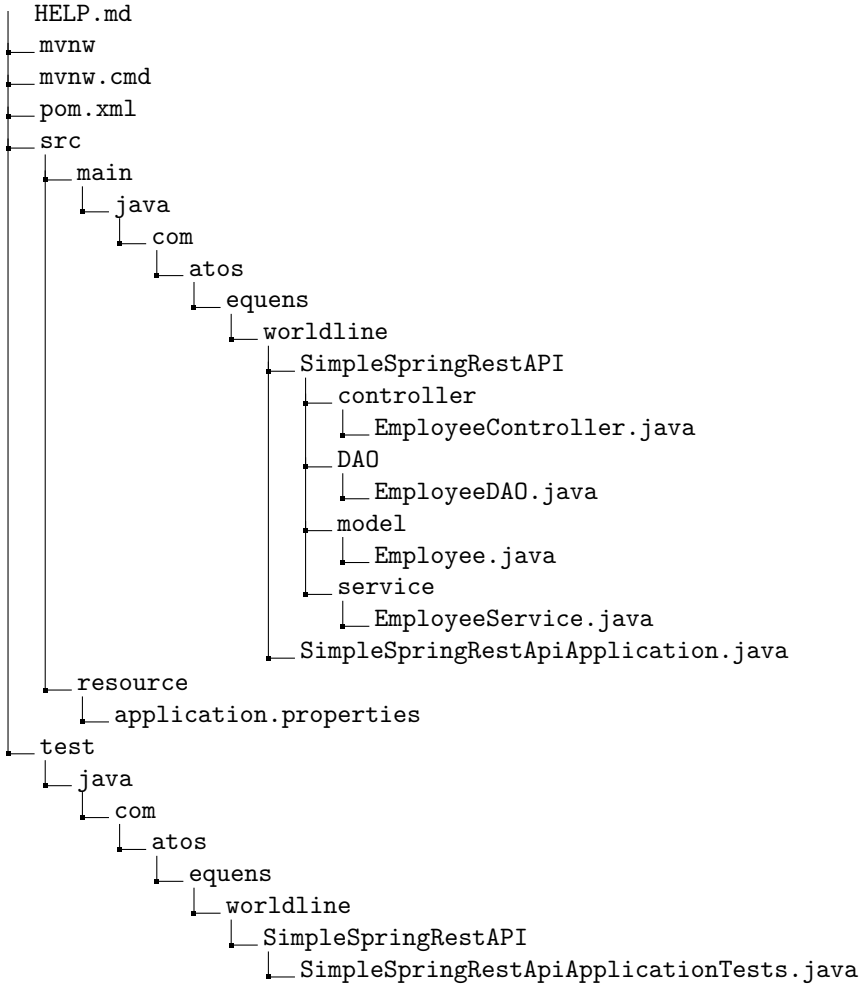
REpresentational **S**tate **T**ransfer

Web Service

Contents

1.0 Project Folder Structure



1.1 Introduction

1.2 Maven

1.3 Rest vs SOAP

1.4 Project Object Model

```
\\pom.xml
  <?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>
    <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.1.6.RELEASE</version>
      <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.atos.equens.worldline</groupId>
    <artifactId>Messenger</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>MessengerRestSpring</name>
    <description>Messenger Spring Boot Rest API project for Spring
        Boot</description>
    <dependencies>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jersey</artifactId>
      </dependency>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
      </dependency>
      <dependency>
        <groupId>com.oracle</groupId>
        <artifactId>ojdbc14</artifactId>
        <version>10.2.0.4.0</version>
      </dependency>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
      </dependency>
    </dependencies>
  </project>
```

The above pom.xml describes a spring boot maven application with minimum dependencies with which we can write the simple spring boot appliaction which illustrates the usange of Rest Web Services and its method. Let's us study the pom.xml file in brief just get to the purpose of each dependencies why being used.

1.4.0 Parent

Parent tag in above pom.xml differentiates the spring boot application from a simple maven appliaction. It tells maven that it is Spring boot appliaction asual as asks for transitive dependencies to be avalble.

```
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>2.1.6.RELEASE</version>
   <relativePath/> <!-- lookup parent from repository -->
</parent>
```

If you are running spring-boot application for the first time, required dependencies which are mentioned in pom.xml will be downloaded from the maven repository (.m2/repository). Downloaded dependencies will be stored in local repository which will be easy to use next time withought having to download every time you create new application.

1.4.1 Dependency

Dependency tag describes the required maven dependencies. In above pom.xml,I tried to use minimum dependencies which required to illustrate the standard Rest Api using Spring Boot. Let us go through each dependencies and purpose of being used.

