Webservices

Topics

* Webservices overview
* Introduction to REST
* Introduction to Jax-RS
* Resources & requests
* HTTP entities with XML & JSON
* Handling responses

Pre-requisites

* Java
  + OOPS
  + Collections
  + Exception Handling

Software requirements

* Java 17 or later
* Eclipse IDE for Web & Enterprise Developers
* Tomcat 10
* Visual Studio Code

Webservices

These are online resources or services which are made available over the internet so that applications written in different technologies can exchange the data

There are two types of webservices

1. SOAP webservice = Simple Object Access Protocol
2. REST webservice = Representational State Transfer

SOAP vs REST

|  |  |
| --- | --- |
| SOAP | REST |
| It exchanges data in XML format only, it is a old webservice before REST | It can exchange the data in various formats like XML, JSON, CSV, TEXT, HTML and so on |
| It uses SOAP protocol, that will have set of rules how the XML structure should be | It doesn’t use any protocol for the structure, however it uses HTTP protocol to transfer the data |

Note: JSON is the most widely used datastructure compare to XML, because JSON is light weight and easy to parse

REST webservices

It is a webservice that can share the data between the applications in various structure like XML, JSON, Text, HTML, CSV

It uses HTTP methods to specify the webservice operations so that consumer & producer will know what operation they are doing, there are following HTTP methods

1. GET: fetch/read
2. POST: store/create
3. PUT: modify / update
4. DELETE: remove/delete

Important points while creating/accessing REST webservices

1. URL: To locate the webservcies
2. HTTP Methods: To specify what kind of operations the consumer & producer wants to do

How to create webservices in Java

Java has a standard specification to create webservices which is Jax-RS, this provides various annotations you can use to create webservices, however you must have an implementation of Jax-RS to create webservice, below are the implementations of Jax-RS

1. Jersey - Eclipse
2. RestEasy - Jboss
3. Apache CFX - Apache

JAX-RS annotations

1. @Path: To specify the webservice URL
2. @GET: To specify the HTTP method get
3. @POST, @PUT & @DELETE: to specify different http methods
4. @Produces & @Consumes: Specifies what kind of data a webservice can produce/consume

How to create our first webservice

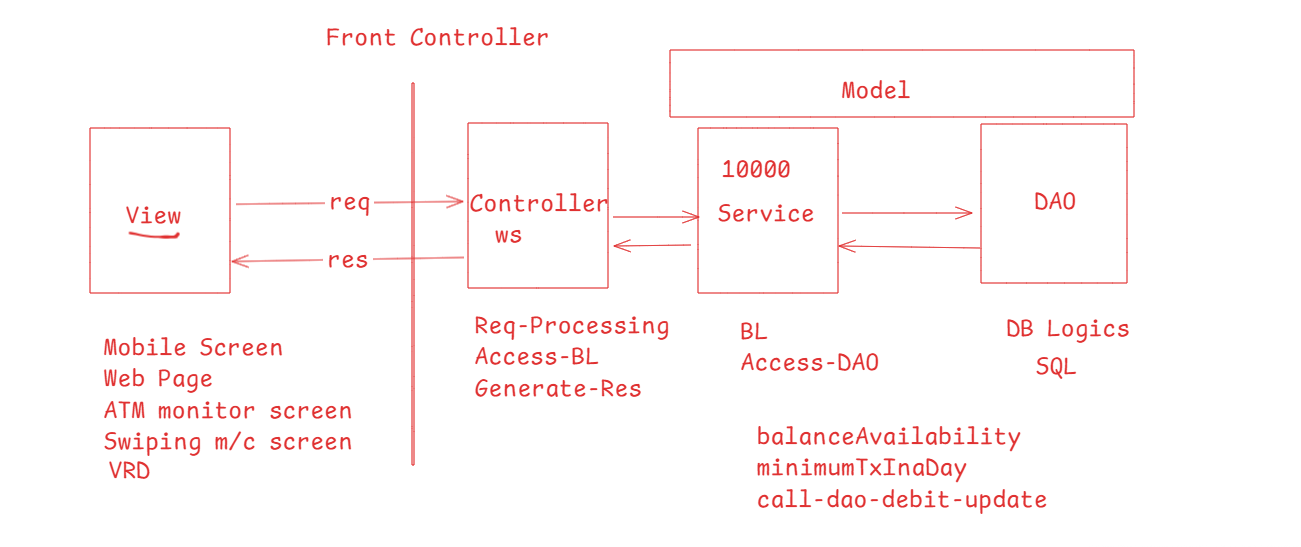
We must create a dynamic web project and configure a server & then convert into maven project(download libraries - pom.xml)

