RESTful Webservices using JAX-RS

Topics

1. RESTful webservice
2. Designing Restful Services
3. Developing & Deploying REST API
4. Content Handling
5. HTTP response & Exception Handling
6. Securing REST Apis
7. Swagger

Pre-requisites

1. Core Java - OOPs, Exception Handling, Collection

Software requirements

1. JDK 17 or later
2. Eclipse or STS
3. Apache Tomcat Server
4. Permissions to download Maven dependencies

Java is an object oriented programming language

OOPS

* Encapsulation
* Inheritance
* Polymorphism
* Abstraction

Encapsulation: Hiding the data (variables) and accessing them only through public methods

User:

* store
* login
* update
* delete

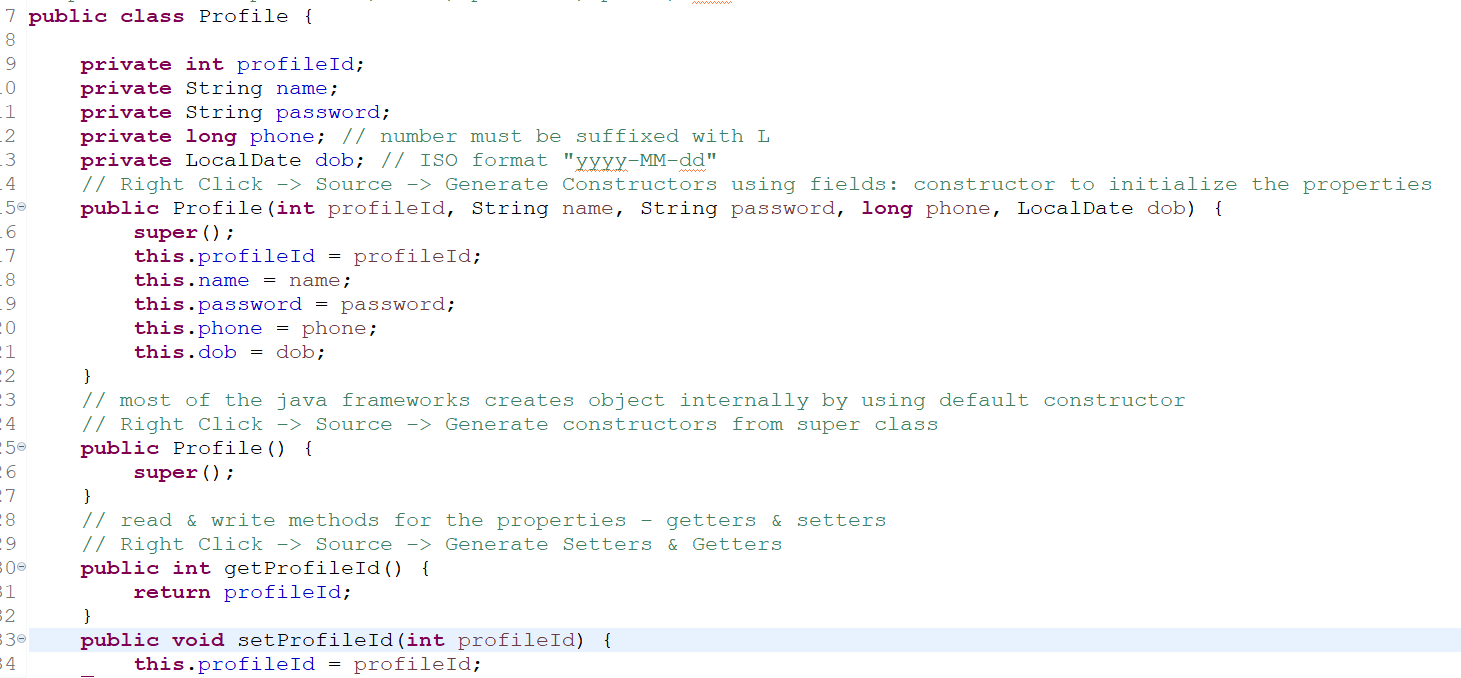
Create a Java Project

Eclipse -> File -> New -> Other -> Type Java project

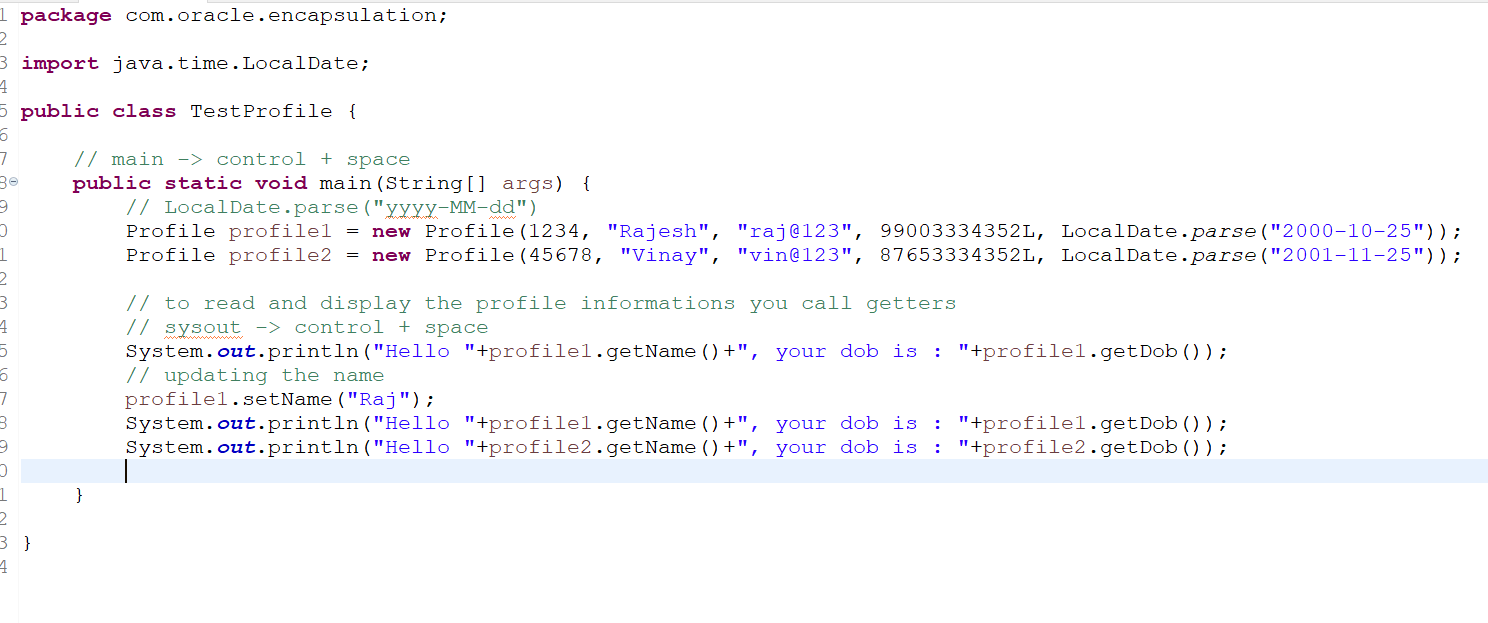
Package: 2 level package structure

com.oracle.dao  
com.oracle.service  
com.oracle.controller

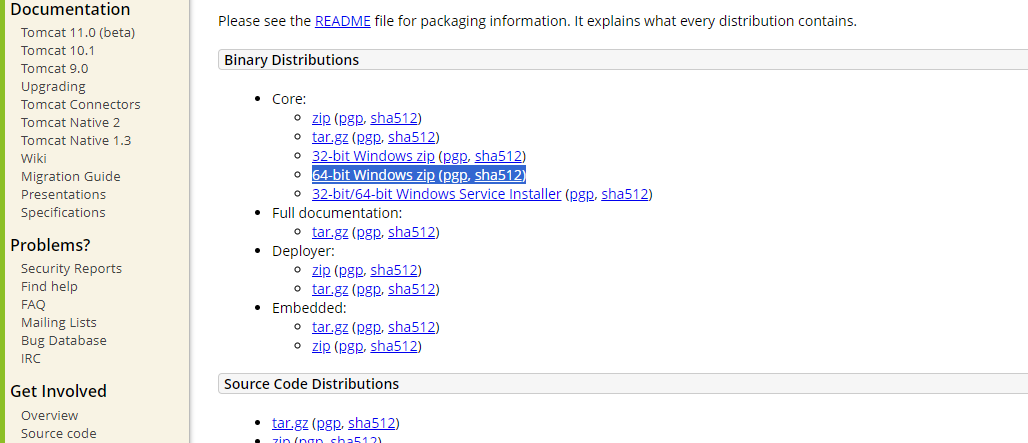
Profile.java



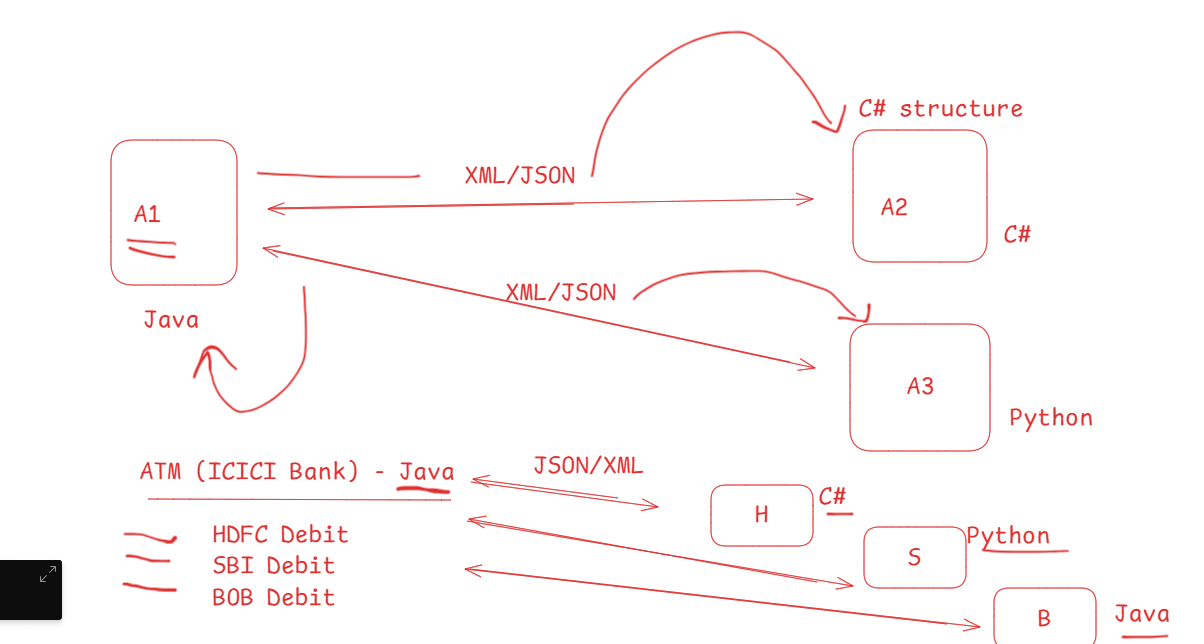
TestProfile.java



Download Tomcat 10 zip & unzip the file



Webservice

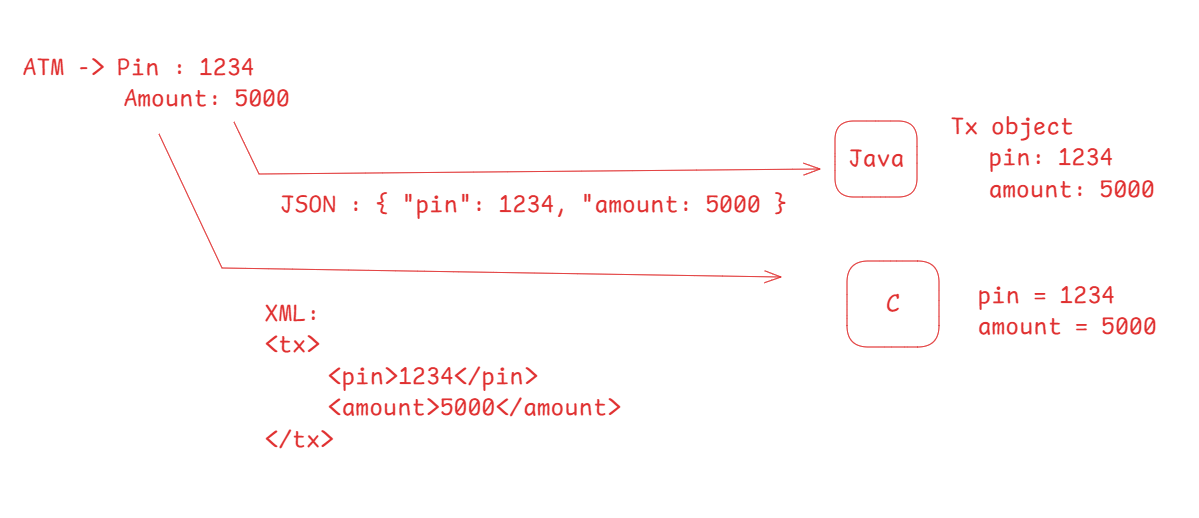


Webservice makes applications written in different technologies to exchange the data in a common format like XML/JSON, these common formats are converted to the respective structure the application is written in

Ex:

Zomato (A) -> Phone Pay(B) ->   
 -> Google Pay (C)  
 -> Simpl (D)

Webservice can use different datastructure also for communication, however JSON/XML more preferred, in that 99% of the time JSON is preferred because its light weight.