1.4.1.0 Jersey Library

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-jersey</artifactId>
</dependency>
```

Jersey is an open source framework for developing RESTful Web Services in Java. It is a reference implementation of the Java API for RESTful Web Services (JAX-RS) specification.

1.4.1.1 Hibernate/JPA

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

The Java Persistence API (JPA) is a Java application programming interface specification that describes the management of relational data in applications

using Java Platform, Standard Edition and Java Platform, Enterprise Edition. Persistence in this context covers three areas:

- the API itself, defined in the javax.persistence package
- the API itself, defined in the javax.persistence package
- object/relational metadata

1.4.1.2 JDBC Driver

```
<dependency>
   <groupId>com.oracle</groupId>
   <artifactId>ojdbc14</artifactId>
   <version>10.2.0.4.0</version>
</dependency>
```

1.4.1.3 Junit/Spring Test

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-test</artifactId>
   <scope>test</scope>
</dependency>
```

Spring Boot provides a number of utilities and annotations to help when testing your application. Test support is provided by two modules: spring-boot-test contains core items, and spring-boot-test-autoconfigure supports auto-configuration for tests. Most developers use the spring-boot-starter-test "Starter", which imports both Spring Boot test modules as well as JUnit, AssertJ, Hamcrest, and a number of other useful libraries.

1.5 application.properties

Properties files are used to keep 'N' number of properties in a single file to run the application in a different environment. In Spring Boot, properties are kept in the application.properties file under the classpath.

The application.properties file is located in the src/main/resources directory. The code for this project's application.properties file is given below.

```
# ================================
# = DATA SOURCE
# ================================
# Set here configurations for the database connection
spring.datasource.url=jdbc:mysql://localhost:3306/Test
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
# Keep the connection alive if idle for a long time (needed in
    production)
spring.datasource.testWhileIdle=true
spring.datasource.validationQuery=SELECT 1
# ================================
# = JPA / HIBERNATE
# ================================
# Show or not log for each sql query
spring.jpa.show-sql=true
# Hibernate ddl auto (create, create-drop, update): with
    "create-drop" the database
# schema will be automatically created afresh for every start of
    application
spring.jpa.hibernate.ddl-auto=create-drop
# Naming strategy
spring.jpa.hibernate.naming.implicit-strategy=org.hibernate.boot.model.naming.ImplicitNamingStrateg
# Allows Hibernate to generate SQL optimized for a particular DBMS
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
server.port=8089
```

1.6 Annotations

1.6.0 JPA Annotations

- @Entity The @Entity annotation is used to specify that the class is an entity.

  ## Attributes:

  Name: The Name attribute is used to specify the entity name. It is an optional attribute. Defaults to the unqualified name of the entity class. It is contained in the javax.persistence package.

- @Table

  The Table annotation is used to specify the primary table for the annotated entity. Additional tables may be specified using SecondaryTable or SecondaryTables annotation.

```
@Entity
@Table(name = "EMPLOYEE")
public class Employee {
   @Id @GeneratedValue
   @Column(name = "id")
   private int id;
}
```

Attribute:

  - Name: The name of the table. It is an optional attribute.
  - Schema: The schema of the table. It is an optional attribute.
  - UniqueConstraints: Unique constraints that are to be placed on the table. It is an optional attribute.
  - Catalog: The catalog of the table. It is an optional attribute.
  - Indexes: Indexes for the table. It is an optional attribute.

- @Column

  The Column annotation is used to specify the mapped column for a persistent property or field. If no Column annotation is specified, the default value will be applied.

Attribute:

  - Name: The name of the column.
  - length: The column length.
  - nullable: Whether the database column is nullable.
  - Table: The name of the table that contains the column.
  - Unique: Whether the column is a unique key.
  - updatable: Whether the column is included in SQL UPDATE statements generated by the persistence provider.
  - Precision: The precision for a decimal (exact numeric) column.
  - insertable: Whether the column is included in SQL INSERT statements generated by the persistence provider.
  - columnDefinition: The SQL fragment that is used when generating the DDL for the column.

- @Id

  The @Id annotation is used to specify the primary key of an entity. The field or property to which the Id annotation is applied should be one of the following types:

  - 1. Any Java primitive type
  - 2. Any primitive wrapper type.
  - 3. String
  - 4. java.util.Date
  - 5. java.sql.Date
  - 6. java.math.BigDecimal
  - 7. java.math.BigInteger

- @GeneratedValue

  The @GeneratedValue annotation provides the specification of generation strategies for the primary keys values.
  Example :

```
@Id @GeneratedValue(strategy=GenerationType.TABLE ,
    generator="student_generator")
```