List of libraries we need for webservices using jersey

1. jersey-server
2. jersey-container-servlet-core
3. jersey-hk2
4. jersey-media-json-binding

Every application follows an architecture called as MVC(Model View Controller) architecture to separate the concerns (logics)

1. Model
2. View
3. Controller

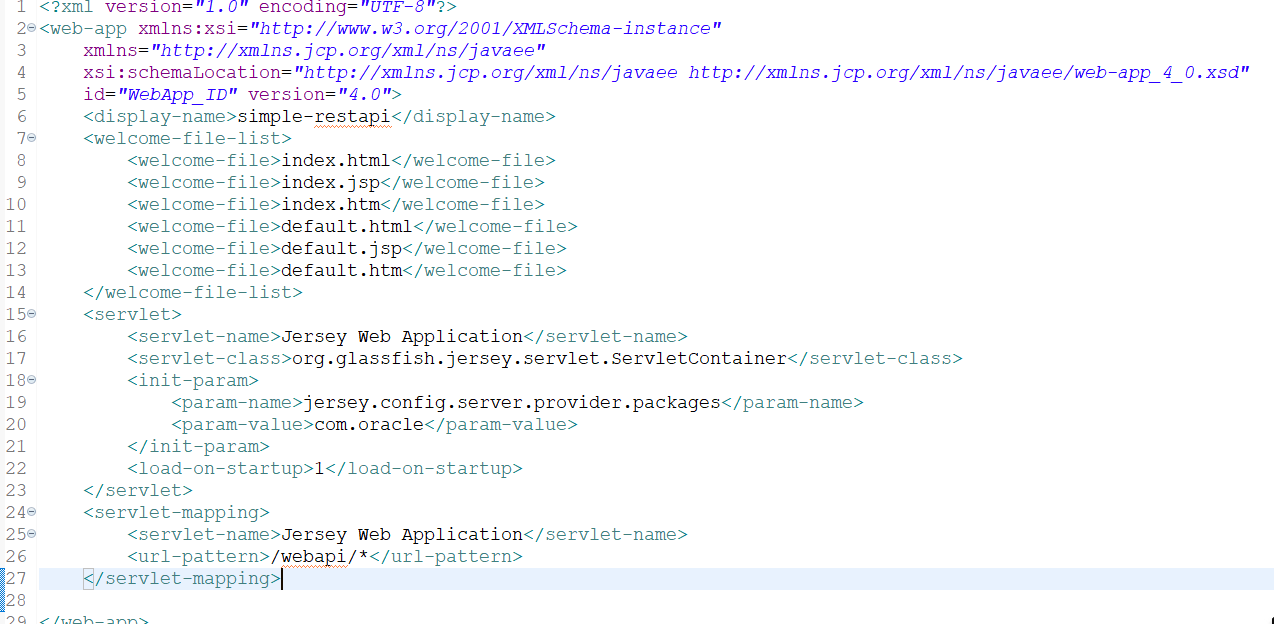


Front Controller: It is a layer in MVC which acts like an entry point for all the requests

How to create our first webservice using jersey

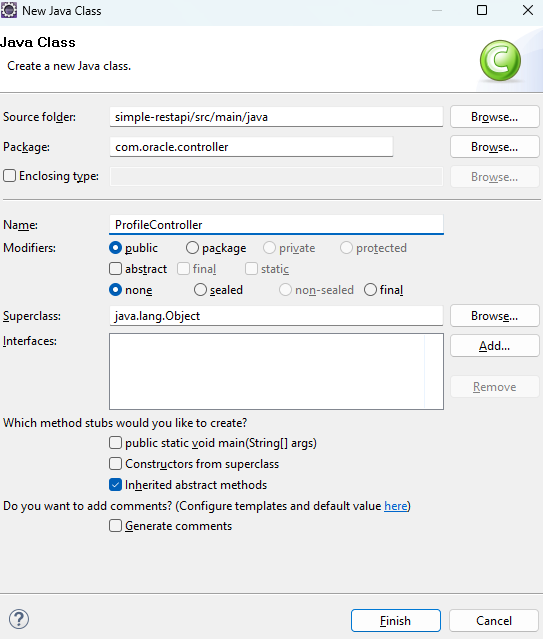
1. Configure the Front Controller in the web.xml file
2. Create classes in the controller layer that can handle the requests