Types of Webservices

1. SOAP Webservice (Simple Object Access Protocol): It is XML based & it is a old webservice before RESTful this was the webservice used
2. RESTful Webservices (Representational State Transfer): It can share the data in various formats like XML, JSON, CSV, Text, HTML and so on

Java has provided specifications for both SOAP & REST

1. JAX-WS: Soap webservice
2. JAX-RS: Rest webservice

Important rules to consider while designing APIs with REST architecture

1. Locate the Web Services using URL
2. Use HTTP methods to access and define the service
   1. HTTP GET: It must be used for reading/fetching the data
   2. HTTP POST: It must be used for creating the new resource
   3. HTTP PUT: It must be used for updating the existing resource
   4. HTTP DELETE: It must be used to delete the resource

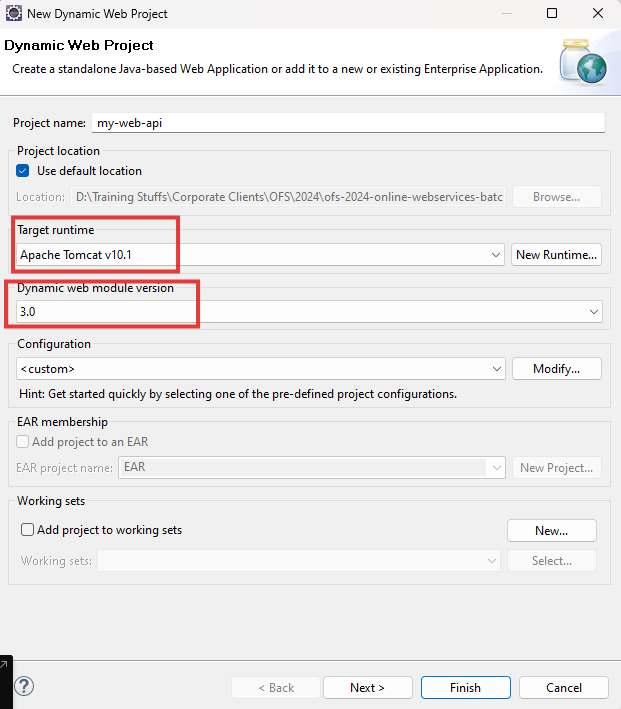
JAX-RS has various implementations like

* Jersey
* Rest Easy
* Apache CFX

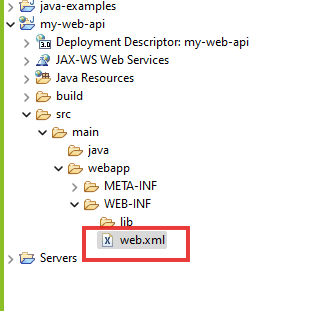
JAX-RS has provided various annotations to develop the webservices which can be used with any implementations like Jersey, Rest Easy, Apache CFX

@Path, @GET, @POST, @PUT, @DELETE, @Produces, @Consumes, @PathParam, @QueryParam, @Context

Create dynamic web project

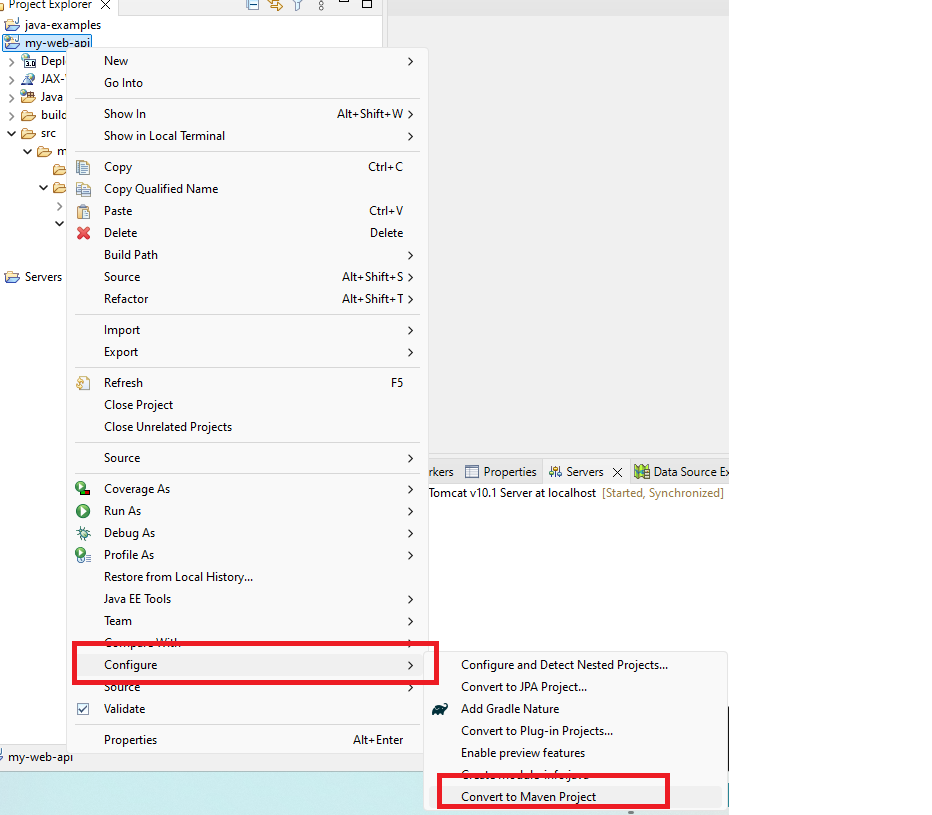


Generate deployment descriptor



Maven Project: It helps you to add java libraries and their dependencies with the help of XML file, which is a pom.xml file

<dependencies>  
 < dependency >…</dependency>  
 <dependency>..</dependency>  
</dependencies>



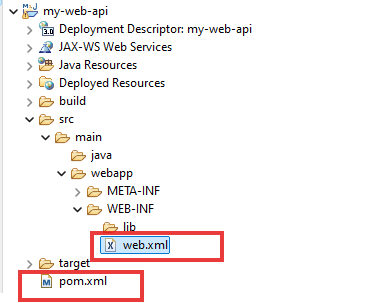
Dependencies for Jersey Implementations

1. Jersey Server
2. Jersey container servlet core
3. Jersey HK2
4. Jersey Media JSON Binding

pom.xml



Project Structure



There are 2 xml files

1. pom.xml: This is a maven configuration file, where you will add project dependencies
2. web.xml: This is a project configuration file, where server looks for request mapping

Initial Setup

1. Front Controller configuration - web.xml
2. Creating Controllers: This can handle the requests

Note: Webservice code will be written in the controller

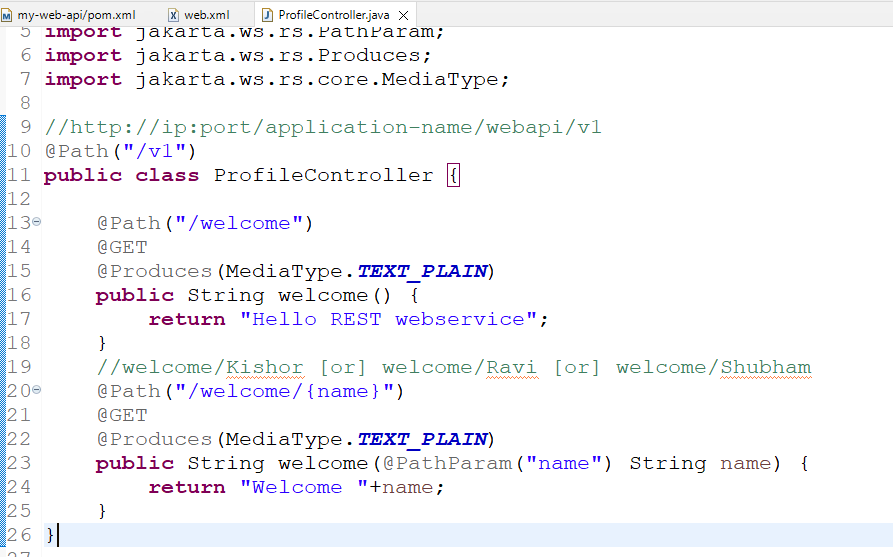
web.xml



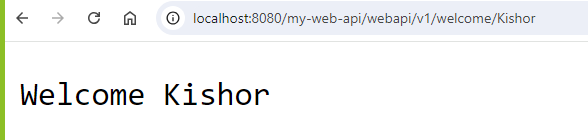
Front Controller takes incoming requests that has /webapi/\* means any sub-path after /webapi/, we must have RESTwebservices configured with a URL, front-controller takes the request & routes to the webservice

Note: You must create your webservice inside the com.oracle package, because <param-value> says the package name is com.oracle

Note: Webservices code are written in the controller layer



Output:



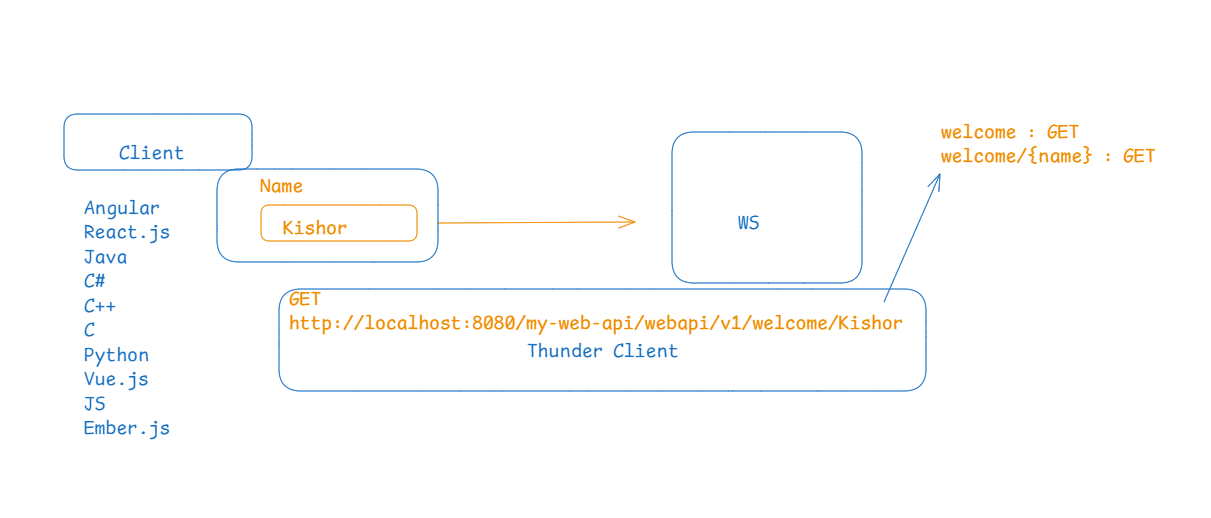
@Path: It is to configure the URL for the resource

@Produces: It is used to produce the data to the clients

@Get: It is used to map the resource with HTTP GET

Tools to test webservice

We cannot test all the HTTP methods using browser, hence we must use some testing tools for webservice like Postman, VS Code Thunder client.