  - 1. Strategy: The strategy attribute is used to specify the primary key generation strategy that the persistence provider must use to generate the annotated entity primary key. It is an optional attribute. Strategy values are defined in javax.persistence.GeneratorType enumeration which are as follows:
    - 1. AUTO: Based on the database's support for primary key generation framework decides which generator type to be used.
    - 2. IDENTITY: In this case database is responsible for determining and assigning the next primary key.
    - 3. SEQUENCE: A sequence specify a database object that can be used as a source of primary key values. It uses @SequenceGenerator.
    - 4. TABLE: It keeps a separate table with the primary key values. It uses @TableGenerator. Note: Default value of Strategy attribute is AUTO.
  - 2. Generator: The Generator attribute is used to specify the name of the primary key generator to use as specified in the SequenceGenerator or TableGenerator annotation. It is an optional attribute. Please follow and like us: Save

1.6.1 Spring Framework Annotations

- @Repository
- @Autowired
- @Service
- @RequestController
- @RequestMapping
- @RequestMethod
- @PathVariable

1.7 Java Code

1.7.0 Problem Statement

1.7.1 Model

```
\\Employee.java
package com.atos.equens.worldline.SimpleSpringRestAPI.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "Employee")
public class Employee {

   @Column(name = "ID")
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private int id;
   @Column(name = "FirstName")
   private String firstName;
   @Column(name = "LastName")
   private String lastName;
   public int getId() {
      return id;
   }
   public void setId(int id) {
      this.id = id;
   }
```

```
    public String getFirstName() {
       return firstName;
    }
    public void setFirstName(String firstName) {
       this.firstName = firstName;
    }
    public String getLastName() {
       return lastName;
    }
    public void setLastName(String lastName) {
       this.lastName = lastName;
    }
}
```

1.7.2 DAO

```
\\EmployeeDAO.java
package com.atos.equens.worldline.SimpleSpringRestAPI.DAO;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.atos.equens.worldline.SimpleSpringRestAPI.model.Employee;

@Repository
public interface EmployeeDAO extends JpaRepository<Employee, Integer>{

}
```

1.7.3 Servcie

```
\\EmployeeService.java
package com.atos.equens.worldline.SimpleSpringRestAPI.service;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.atos.equens.worldline.SimpleSpringRestAPI.DAO.EmployeeDAO;
import com.atos.equens.worldline.SimpleSpringRestAPI.model.Employee;

@Service
public class EmployeeService {

   @Autowired
```

```
   EmployeeDAO employeeDAO;

   public List<Employee> getAllEmployee(){
      return employeeDAO.findAll();
   }

   public Employee addEmployee(Employee employee) {
      return employeeDAO.save(employee);
   }

   public Optional<Employee> getEmployeeByID(int id) {
      return employeeDAO.findById(id);
   }

   public Employee updateEmployee(Employee employee) {
      return employeeDAO.save(employee);
   }

   public void deleteEmployeeById(int id) {
      employeeDAO.deleteById(id);
   }

   public void deleteAllEmployee() {
      employeeDAO.deleteAll();
   }
}
```

1.7.4 Controller

```
\\EmployeeController.java
package com.atos.equens.worldline.SimpleSpringRestAPI.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.atos.equens.worldline.SimpleSpringRestAPI.model.Employee;
import
    com.atos.equens.worldline.SimpleSpringRestAPI.service.EmployeeService;

@RestController
@RequestMapping("/employee")
public class EmployeeController {

   @Autowired
```

```
    EmployeeService employeeService;

    @RequestMapping(value = "/getallemployees",
        method=RequestMethod.GET)
    public List<Employee> getAllEmployee(){
       return employeeService.getAllEmployee();
    }


    @RequestMapping(
       value = "/addemployee",
       method = RequestMethod.POST,
       consumes = MediaType.APPLICATION_JSON_VALUE,
       produces = MediaType.APPLICATION_JSON_VALUE
    )
    public Employee addEmployee(@RequestBody Employee employee){
       return employeeService.addEmployee(employee);
    }

    @RequestMapping(
       value = "/updateemployee",
       method = RequestMethod.PUT,
       consumes = MediaType.APPLICATION_JSON_VALUE,
       produces = MediaType.APPLICATION_JSON_VALUE
    )
    public Employee updateEmployee(@RequestBody Employee employee) {
       return employeeService.updateEmployee(employee);
    }

    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
       public Optional<Employee> getEmployeeById(@PathVariable int id){
          return employeeService.getEmployeeByID(id);
    }

    @RequestMapping(value = "/deleteallemployee", method =
        RequestMethod.DELETE)
    public void deleteAllEmployee(){
       employeeService.deleteAllEmployee();
    }

    @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
    public void deleteEmployeeById(@PathVariable int id){
       employeeService.deleteEmployeeById(id);
    }

}
```

1.8 Testing with Rest Client [ARC]

1.9 Spring Boot Framework Plug-in Eclipse

1.10 Assignments for practice