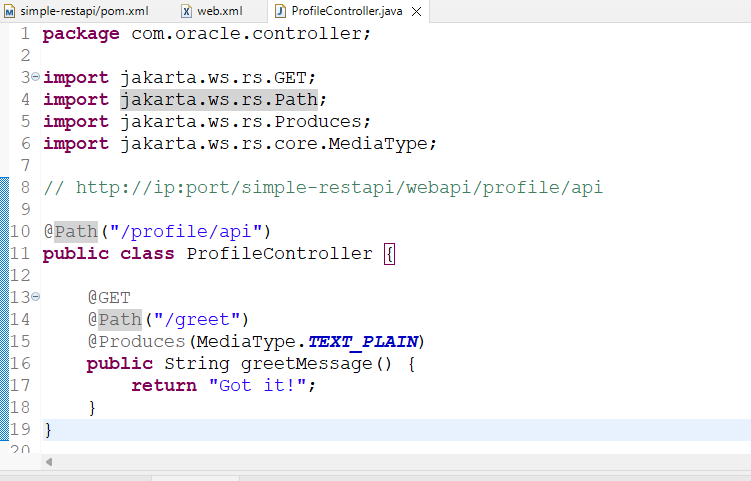
web.xml



Look at the <param-value> at line 20, which has com.oracle this must be the package where your controller class must be present

src/com/oracle/ProfileController.java





You must use client program to access this webservice, they must use HTTP client & the url of the webservice

request GET <http://localhost:8080/simple-restapi/webapi/profile/api/greet>

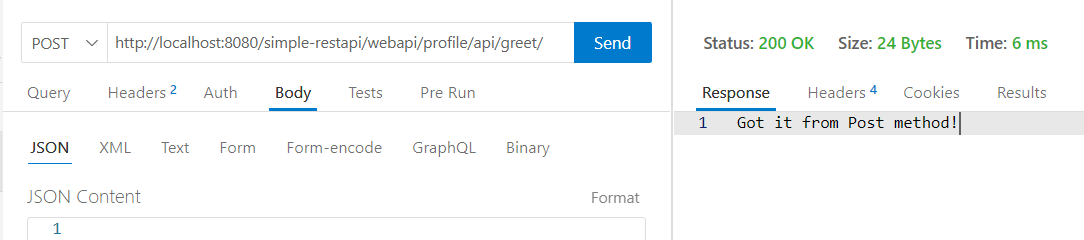
Since its practically impossible to test every webservice from every client program, we have to use a testing application that mimics the client program, but it will only create intermediate request & response, list of testing applications are

1. VS Code thunder client
2. Postman
3. SOAP UI

We can create multiple webservices with same URL but different HTTP methods



Output:



Response class

It helps you to customize the response so that you can have different status and different content in it

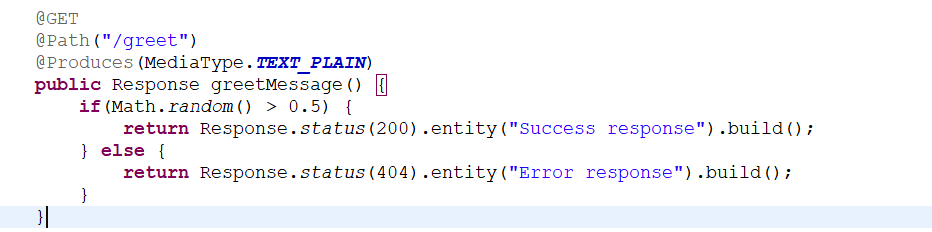
It provides methods like

status(int): Where you can pass status code

entities(content): Where you can pass the data like simple or complex types

How to create Response

Response.status(200).entities(“Hello World”).build();