Activity:

Try to create few more methods that gives welcome <<yourname>> in text format with other HTTP methods like post, put & delete, use the same url for all the methods

ex: /welcome/{name}

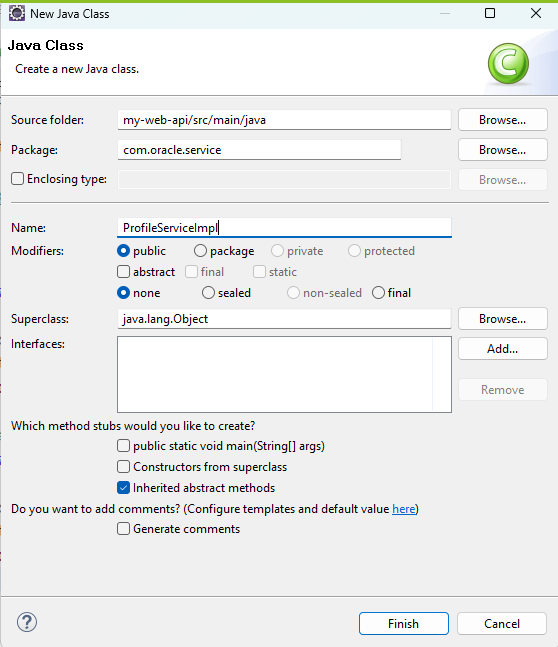
Summary:

1. Types of webservices
2. RESTful webservices - HTTP protocols, HTTP methods, URL
3. Annotations: @Path, @Produces, @GET, @POST, @PUT, @DELETE
4. Front Controller -> routes -> Controller (Webservice)
5. MVC Architecture

Implementing the Service layer

Service layer is going to have business logics, this will be invoked by the controller which is a webservice & service layer invokes the logics written in the DAO layer where the Database logics are written

Create a class ProfileServiceImpl in a com.oracle.service package



Implementing the DAO layer

Create a class ProfileDaoImpl inside a package com.oracle.dao

We are using Collection Framework to maintain the data, because it is dynamic in nature and it provides inbuilt methods to add, remove, iterate the data

ex:

ArrayList<String> al = new ArrayList<String>(); // this can maintain string objects

List<String> al = new ArrayList<String>(); // valid statement

List<Profile> items = new ArrayList<Profile>();

Collection APIs are present in java.util package, hence you must import them

Create a Profile class in com.oracle.beans whose objects we are going to store

List of things we are going to create

1. ProfileController ( already present)
2. ProfileServiceImpl
3. ProfileDaoImpl
4. Profile - properties like profileId, name, password, phone, dob

Profile.java



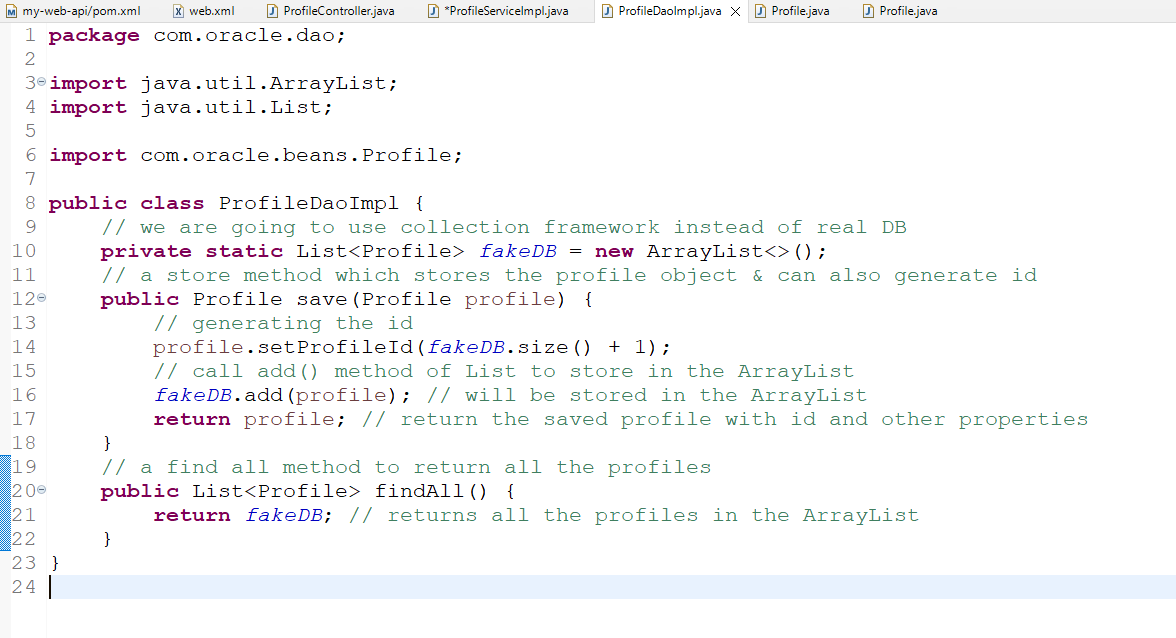
Now we must perform CRUD operations in the DAO layer, however we are going maintain the data not in the DB instead Collection

ProfileDaoImpl must have following methods

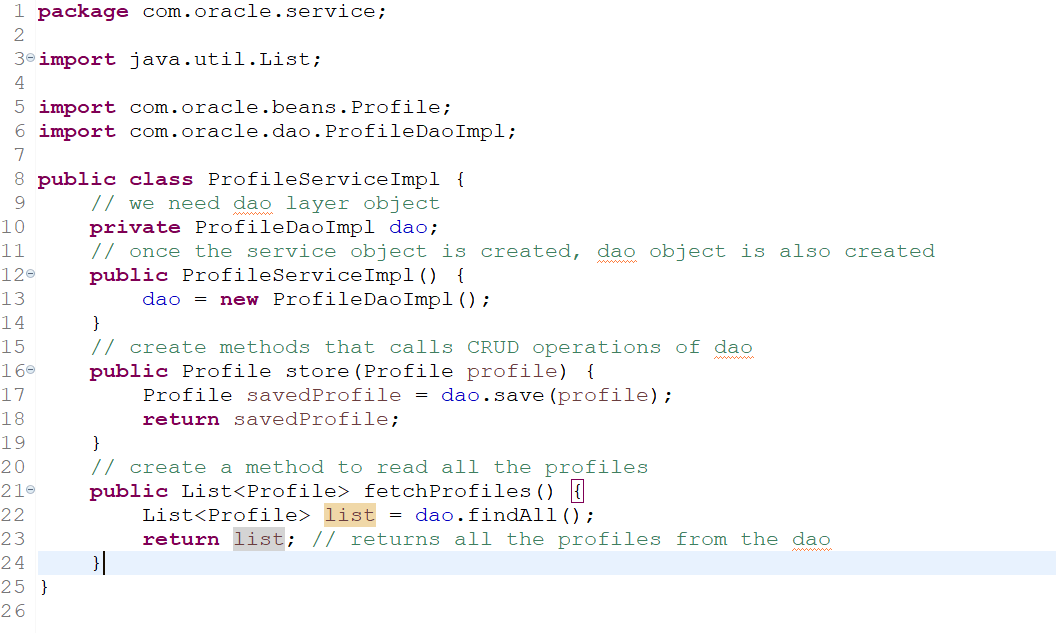
1. store(Profile p): takes profile object & stores in the db or collection
2. findAll(): returns all the profiles from the db or collection
3. findById(int profileId): return a profile that matches to the profileId or return null
4. delete(Profile): deletes the profile based on the profile id
5. update(Profile): update the profile properties using the profile id

Initially we will implement store & findAll in the DAO layer

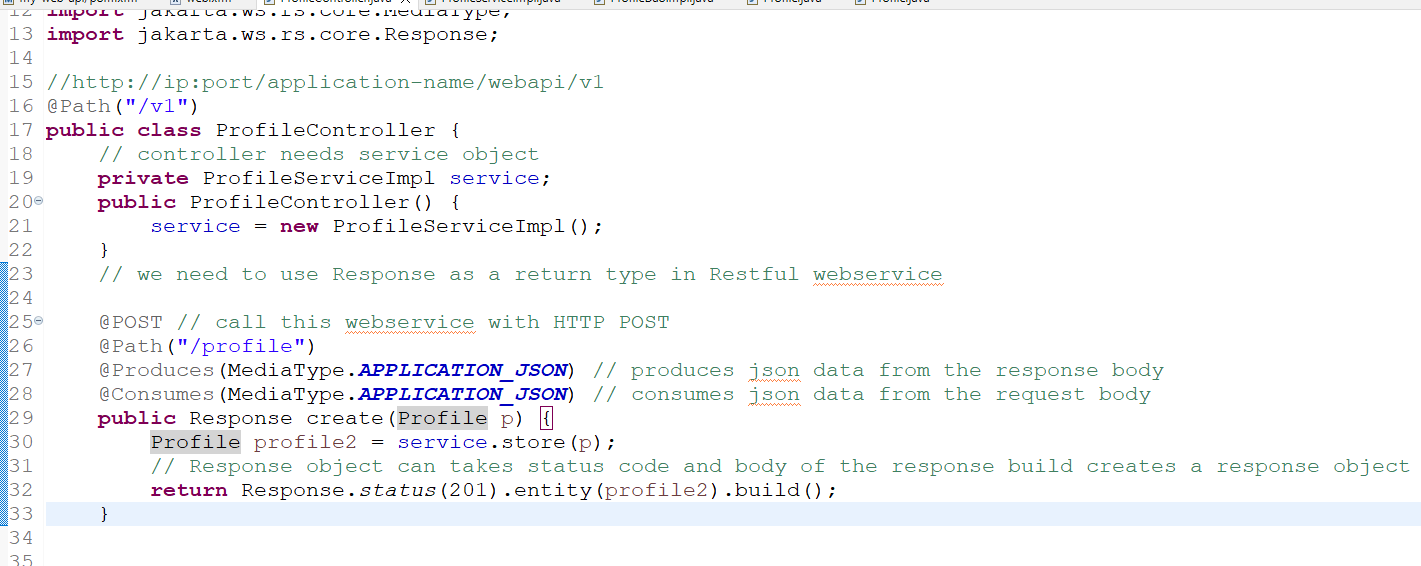
ProfileDaoImpl.java



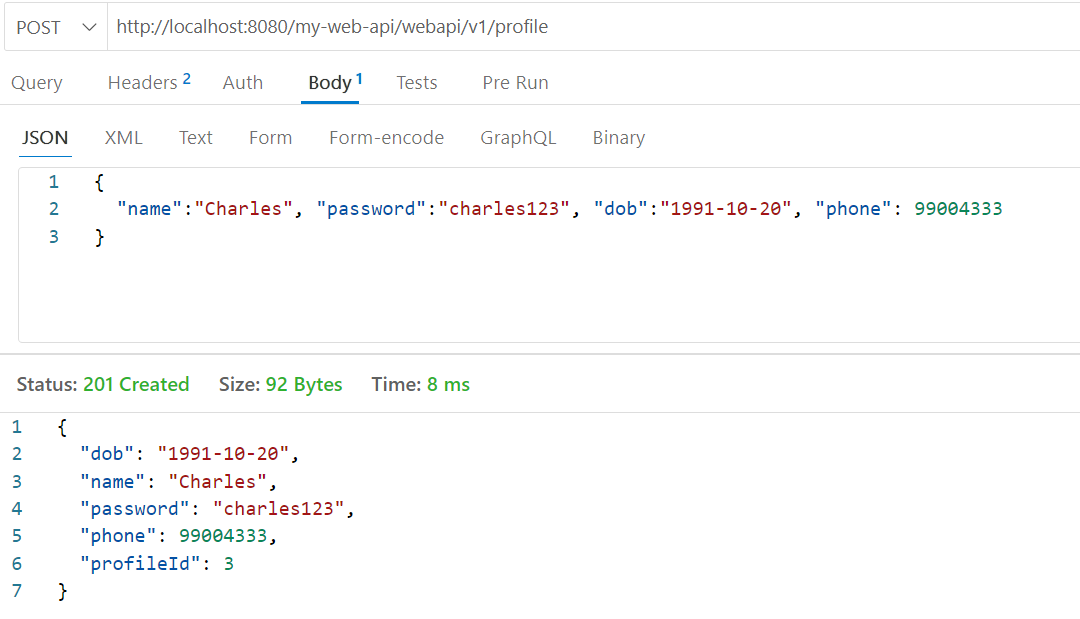
Implementing the service layer to invoke dao layer