Generating XML, JSON, HTML response entities (contents)



Output



Ways view can pass the data to the webservice

1. Using path parameters where you can have a dynamic path
2. Request body having JSON data
3. Query parameters ( optional parameter )

There are annotations to read the inputs from the client

1. Path parameter: @PathParam(“key”)

@Path(“/{id}/{name}”)  
m1(@PathParam(“id”) int x, @PathParam(“name”) String y) { }

URL is : /20/Raj then x = 20, y = Raj  
URL is: /30/Vijay then x = 30, y = Vijay

1. Request body: @Consumes(MediaType.JSON)

m1(Profile p) { … }

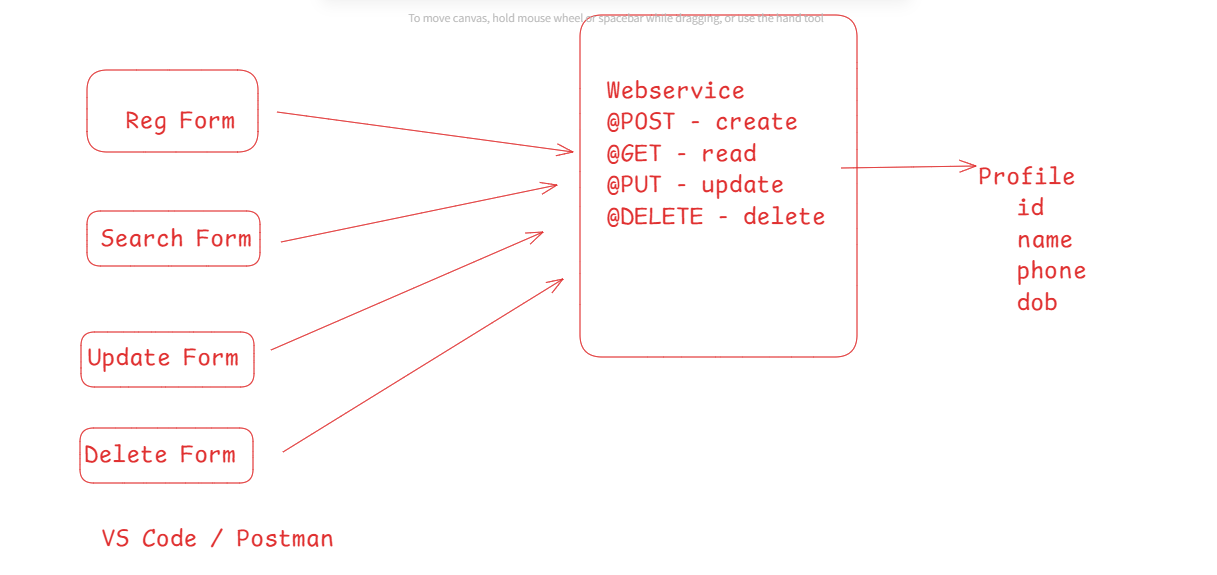
{ “name”:”Raj”, “password”:”1234”, “dob”:”2000-10-20”}

You must have a Profile class with name, password & dob properties

1. Query Parameter: @QueryParam(“key”)

/employee?id=20

m1(@QueryParam(“id”) int id) { … }

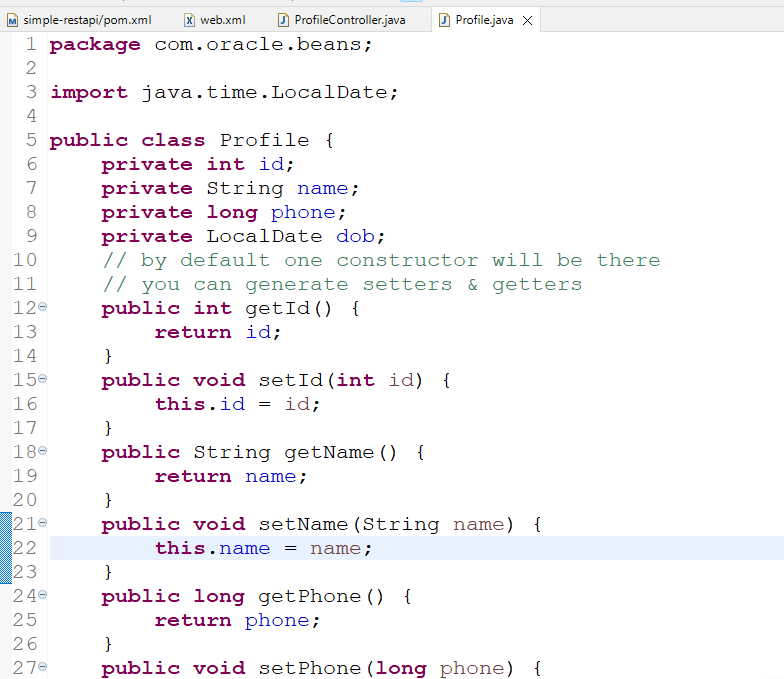


Note: We will be using Collection Framework to maintain multiple profile objects

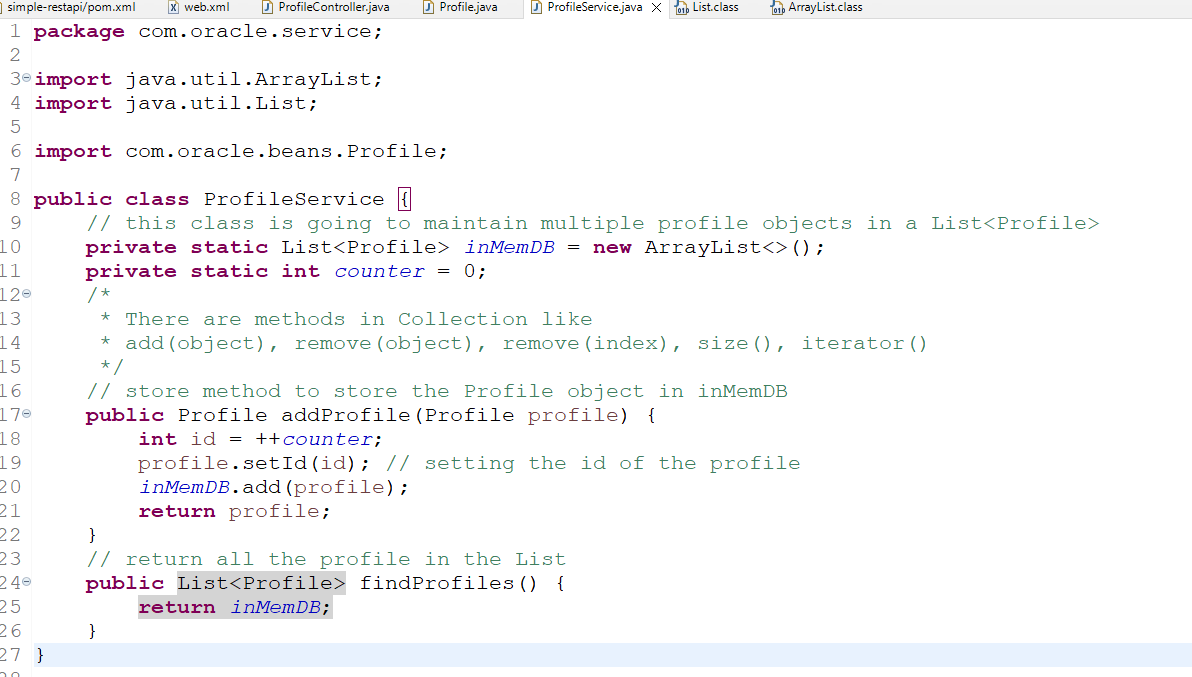
Things to create

1. Profile - id, name, phone, dob , setters, getters, constructors
2. ProfileService - crud operation performed on the Collection<Profile>