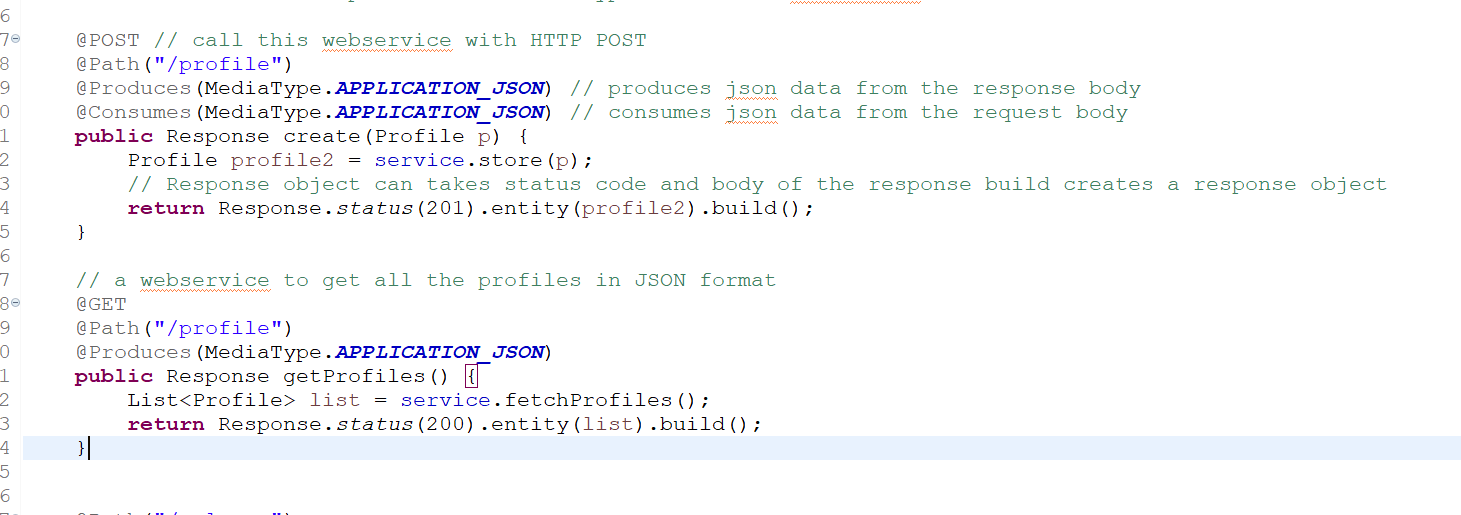
Implement the controller to call the service layer



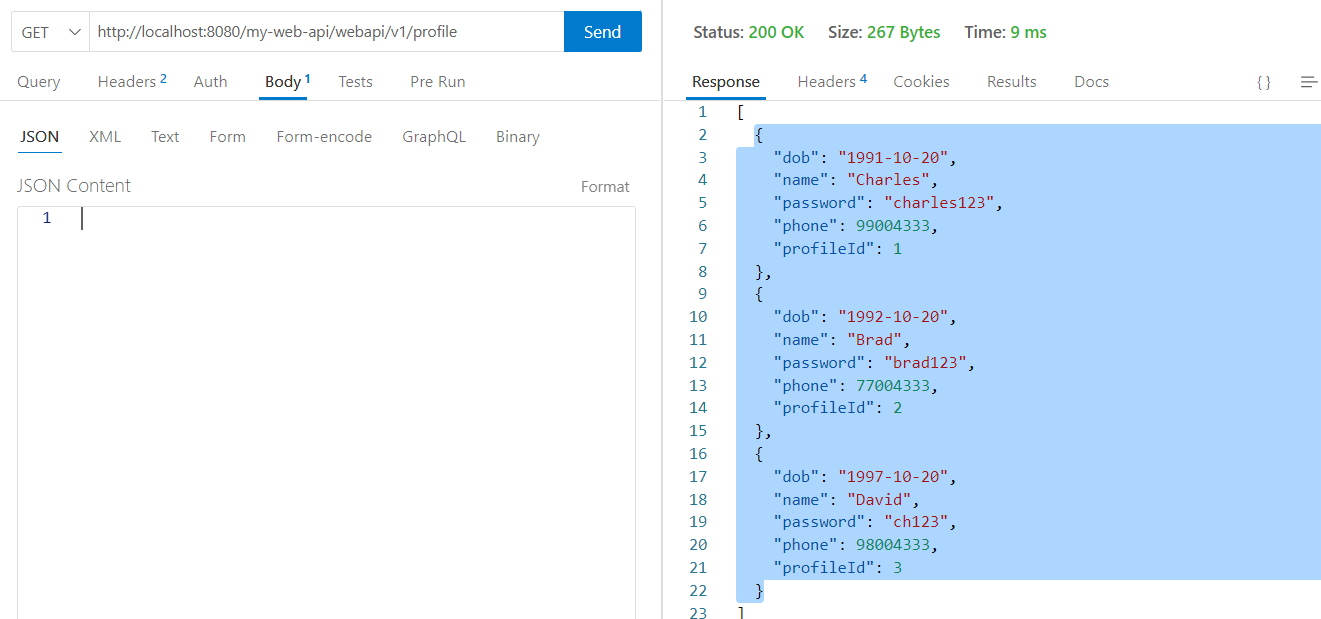
Run the application on the server



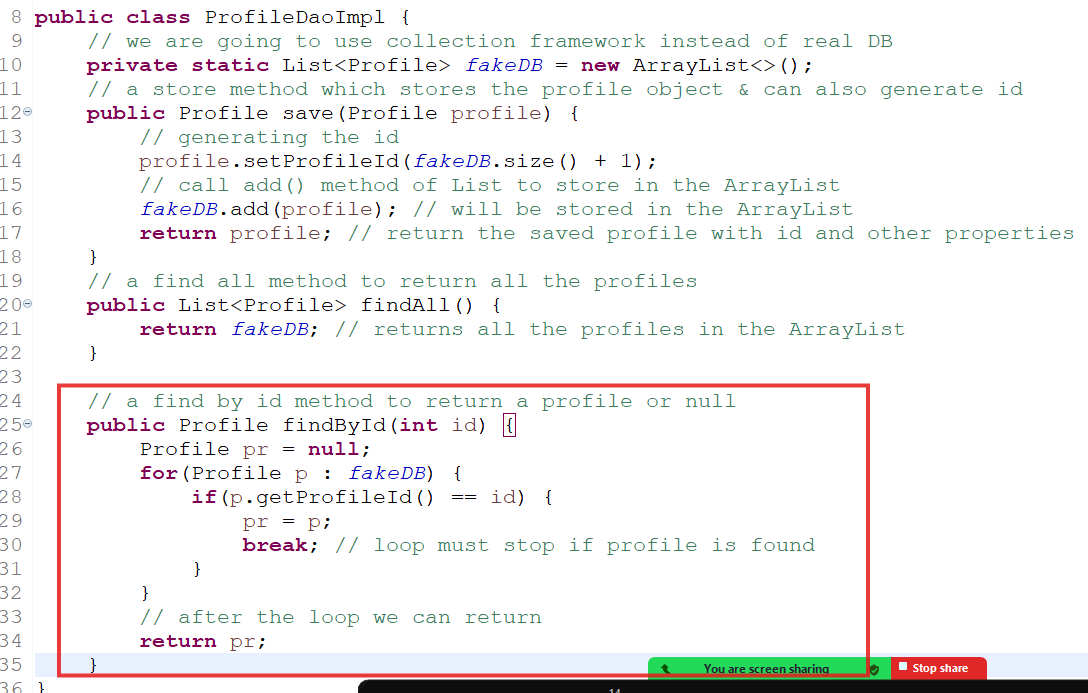
Webservice to read all profiles



Output:



Another method in the DAO that takes the id and returns the profile



Note: When you have null, there could be possibility of NullPointerException, if exception occurs then program will stop, hence we need to handle it and tell the user what went wrong

Exception Handling in RESTful webservice

In Restful webservice we must register our own exceptions so that whenever such exception occurs an response with an error message will be automatically created by the Jax-RS

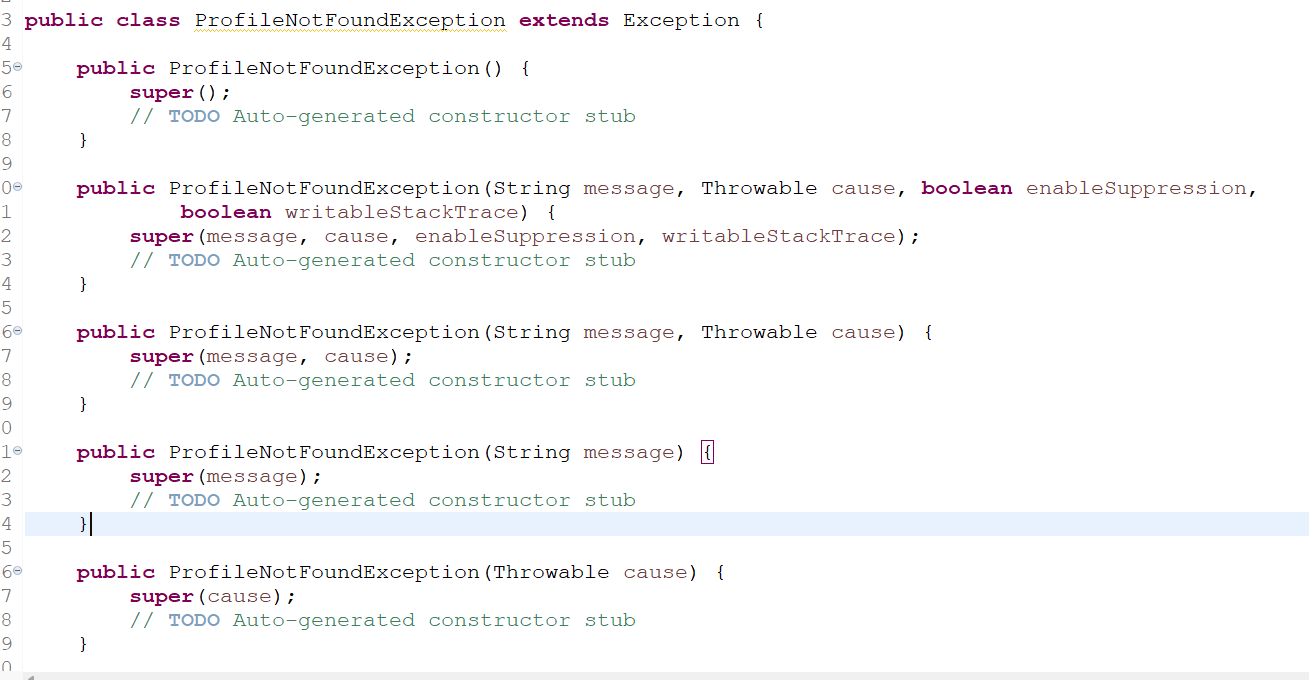
Steps:

1. We must create our own exception class by extending an exception class

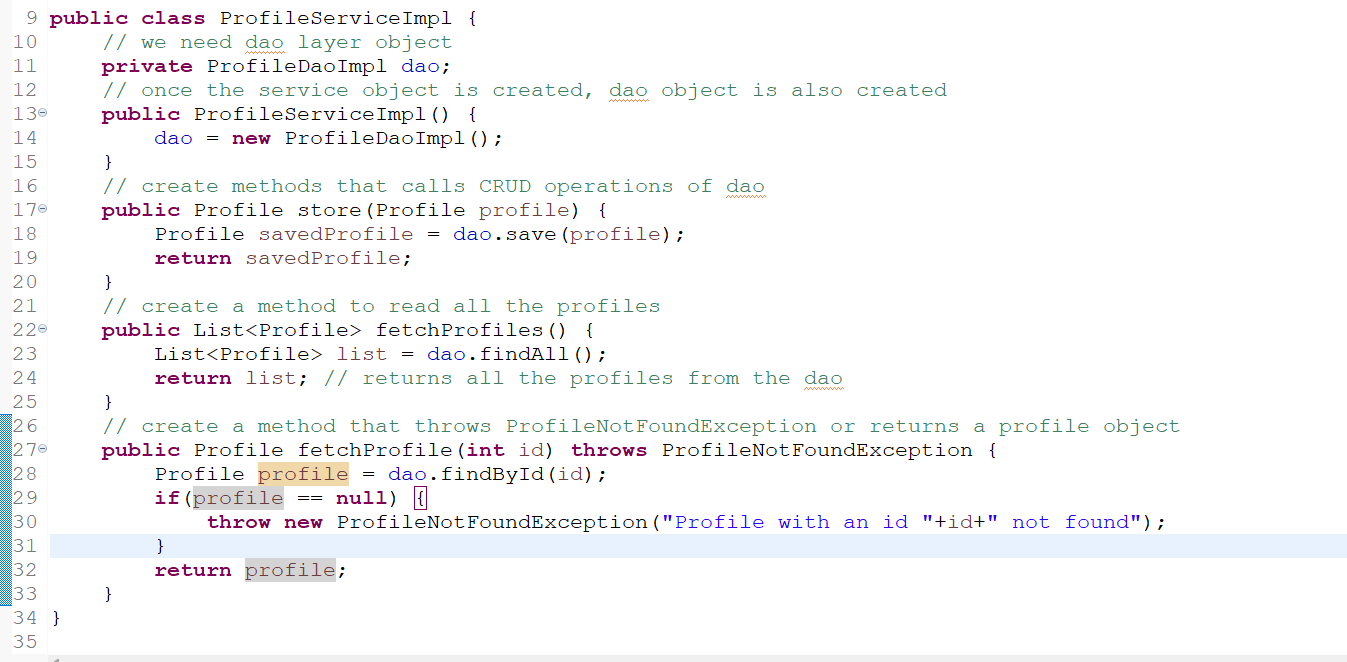
ex: class ProfileNotFoundExcepiton extends Exception