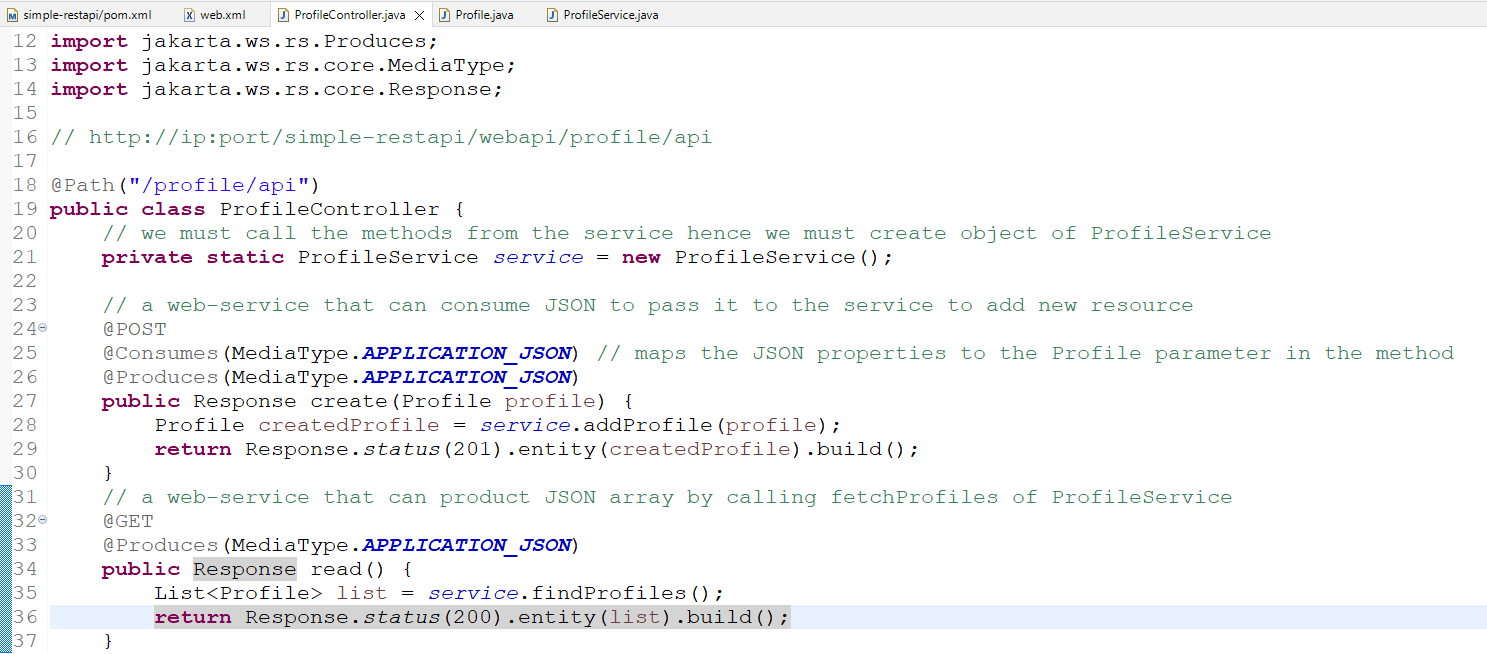
Profile.java



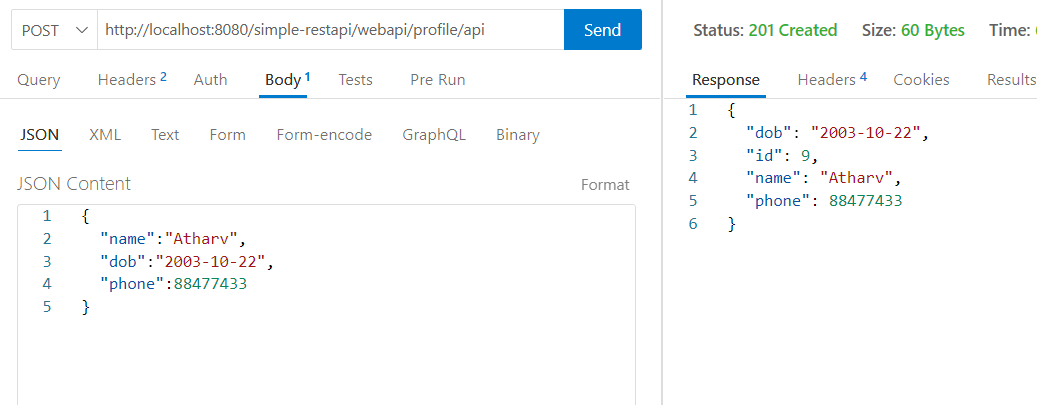
src/com/oracle/service/ProfileService.java