1. We must generate this exception when certain condition is met
2. We must register this exception by using ExceptionMapper that is going to be the provider
3. If there’s an exception then JAX-RS can produce an error response

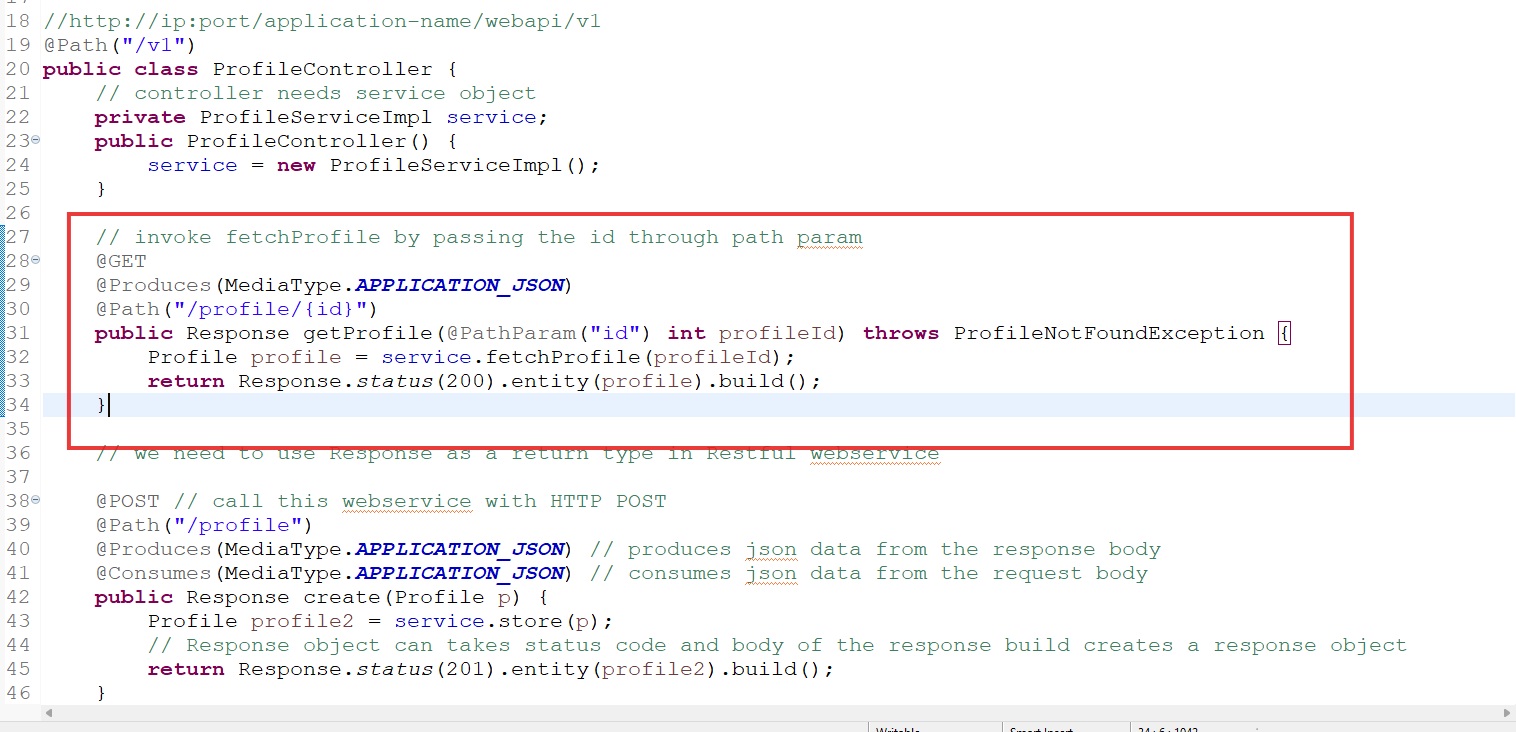
Create a class ProfileNotFoundException in a package com.oracle.exceptions, extend that class with Exception (Predefined exception class in Java)



ProfileServiceImpl.java

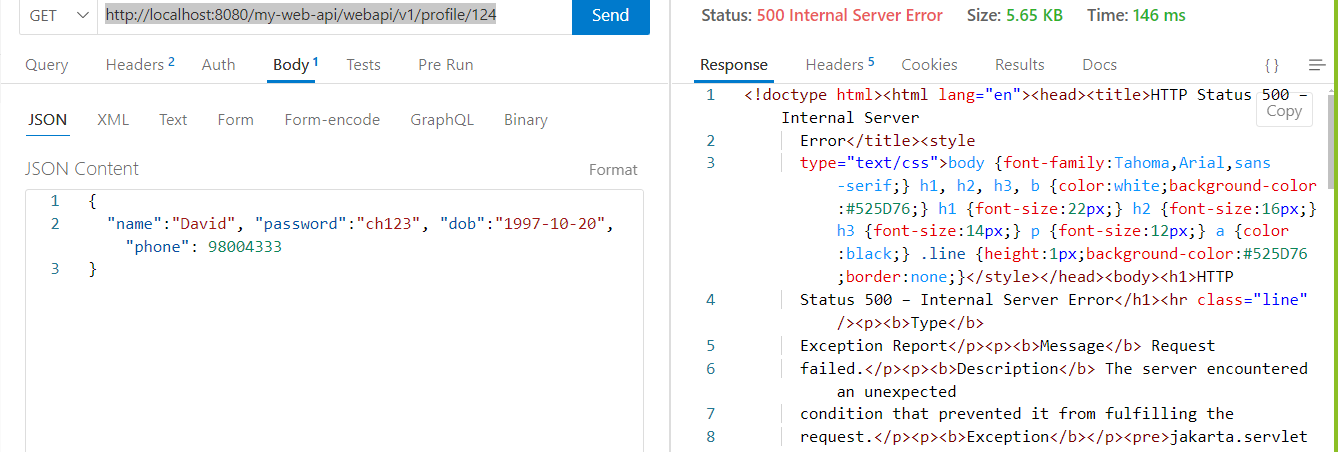


Invoking this method in the controller



Here we are not handling the exception instead propagating to the caller, it will be handled by JAX-RS, since ProfileNotFoundException is not registered you will get an error page instead of a json error response

When you send request you get an HTML error response which is not understood by the user



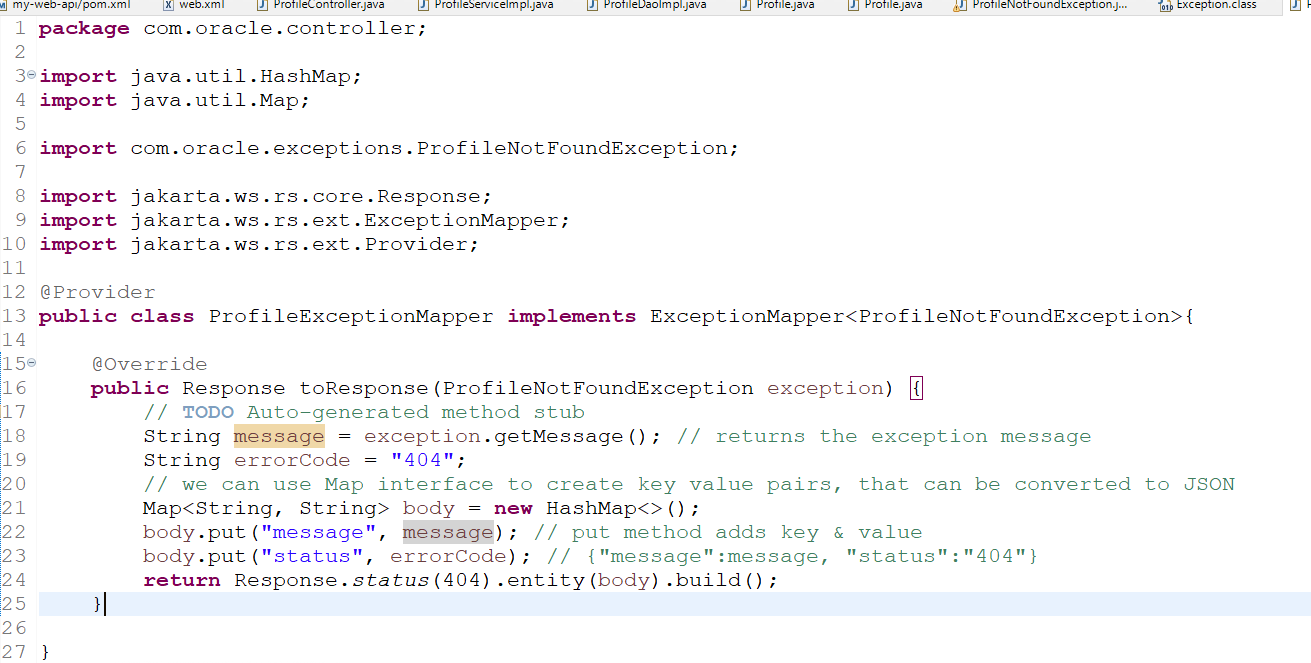
What is the solution

We need to register this exception called ProfileNotFoundException in JAX-RS so that it can handle it and send a JSON response instead of HTML response

i.e,

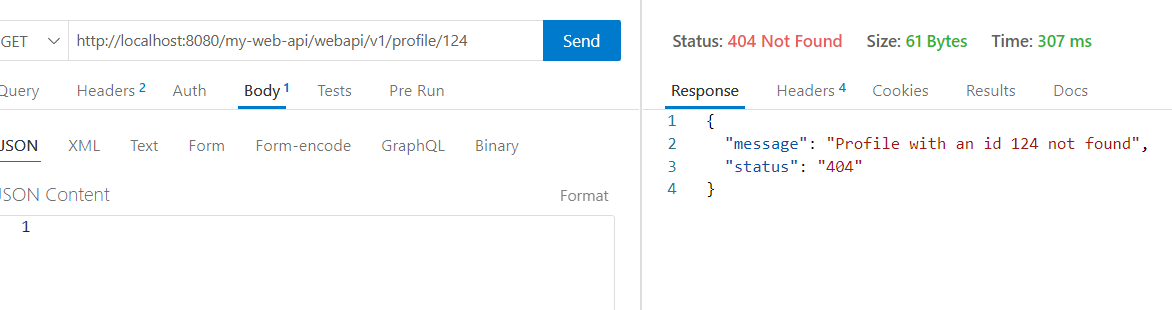
{   
 “message”: “Profile with an id 124 not found”  
 “status” : 404  
}

Create an exception mapper that can recognize ProfileNotFoundException, we must use @Provider on top of the class to register anything in the Jax-RS

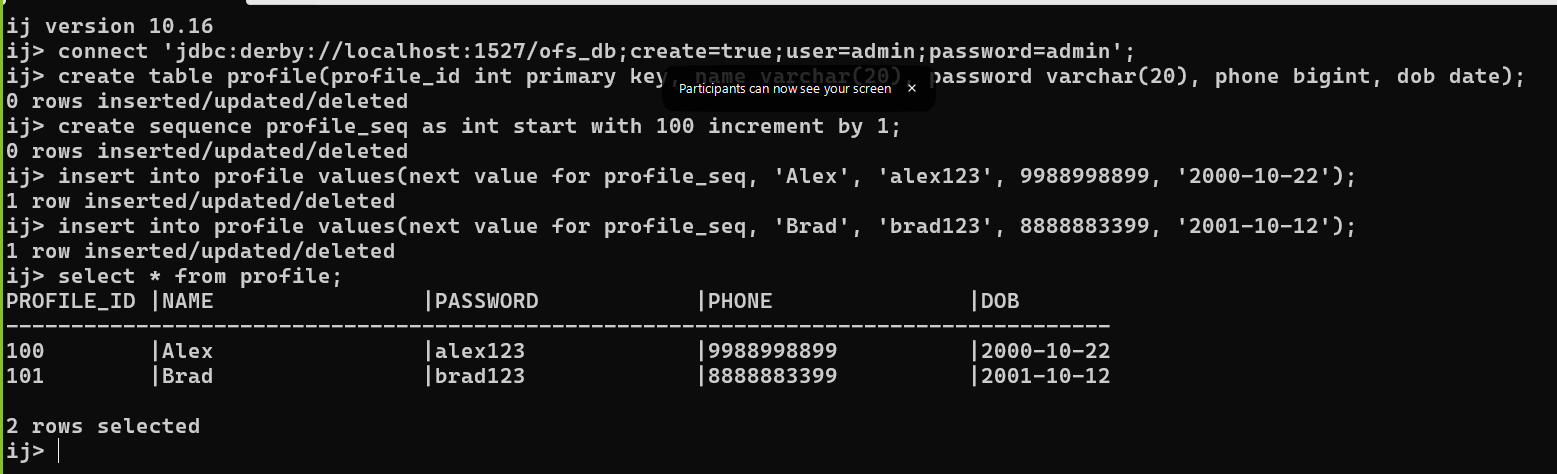


Now if ProfileNotFoundException occurs you get an error message in the JSON format instead of HTML format

Output:



Try to store two records & call the webservice by passing the id and check if you are able to see the profile or not

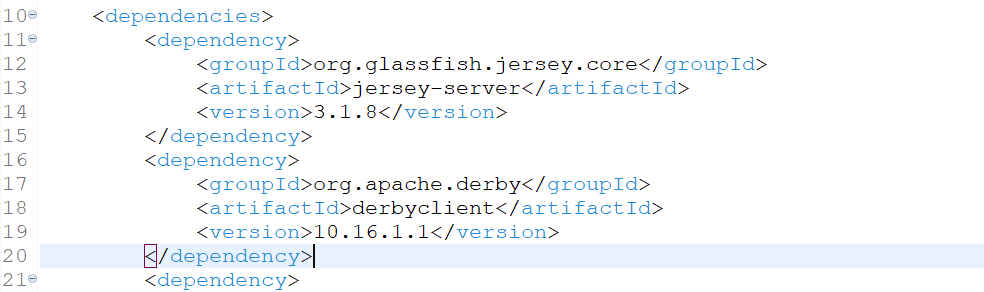


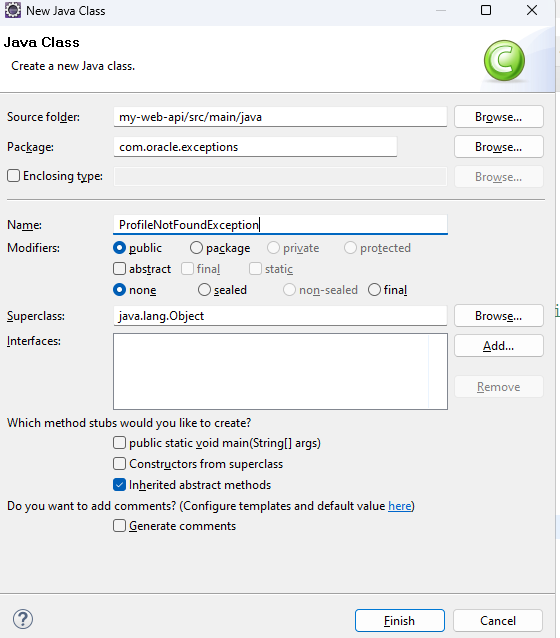
Now we must use JDBC program in our Java application to interact with any database

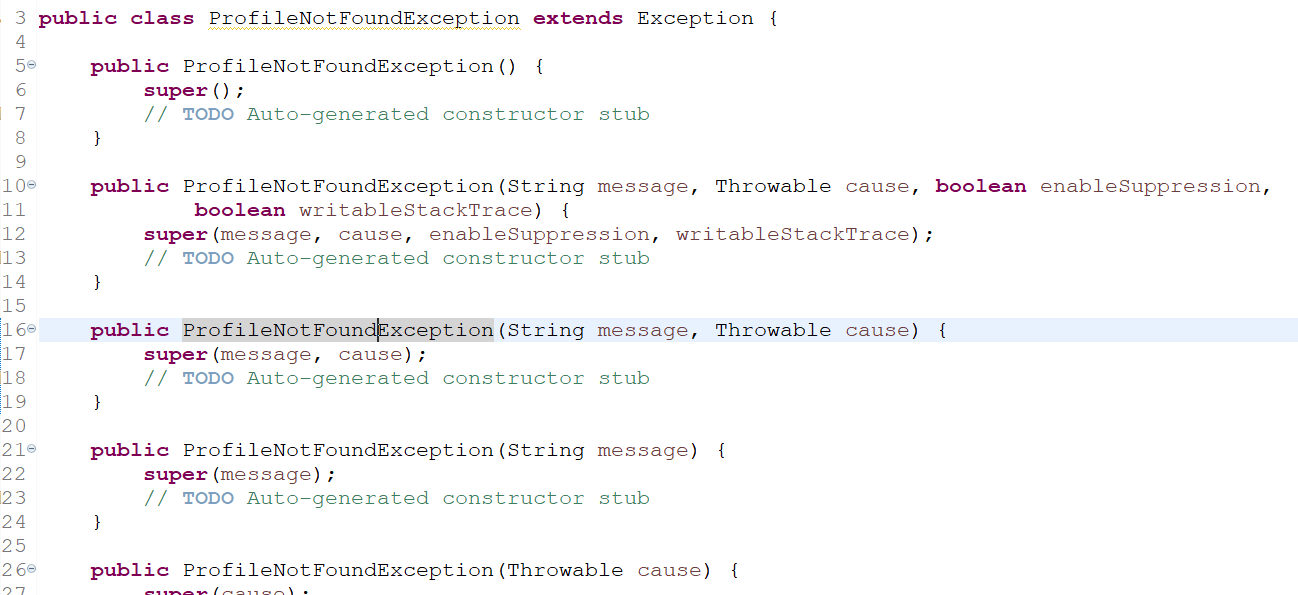
Steps

1. Install derby-client library in your project through maven
2. Follow the JDBC steps to interact with the DB like Connecting, Statements, ResultSet