Modify the controller to call the methods of service

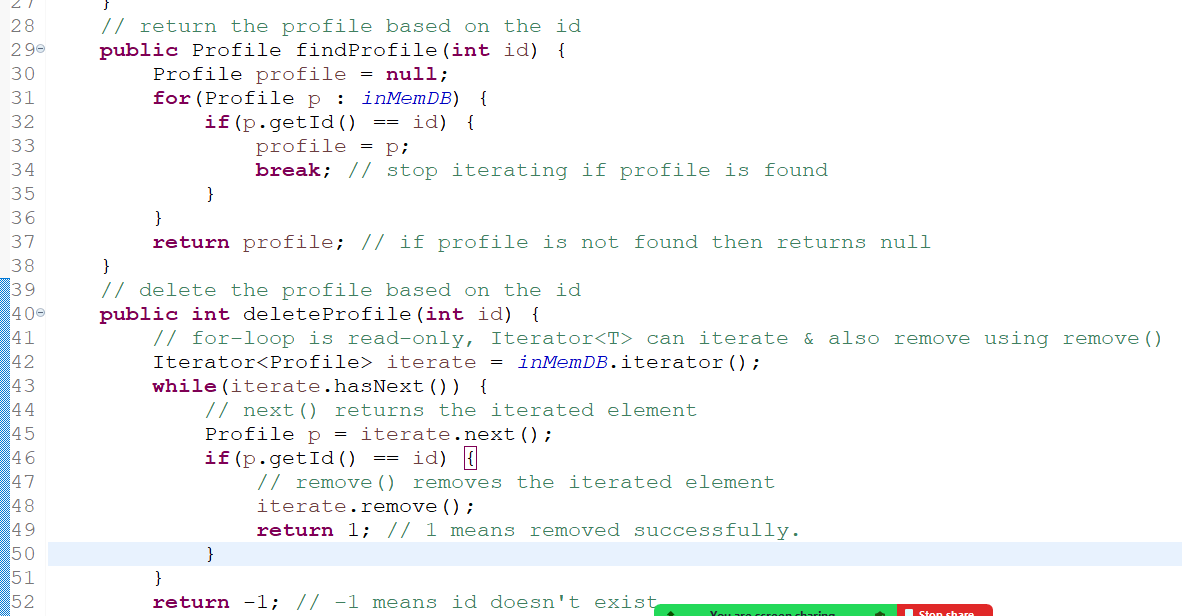


Output:



Passing parameter to the webservice using path parameter

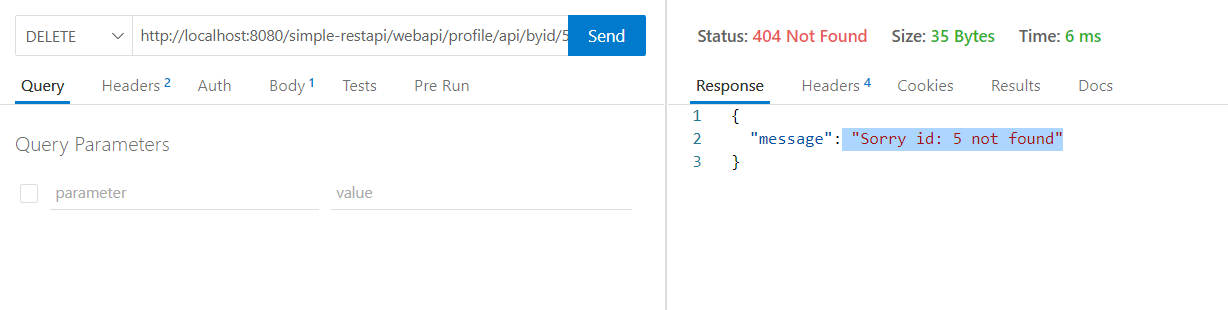
ProfileService.java



Create webservice to call findProfile by id & deleteProfile by id



Output:



Activity:

Use @PUT to update the phone number using id, send phone & id in the path parameter, implement a method in the service layer that can iterate the array list & set the phone number to the profile that matches to the id, test the webservice from the vs-code.

Hint: use, @Path(“updatePhone/{id}/{phone}”) & @PUT