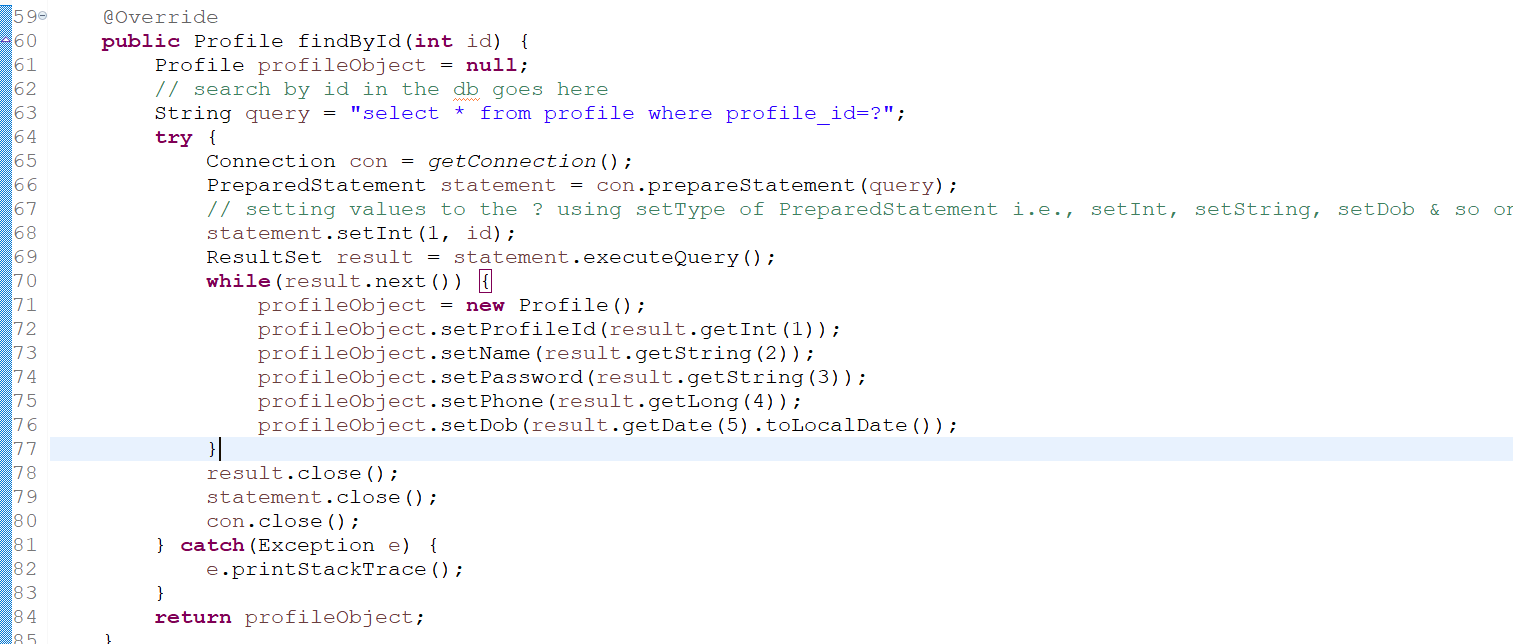
pom.xml



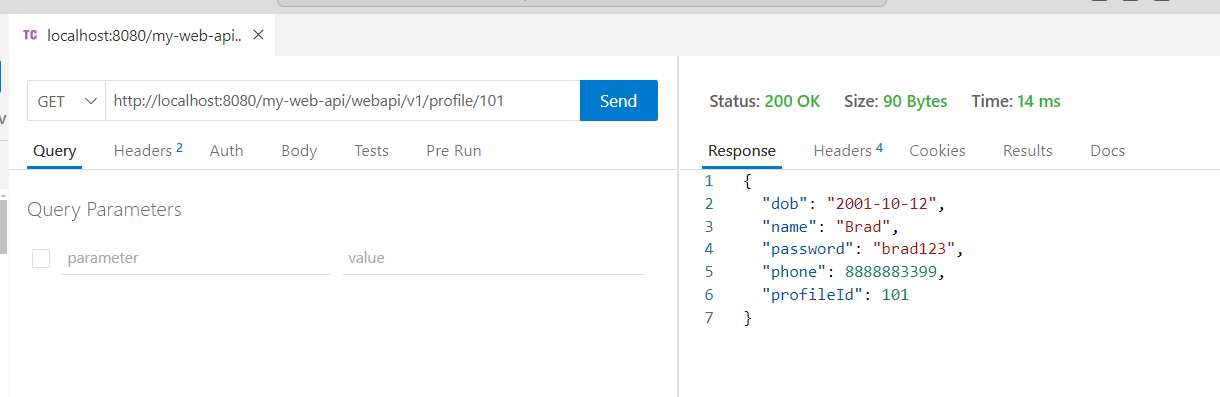




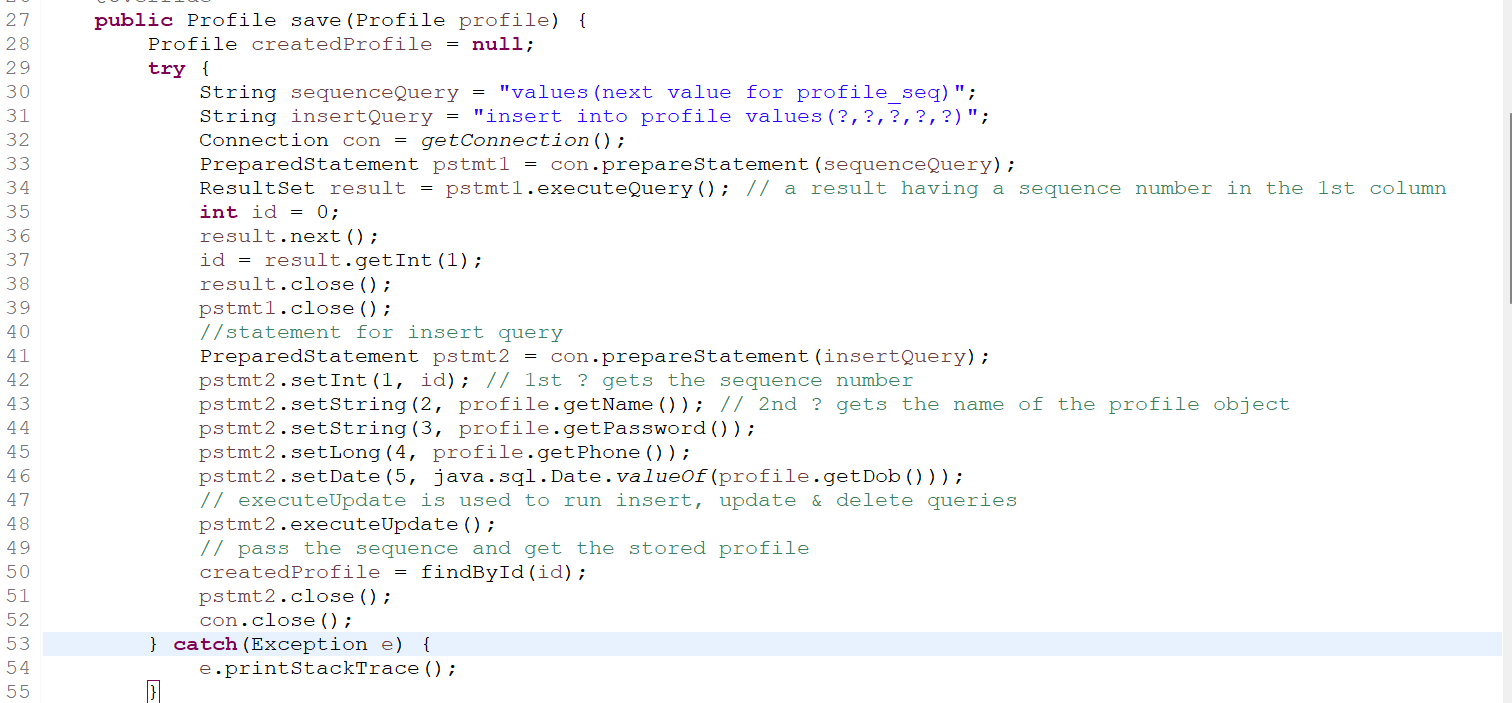
Dao layer having JDBC code



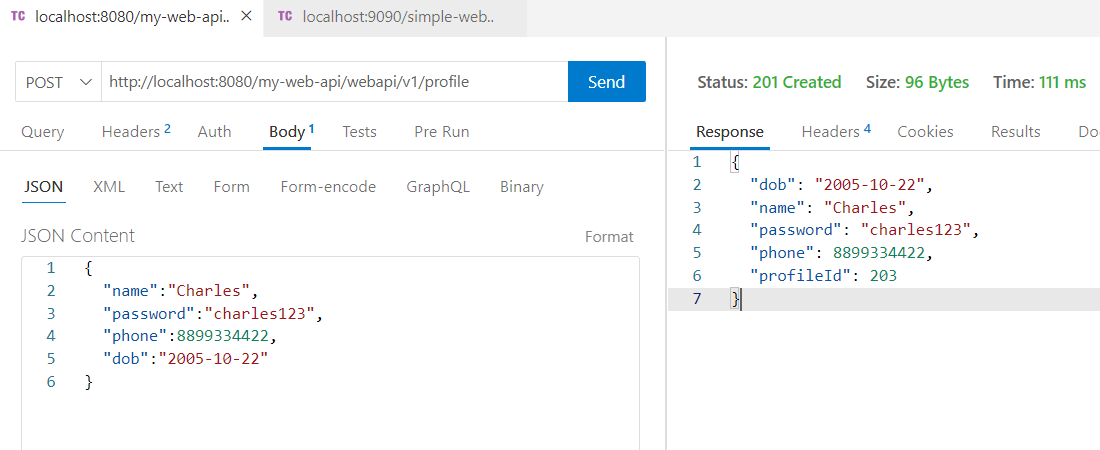
Output:



Implementing save method of DAO



Output:

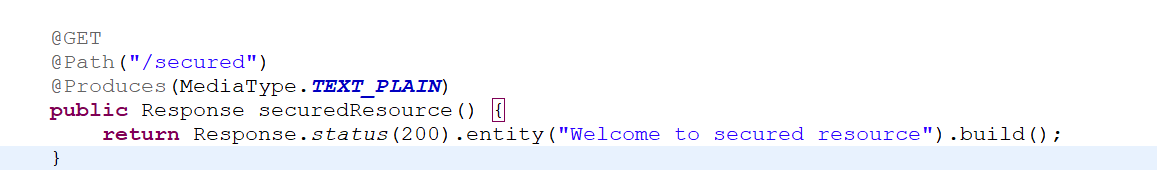


JAX-RS Security

Whenever you access any webservices it must be an authenticated and authorized request, if not the request must be denied.

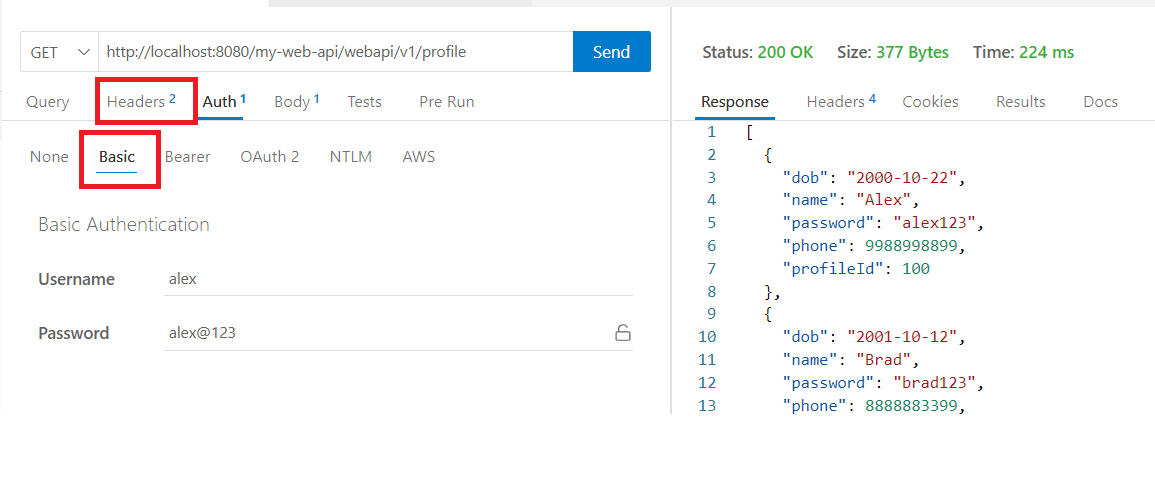
Authentication: Identifying the user is right or wrong like username & password

Authorization: Whether the user has permission to access the resource





While sending the request you must send the Authorization header



Completing the CRUD operations on profile

We have created webservices for Post & Get, however we still need to create webservices for Put & Delete

We need to have update & delete operations which can work on profile table, hence in the DAO layer we must have the methods to perform update & delete.

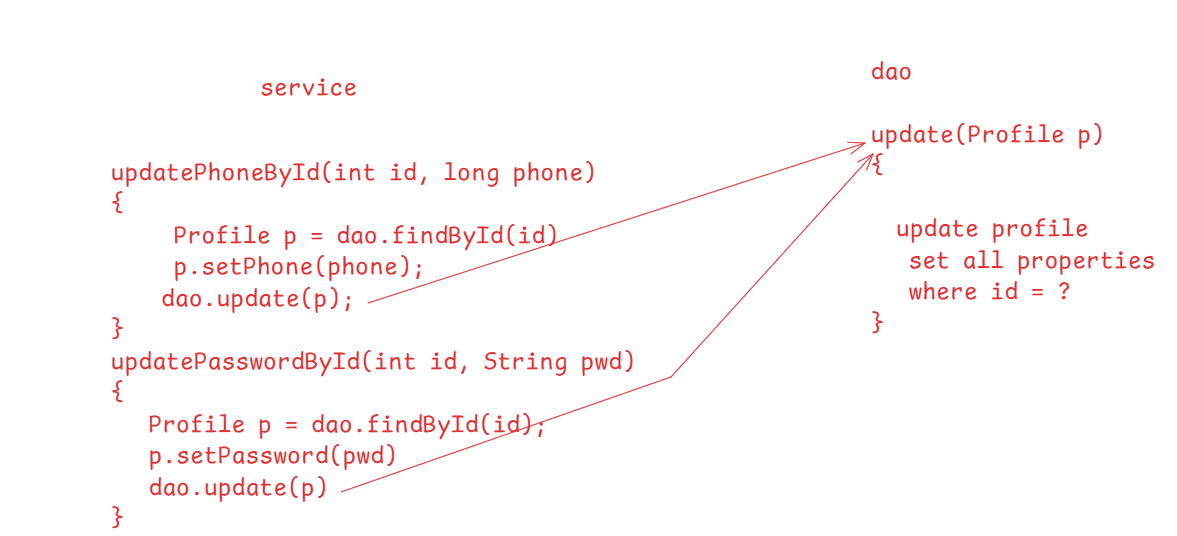
ProfileDao

Existing methods: save, findAll, findById

New methods: update, deleteById

Note: We must implement all the 3 layers i.e., controller, service & dao

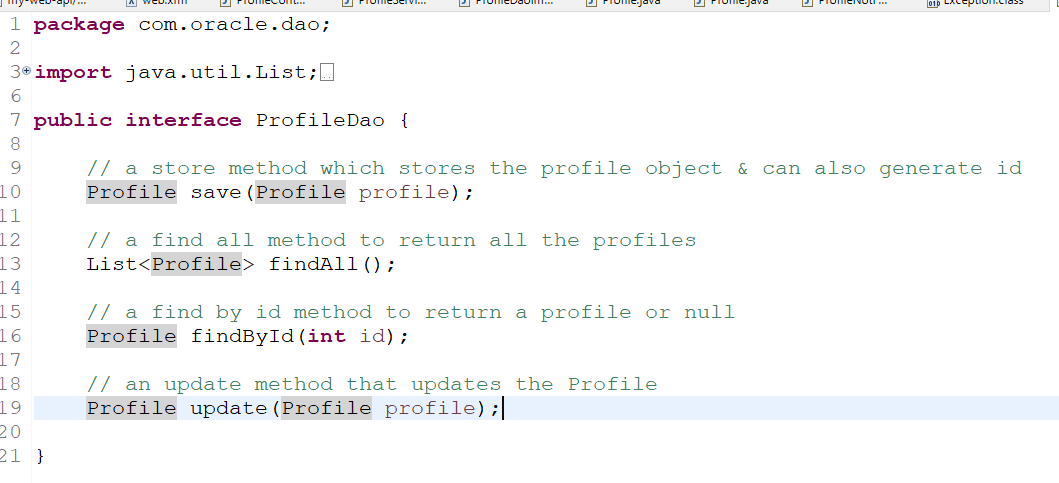
In Dao we will have a single update method called by service layer



Steps

1. create update(Profile p) in ProfileDao
2. Implement update(Profile p) in ProfileDaoJdbcImpl
3. create update phone, password, by profile id in the Service layer
4. create controller that calls service layer methods to update profile properties
5. Test those operations from the VS code

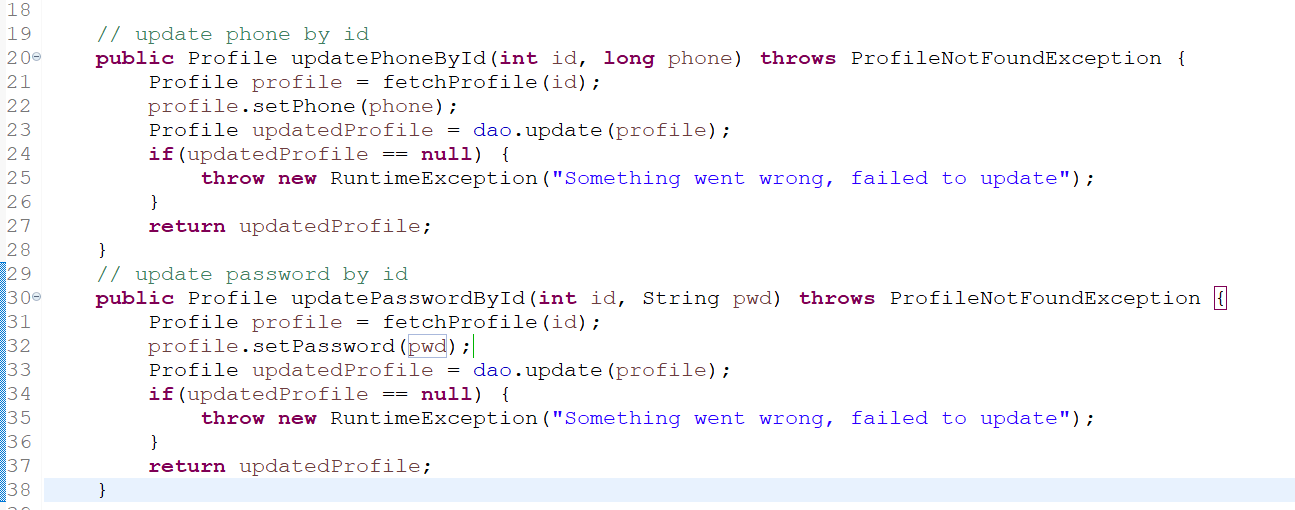
ProfileDao.java



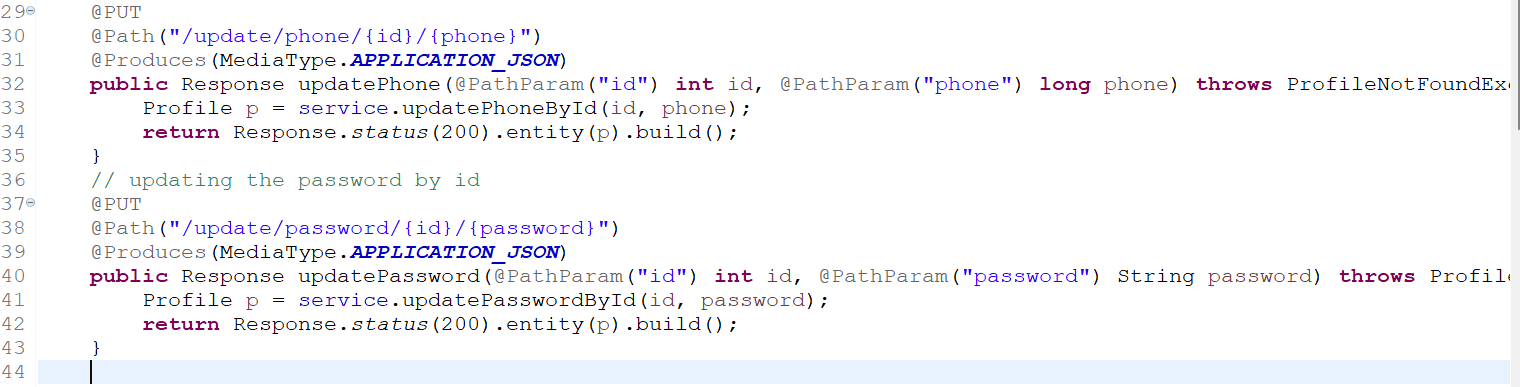
ProfileDaoJdbcImpl.java



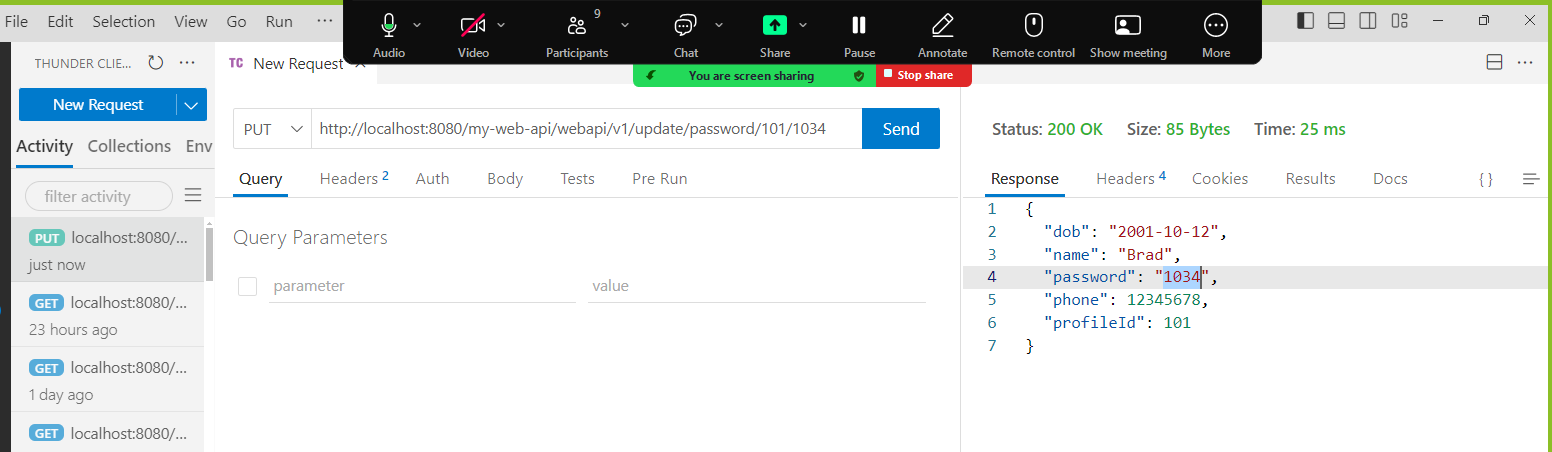
ProfileServiceImpl.java



ProfileController.java



Output:



Swagger

It is a tool that documents the API you design in good look & feel, so that you don’t have to remember the API end points, developers can share the URL of the swagger dashboard to others so that they will know what are the end points of the API in an understandable way.

JAX-RS annotations vs Spring REST annotations

|  |  |
| --- | --- |
| JAX-RS | Spring REST |
| @GET | @GetMapping |
| @Path | @RequestMapping |
| @Path(/url), @GET, @POST, @PUT, @DELETE | @GetMapping(/url), @PutMapping(/url), @PostMapping(/url), @DeleteMapping(/url) |
| @Consumes | @PostMapping(consumes=json) |
| @Produces | @PostMapping(produces=json, consumes=json) |
| @Path: To configure the webservice class  @Path class ProfileController { } | @RestController to configure the webservice class  @RestController  class ProfileController { } |

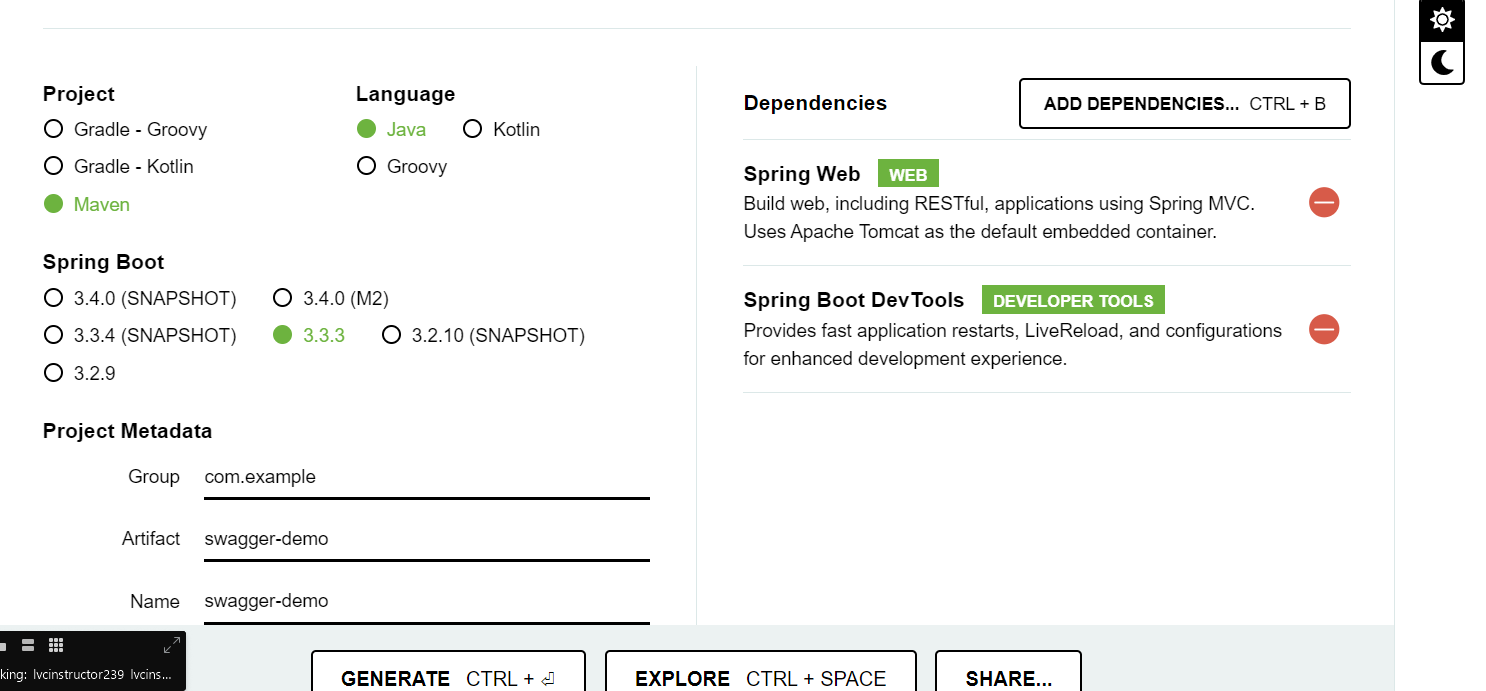
Spring has a website which allows you to create a project template & also gives you an option to add maven libraries

Note: Spring project comes with inbuilt tomcat server that runs by default in 8080 port

Note: Spring project will have main method to run the project, it automatically runs in the inbuilt server

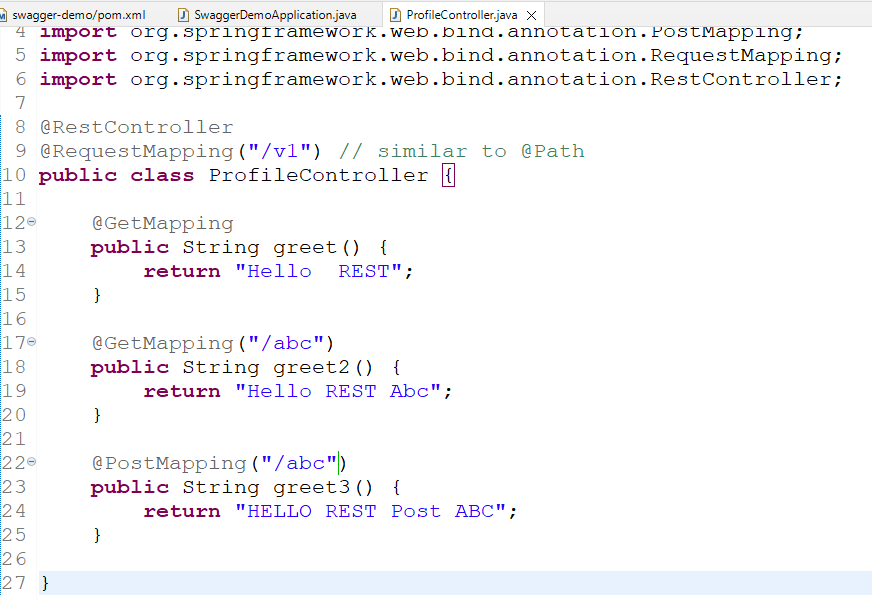
Dependencies

1. Spring Web
2. Devtools
3. Spring Doc Open API Starter WebMVC ui: this for swagger, you must get this from the maven repository



Open pom.xml and add the below dependency





spring provides application.properties file where you can configure application related properties like swagger url, server port, and so on.

src/main/resources/application.properties



Run the main method

Output:

