Course TOC

1. Spring Microservices
2. React.js

Microservices: they are small independent service which can be developed, test & deployed independently

Service Discovery: It is a program that registers the microservices, in Spring there’s a Eureka Server that acts like service discovery

Discovery Client / Microservices: These are the programs which registers in the service discovery, in Spring there’s Eureka Client that takes care of all the common jobs of the microservice like:-

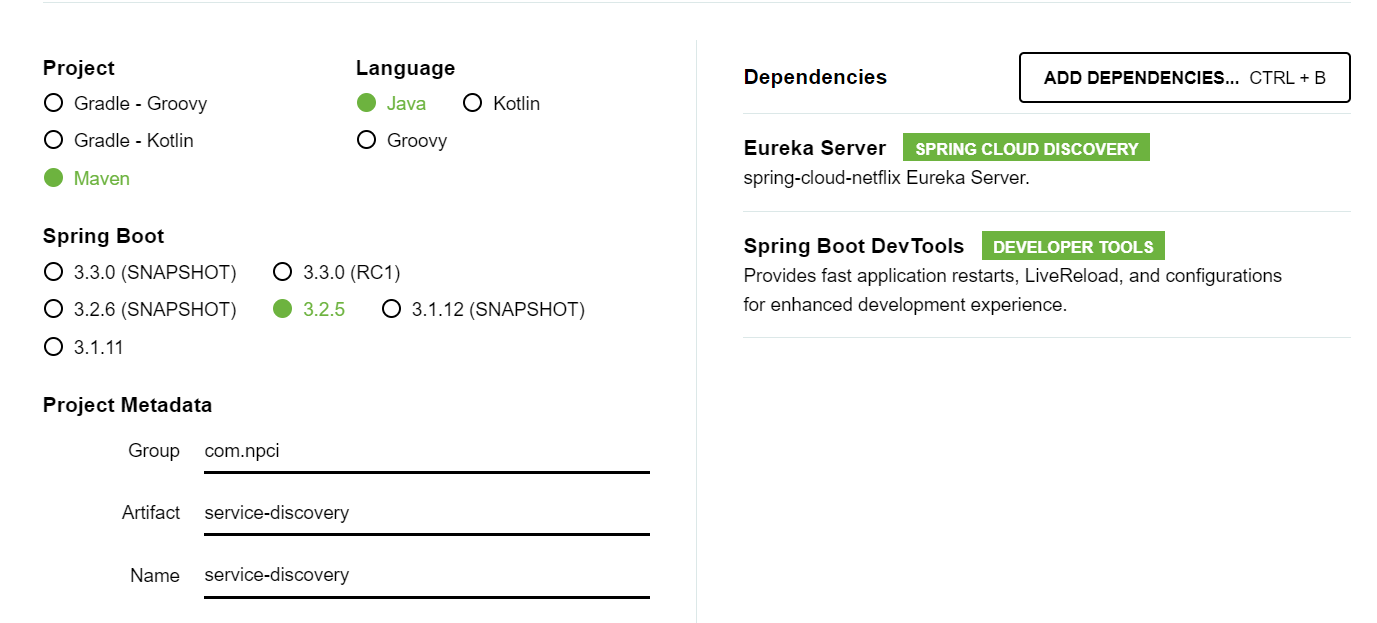
1. Automatically registering in the Eureka Server
2. Constantly pinging to the Eureka Server about its health status every 30s
3. Registers to the Eureka Server by default searching in 8761

Pre-requisites

1. Spring Boot 3.x
2. JDK 17 or later

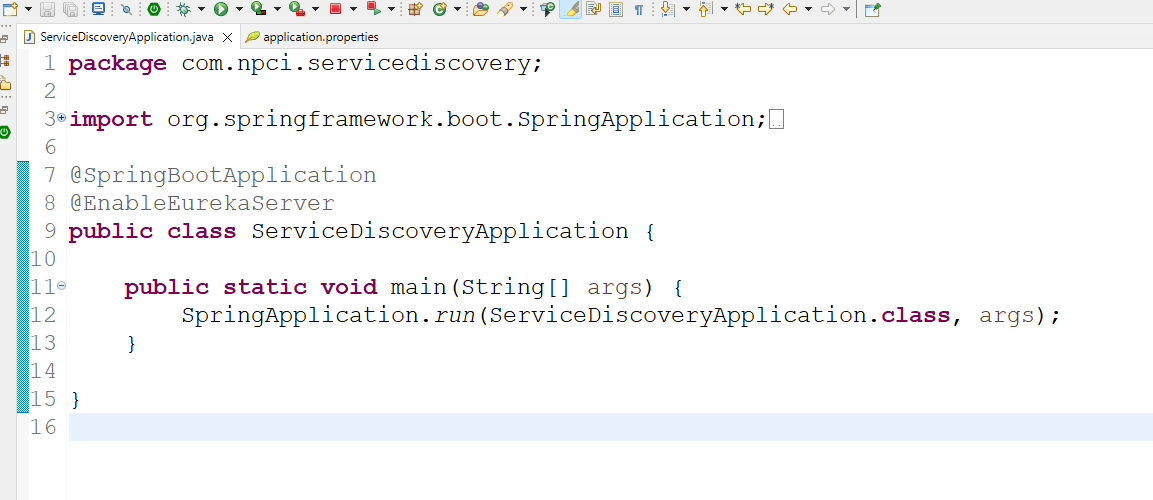
Creating service discovery

1. Eureka Server

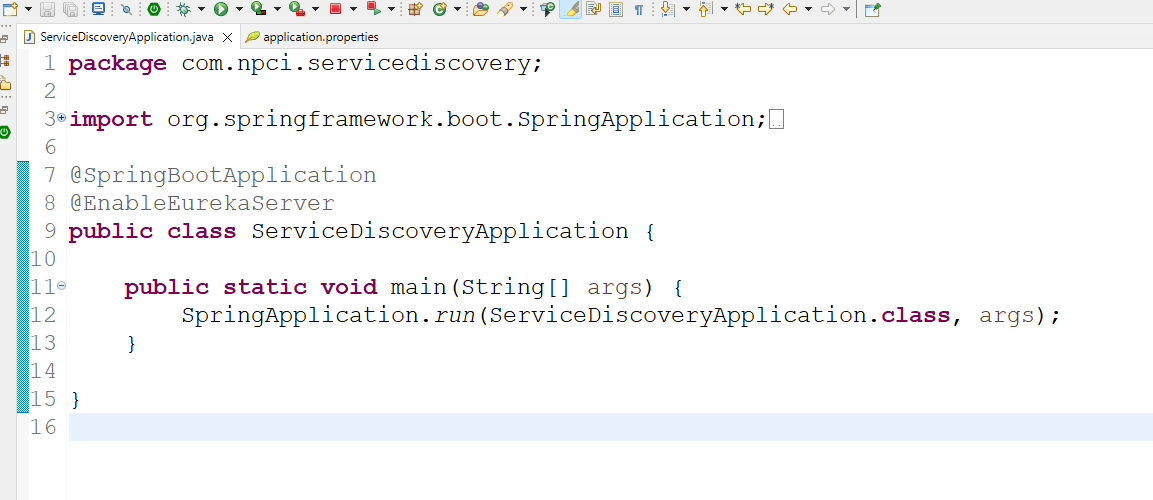


@EnableEurekaServer: This annotation enables the program to behave like service discovery & provides all the service discovery features, like registering & deregistering the microservices, recording the instance-id & physical address of the microservices, providing the UI dashboard to easily watch the registered microservices from any location

Note: Eureka Server downloads Eureka Client library also, we need to disable some client features that are automatically enabled by the spring boot like register with eureka & fetch from registry



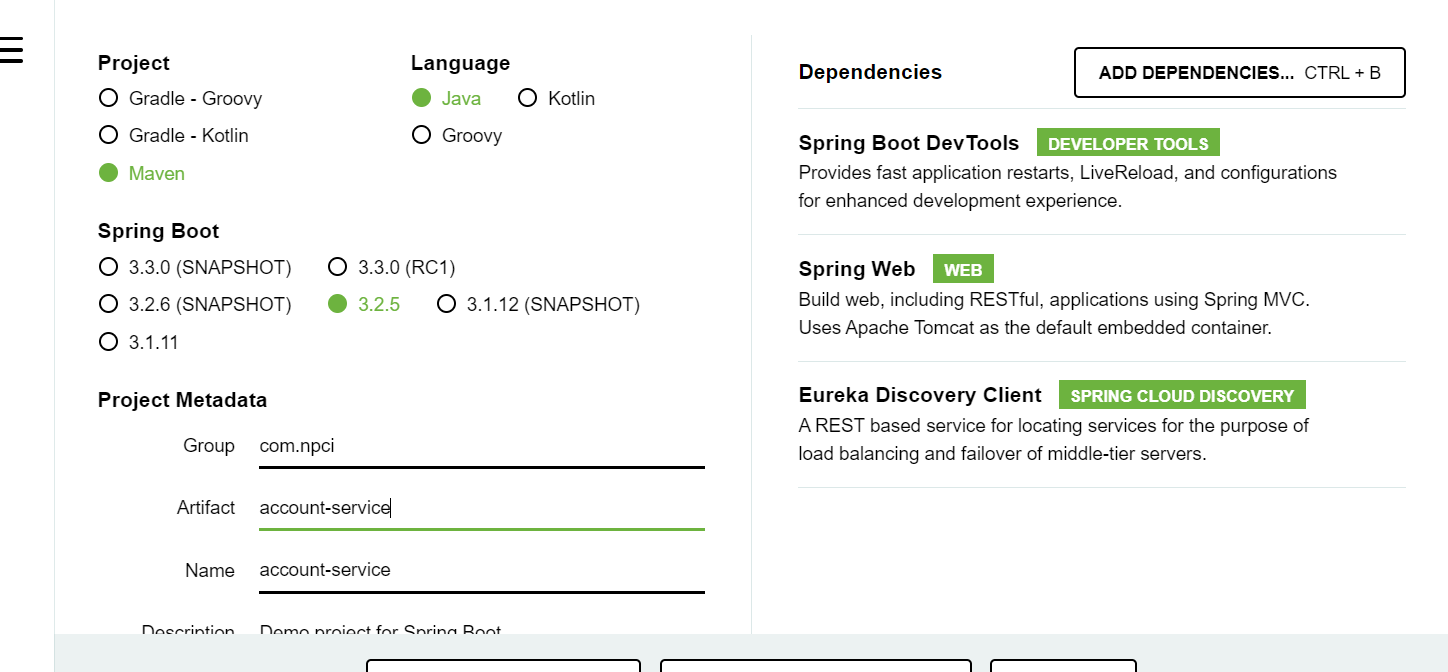
application.properties



Creating Microservices

Library:

1. Eureka Client
2. Web
3. Devtools

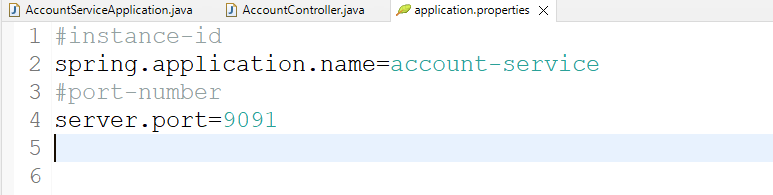


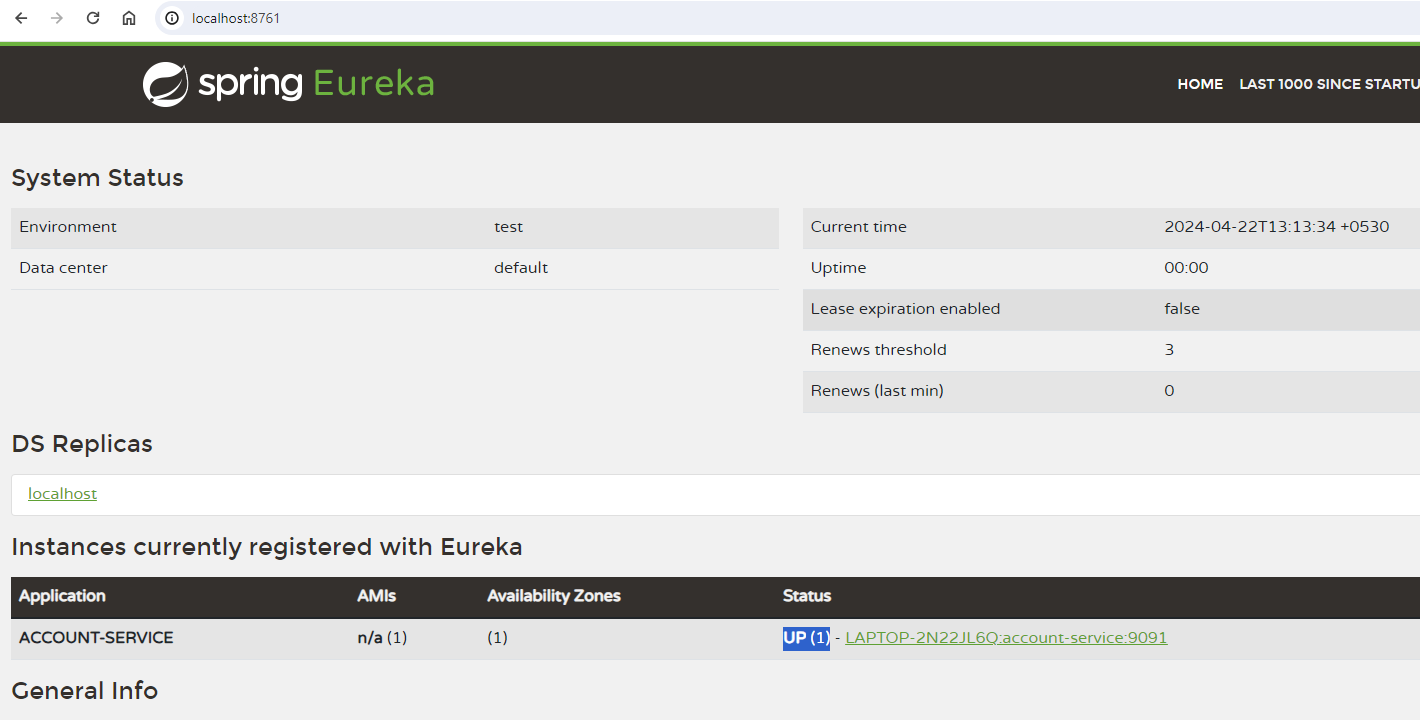
Note: This program automatically registers in the Eureka Server

AccountController.java



application.properties





Client Side Load Balancer:

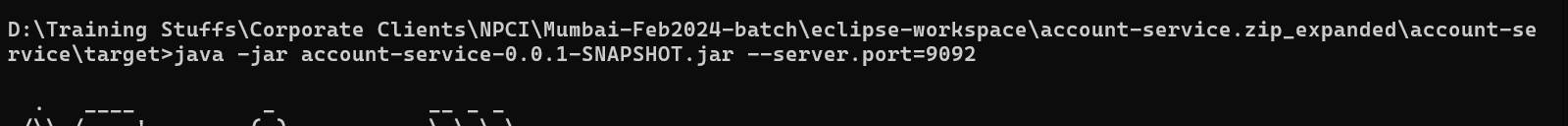
It is a program runs in the microservice to

1. Resolve the actual location of the microservice using the instance-id
2. Distribute the load across the multiple instances of the same instance-id if there are more than one instance.

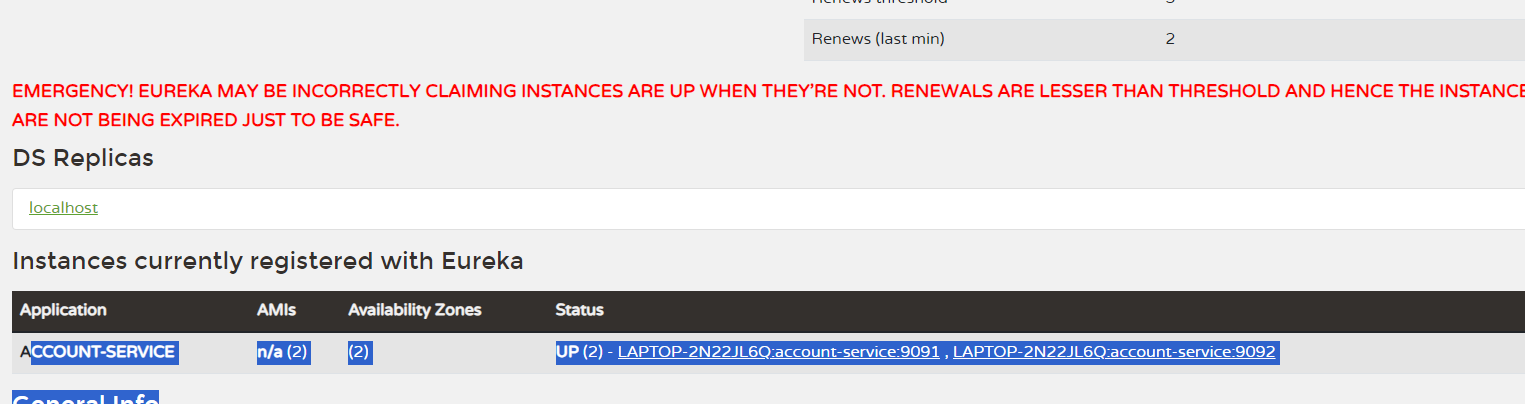
@LoadBalanced: This annotation creates the client side load balancer

How to create multiple instances:

we must have jar file to run the same program in different port number using terminal, but from eclipse you can run the same program only once



You can see 2 instances of account-service in the eureka dashboard



How to communicate from one microservice to another

a client microservice must use the instance-id of the remote microservice, to communicate we can use either

1. RestTemplate [or]
2. Feign Client: It is simplified approach than RestTemplate

RestTemplate: It is used in the client program to access the remote service, it provides inbuilt methods to make HTTP calls via GET, POST, PUT & DELETE, these methods would convert the JSON response to the java objects automatically based on the parameter

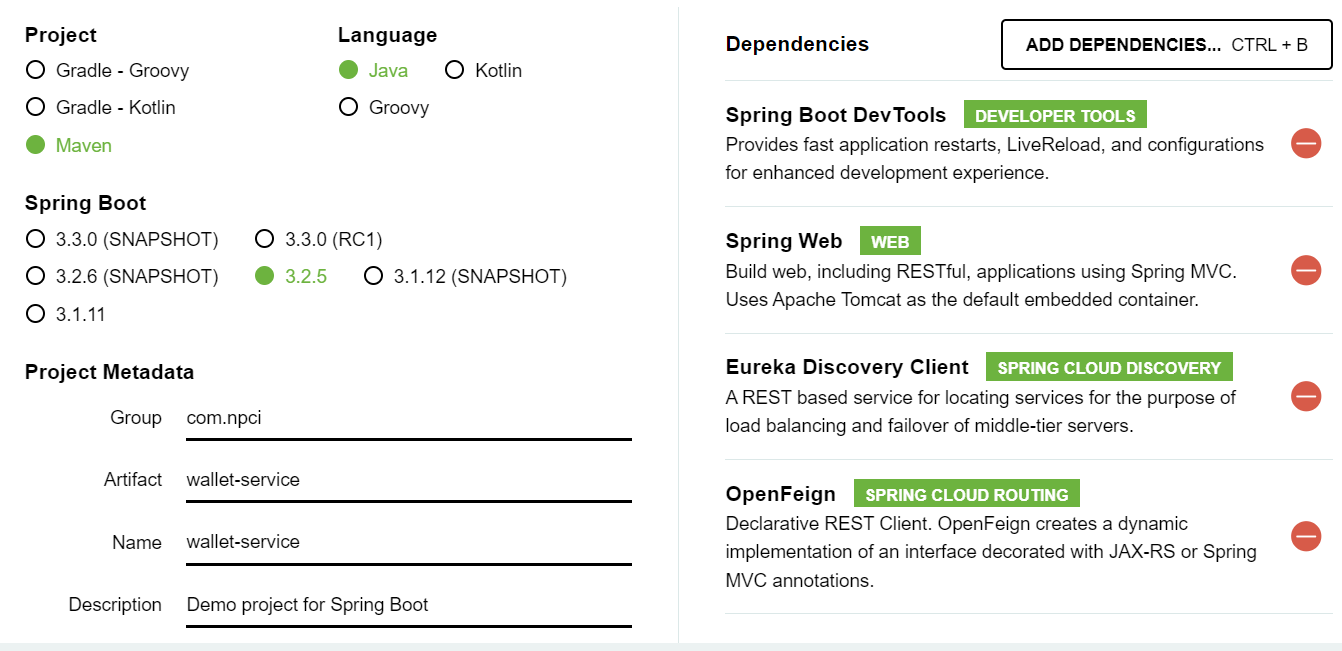
We must create RestTemplate with LoadBalanced attached to it as below:

@LoadBalanced  
@Bean  
public RestTemplate template() {   
 return new RestTemplate();  
}

@LoadBalanced enables rest template to send request that is distributed across the multiple instances

We need to autowire the RestTemplate wherever we want to send request to the remote service  
@Autowired  
RestTemplate rest;  
  
rest.getForObject(URL, String.class): This sends GET request to the remote service & converts the response in string format  
rest.getForObject(URL, Account.class): This sends GET request to the remote service & coverts the response in the form of Account object

Wallet Microservice: This will access Account Microservice

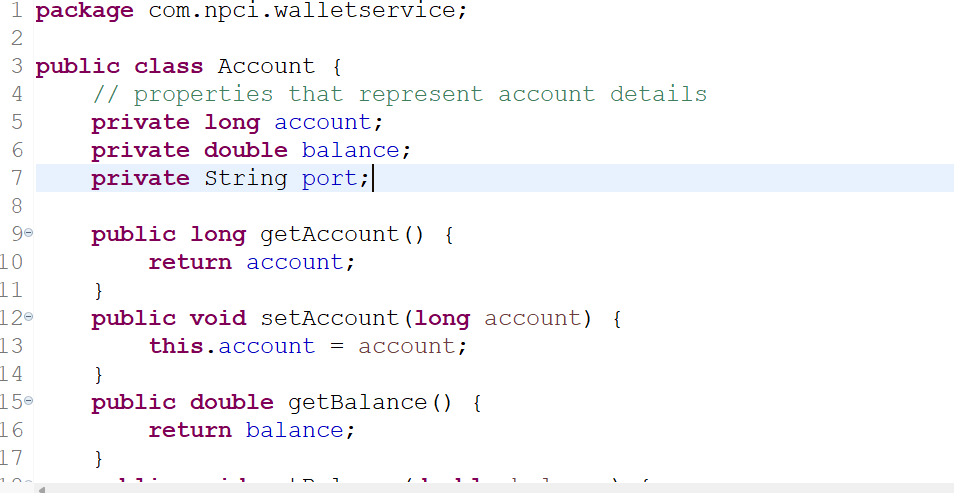


Note: OpenFeign is the library which can also make HTTP calls, but it has inbuilt load balancer, this is provided by spring cloud to simplify the HTTP calls, better than RestTemplate

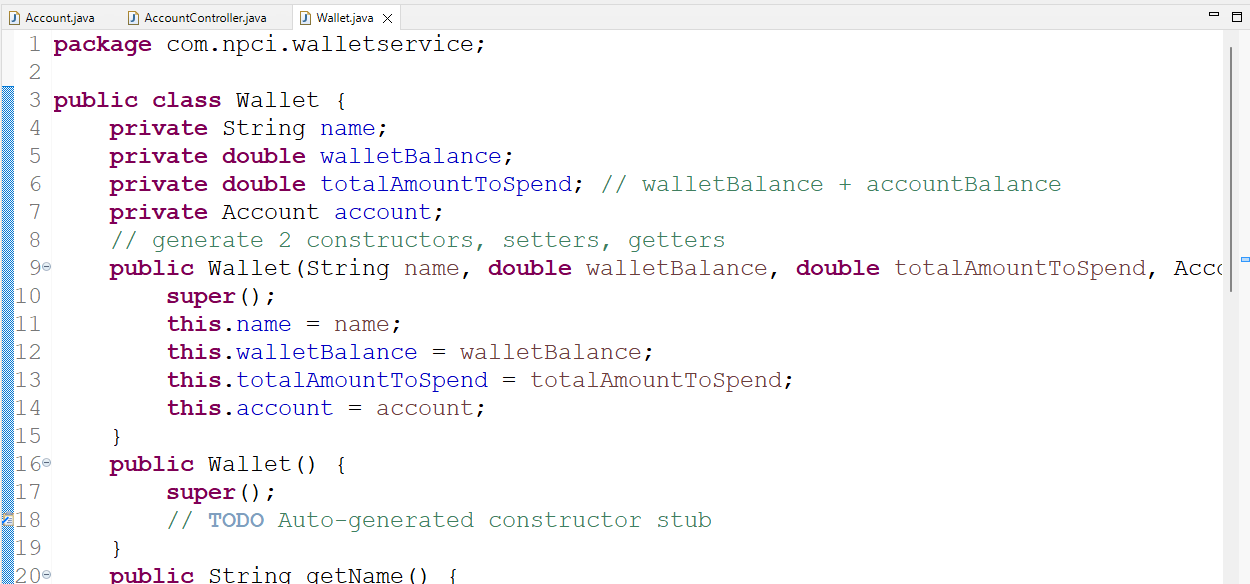
We need to create following classes

1. Account: To represent account details
2. Wallet: To represent account & wallet details
3. WalletService: To communicate with the remote service
4. WalletController: To handle the request from the user & call the WalletService

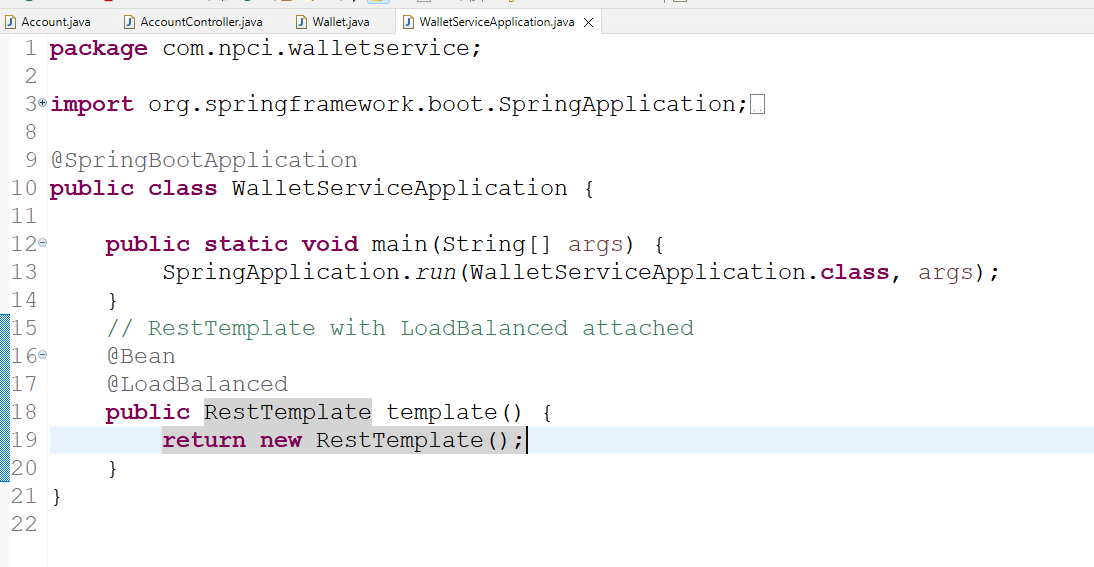
Account.java



Wallet.java

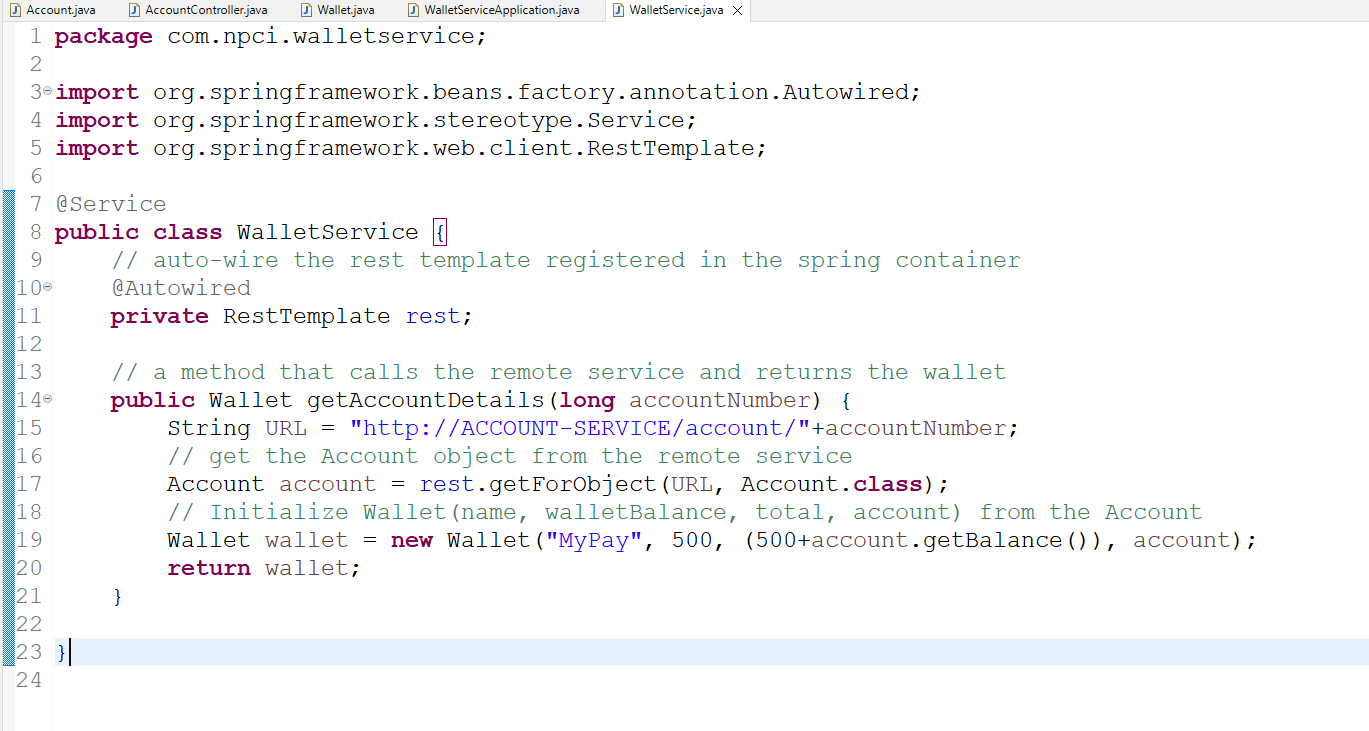


Create RestTemplate & register in the spring container



Autowire the RestTemplate in the Service layer and return the wallet details by extracting the Account details

WalletService.java

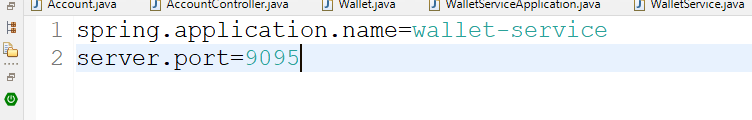


Controller to handle the user request and call the WalletService

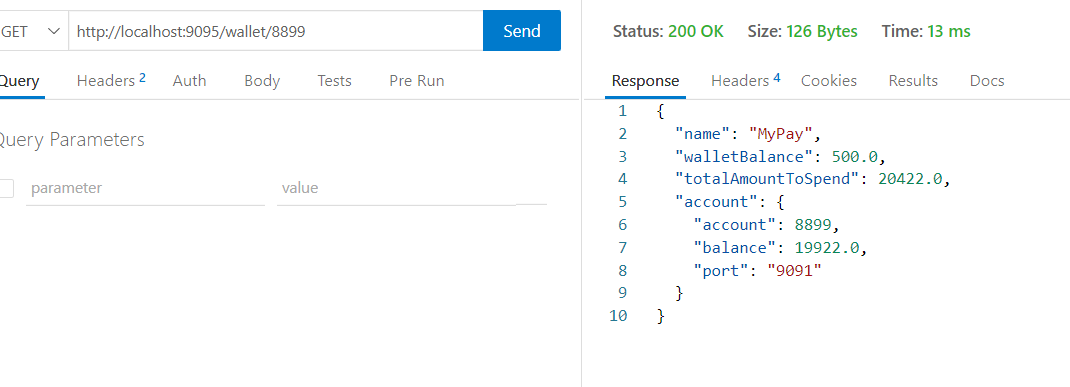
WalletController.java



application.properties



Output:



Drawbacks of RestTemplate

1. It is an older API
2. It can’t be reused like URL need to be repeated

Feign Client

It is used to make remote calls, it is released when spring microservices was released

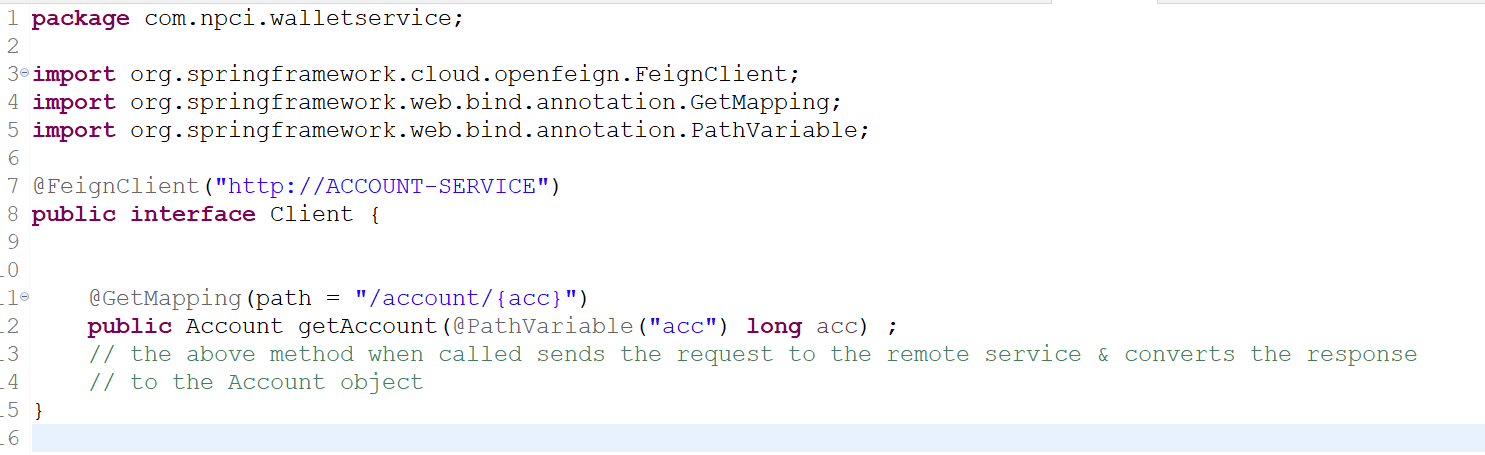
* It helps you to reuse the remote calls
* It internally uses the client side load balancer - you don’t need to use @LoadBalanced

You need to create a reusable interface which helps to create a reusable remote calls

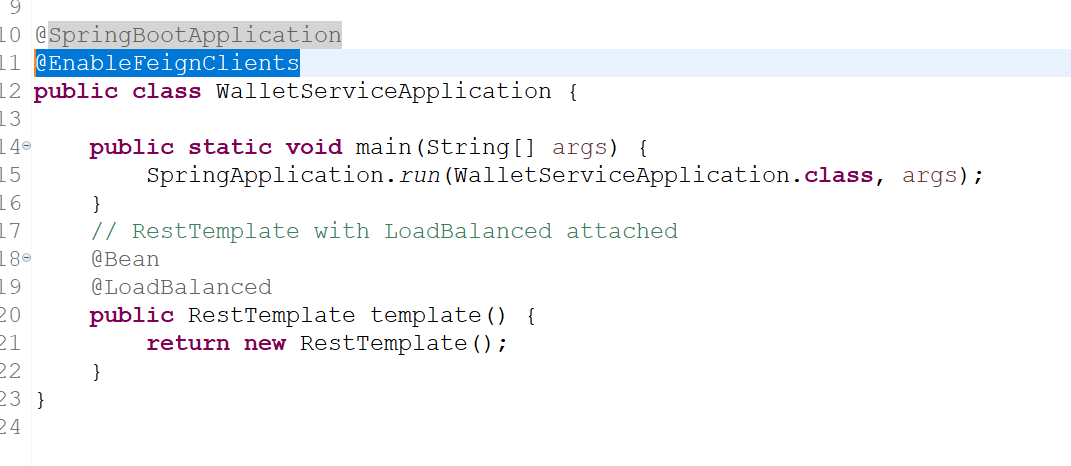
@FeignClient(“http://ACCOUNT-SERVICE”)  
interface Client {   
 @GetMapping(“/account/{accountNumber}”)  
 public Account getAccount(@PathVariable(“accountNumber”) long accountNumber);  
}

Calling getAccount sends a GET request to the remote microservice, the return type tells the response must be converted to which object ex: JSON to Account object  
Note: You don’t have to implement this interface, spring boot automatically implements the interface we only need to use @EnableFeignClients to let spring boot implement the interface  
@EnableFeignClients: This scans all the @FeignClient interface & lets spring boot to implement that interface & register the implemented object in the spring container

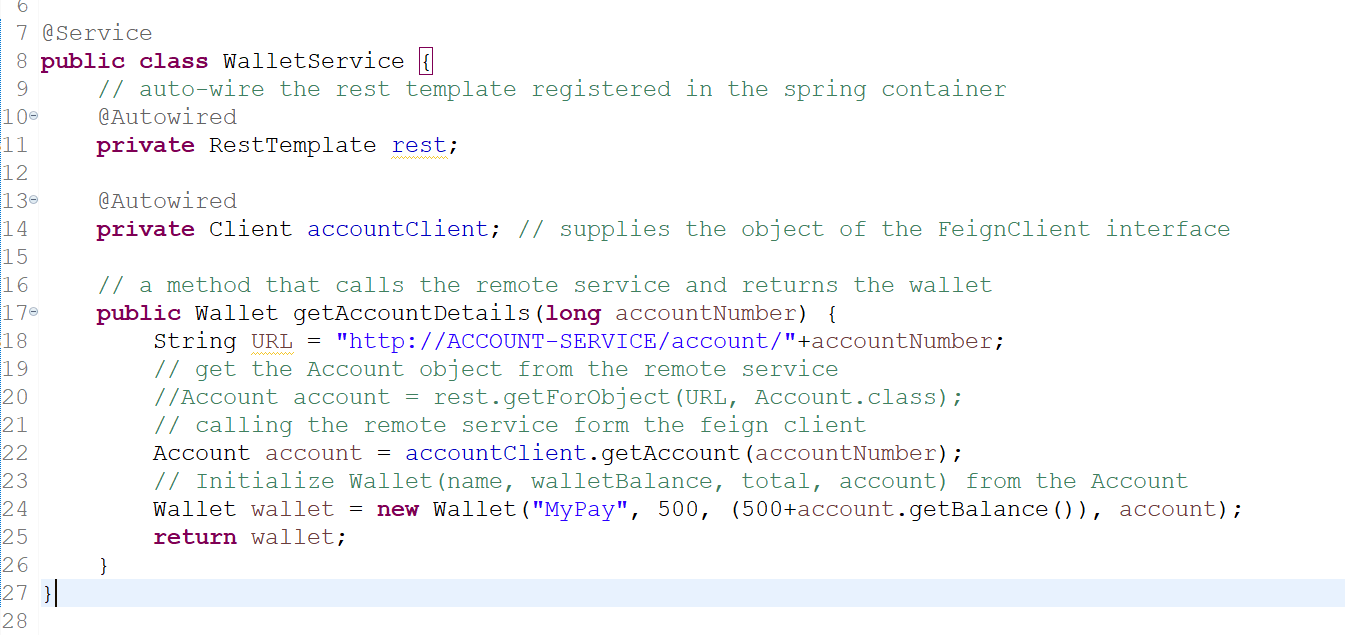
Client.java



We need to mention @EnableFeignClients in the main class so that this interface will be implemented using LoadBalancer behind the scene.



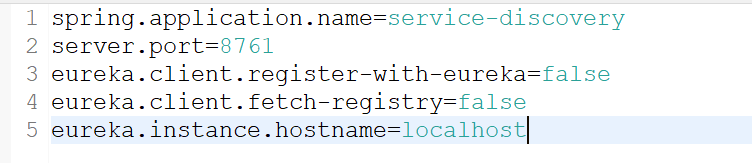
Change the service layer to use the FeignClient interface instead of RestTemplate



How to deploy the microservice in different machine

1. We need to create an EC2 machine
2. Service discovery application.properties must have some changes

application.properties

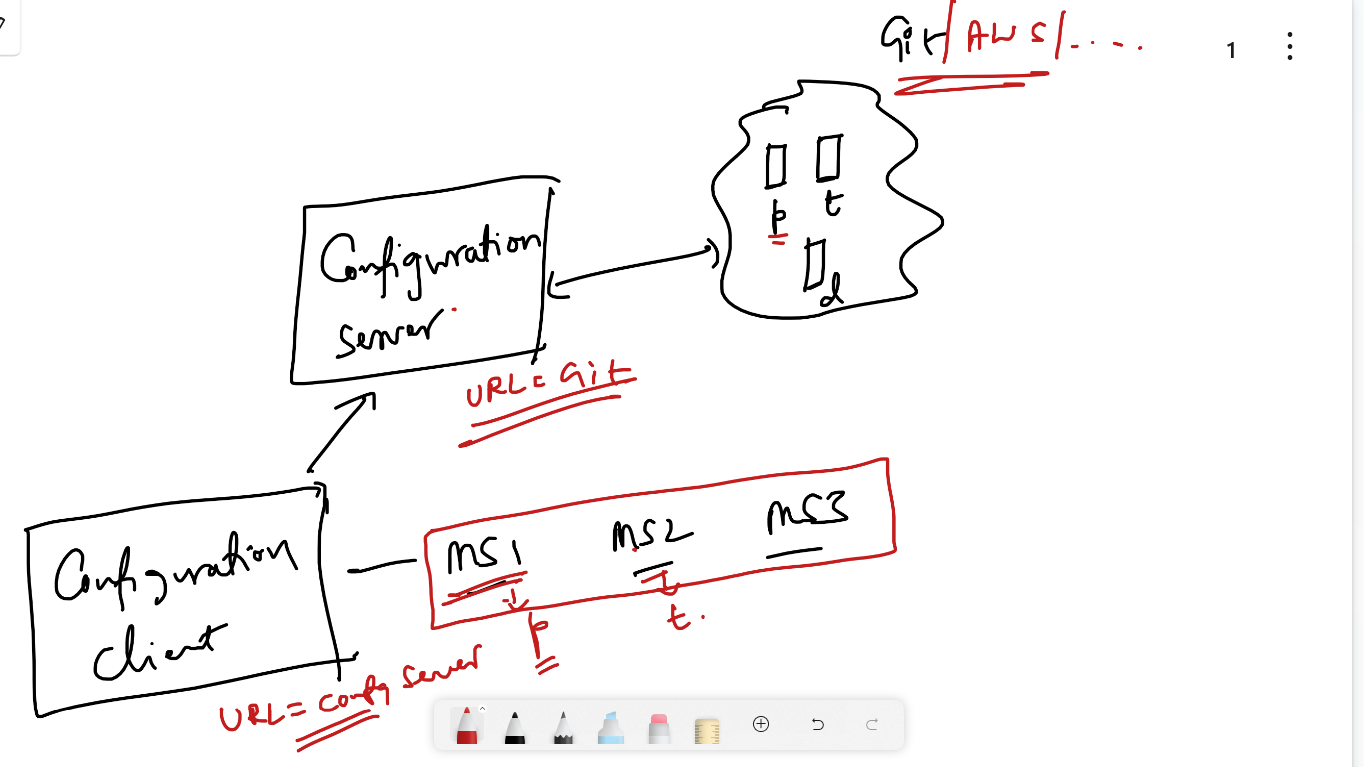


Distributed Cloud Configuration

It is mainly used when you want to share common configurations to the multiple microservices without changing the configurations locally and also when same microservice wants to load different configuration file for different environment (development, production, testing)

Spring helps you to use this type of configurations using 2 programs

1. Configuration Server
2. Configuration Client



Configuration Server: It is a program that connects to the centralized repository like GIT, SVN, AWS and etc.

@EnableConfigServer: This annotation creates a config server which connects to the centralized repository based on the URL provided in the property file & fetches the configurations for the client

Library you must use for server

Config Server

Configuration Client: It is a program that connects to the configuration server so that the config server will pull the configurations for the client, in the property file you will be mentioning the properties that connects to the Config Server & also which configuration file it needs

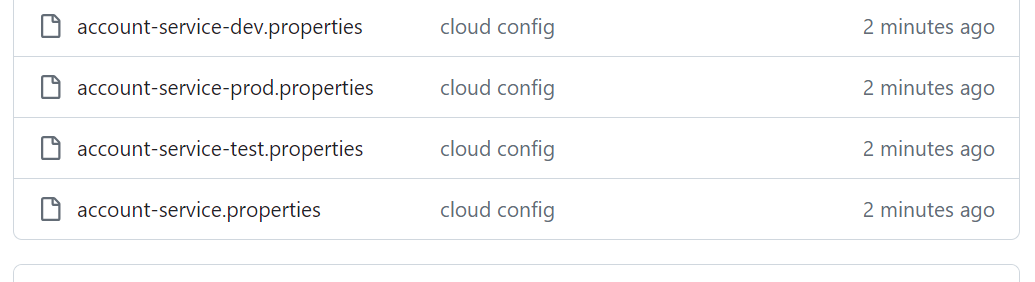
Library you must use for client is

Config Client

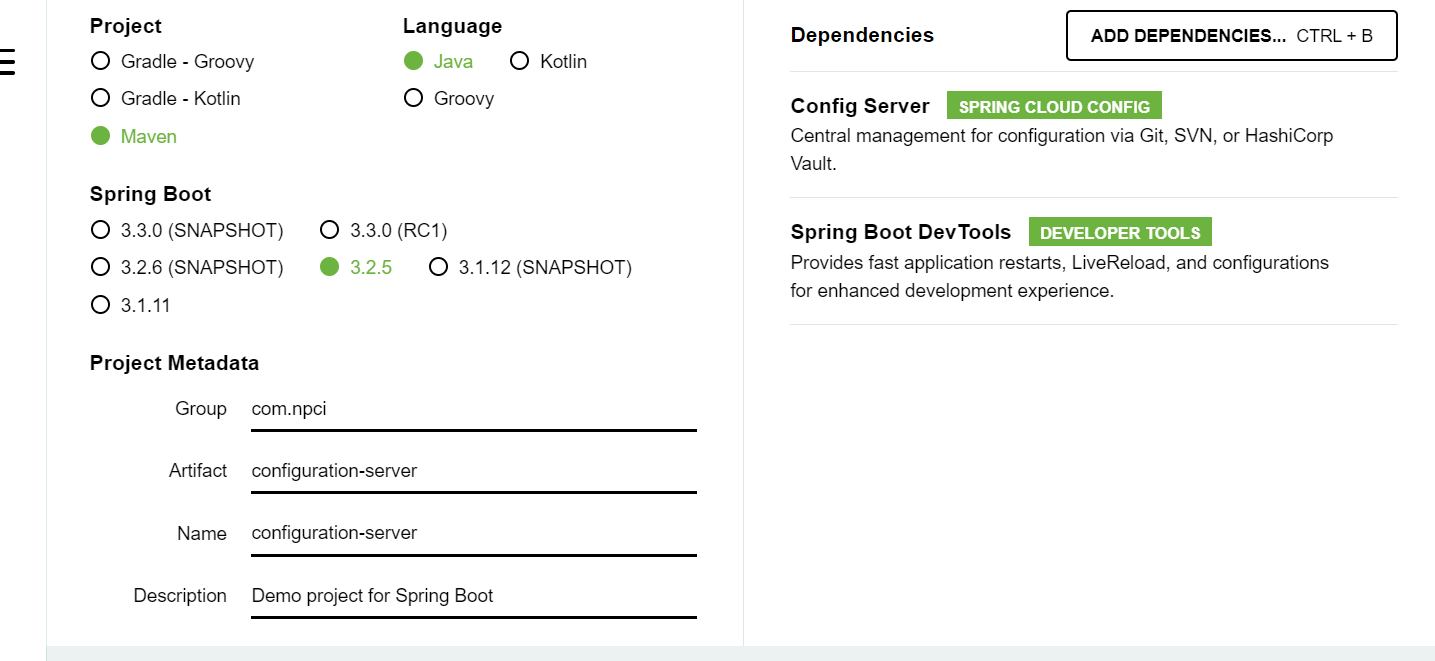
Steps to use the distributed cloud configuration

1. Push some configuration files to the GIT, separate the file names with -dev, -test, -prod i.e., hello-dev.properties, hello-test.properties, hello-prod.properties, hello.properties
2. Create configuration server to connect to the GIT
3. Create configuration client to connect to the Configuration Server & mention in which profiles(environment) you are running the program

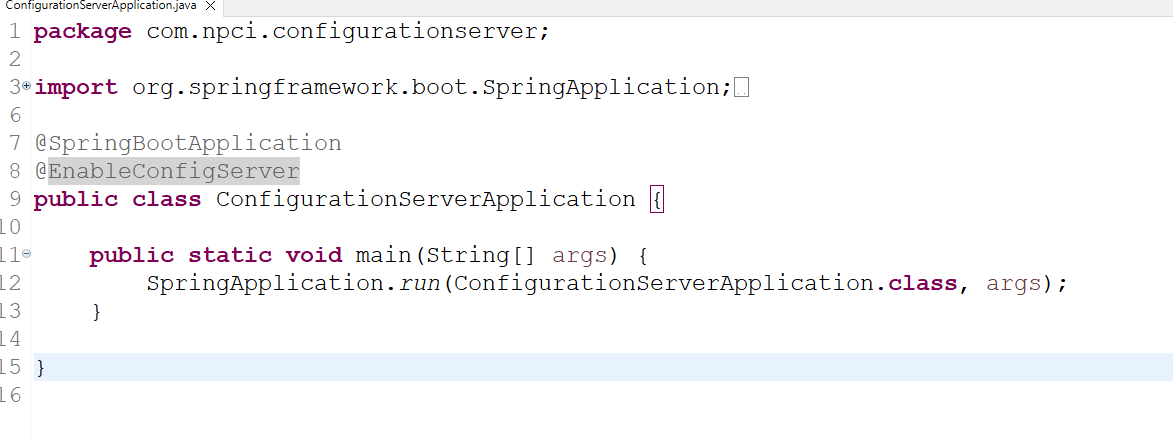
Creating configuration files in the GIT



Creating the configuration server program



Add @EnableConfigServer to the main class

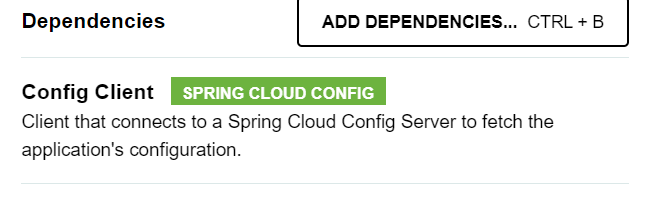


application.properties

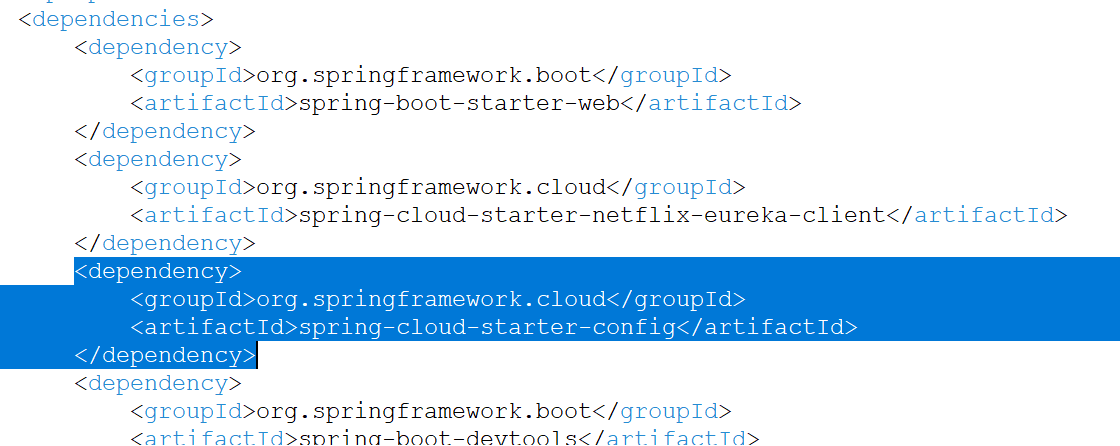


Now the client program should get the configurations from the configuration server by mentioning the configuration server URL & also which configuration file it needs

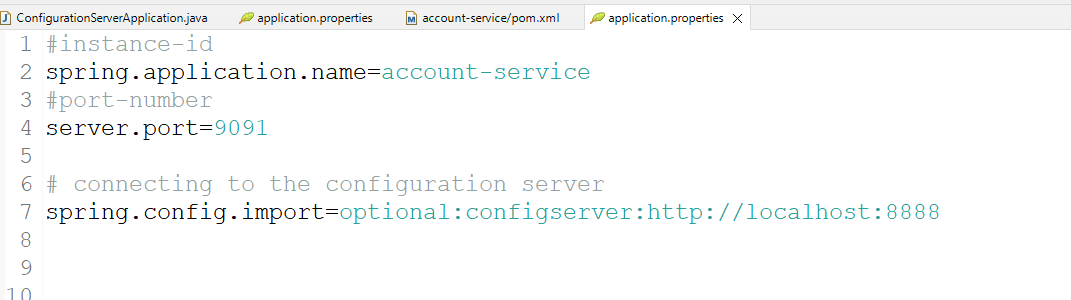
Firstly you need to add config client library



Update the account-service/pom.xml



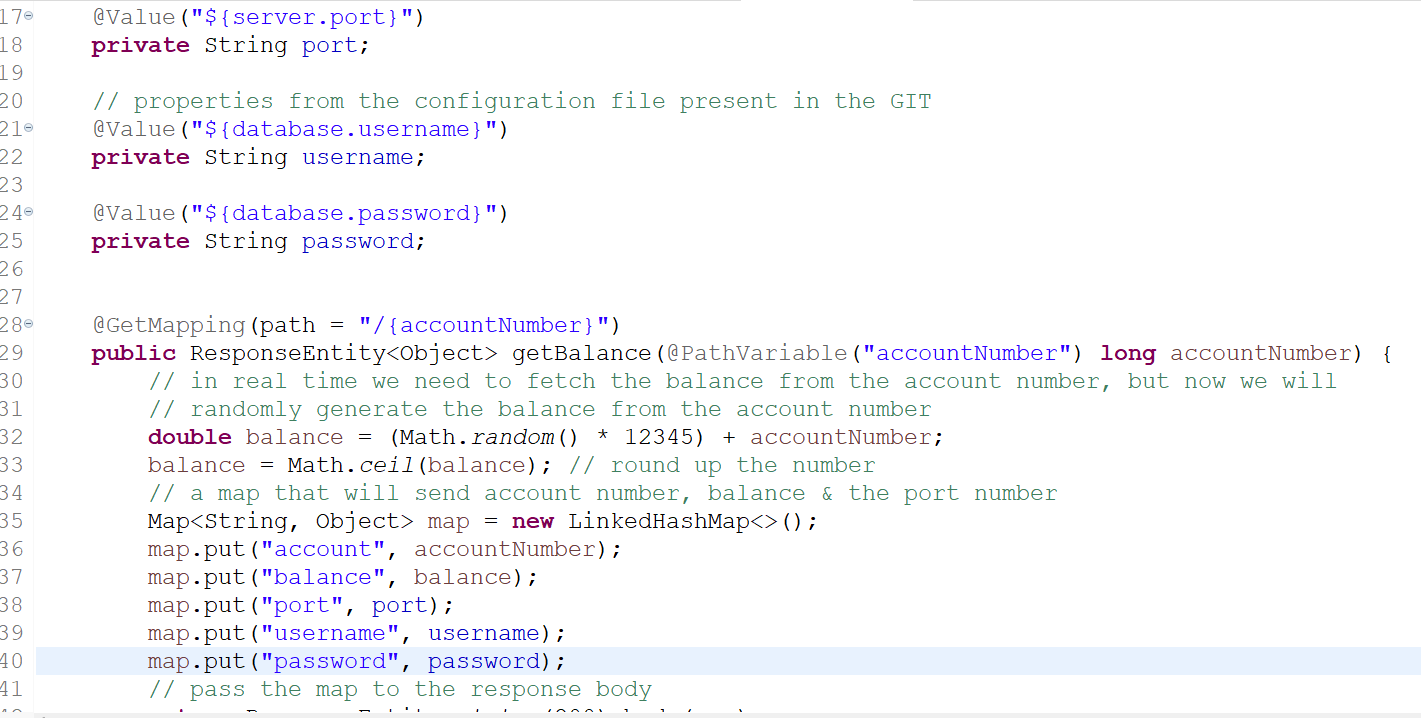
account-service/application.properties



By default it gets account-service.properites from the config server, however you can use a property called

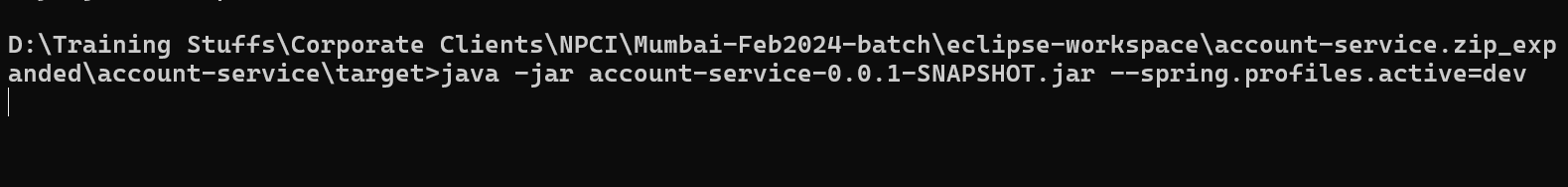
spring.profiles.active=dev # this gets account-service-dev.properties  
spring.profiles.active=prod # this gets account-service-prod.properties

All the property files have common properties like database.username & database.password hence you can have a webservice to read these data so that you will know which property file config client got



Now run these programs in this order

1. Service discovery
2. Config Server
3. Account Microservice : build the jar and run this program by providing spring.profiles.active



Day 2 Agenda

1. Encrypting & Decrypting sensitive data
2. Circuit breaker pattern
3. Reactive Programming using Web Flux
4. Spring Security

Encrypting & Decrypting sensitive data

Sensitive information’s must be encrypted so that end users must not able to understand, there will be a program which will decrypt the data

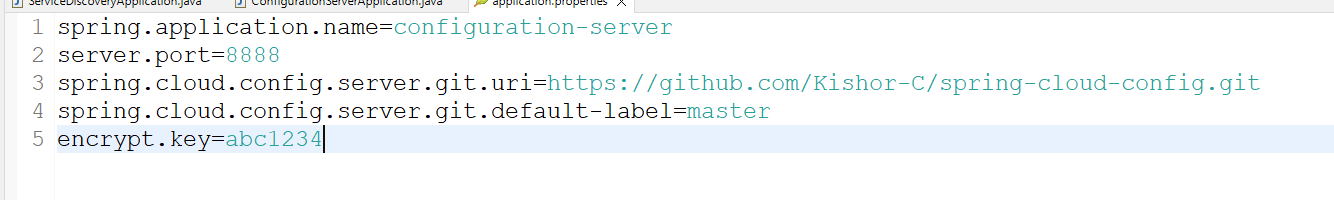
Configuration Server provides two url’s to encrypt & decrypt the url is

1. config-server-ip:port/encrypt
2. config-server-ip:port/decrypt

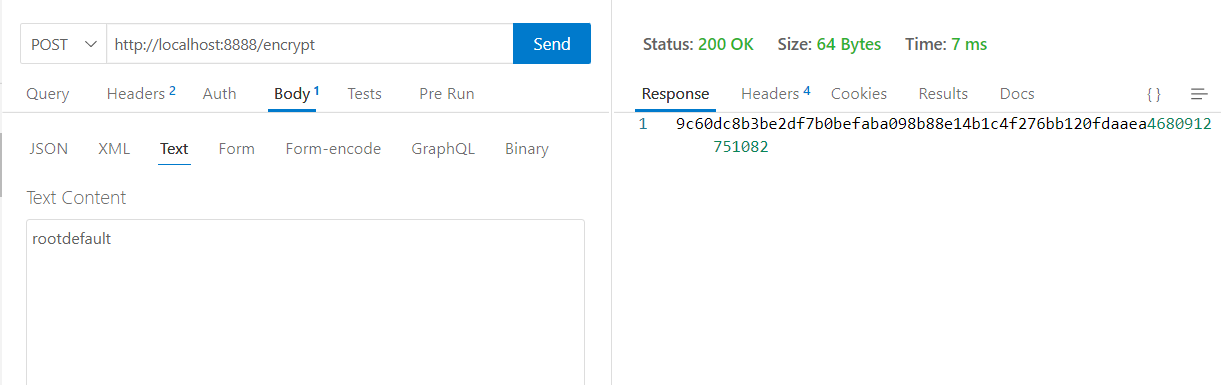
Configuration server needs a key to encrypt or decrypt

encrypt.key = abc1234

configuration-server/application.properties

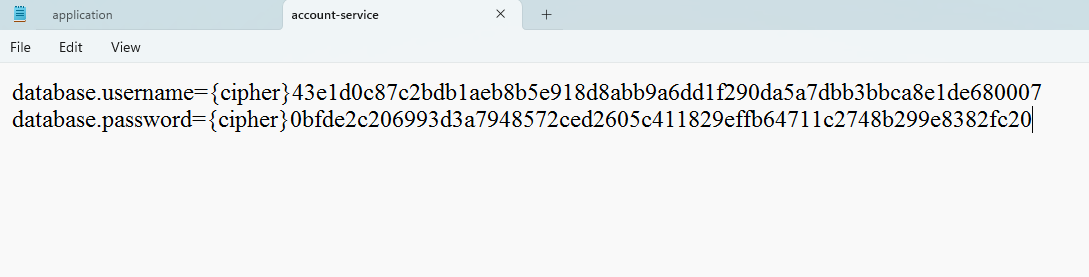


Now you can use /encrypt & /decrypt url to encrypt the data.

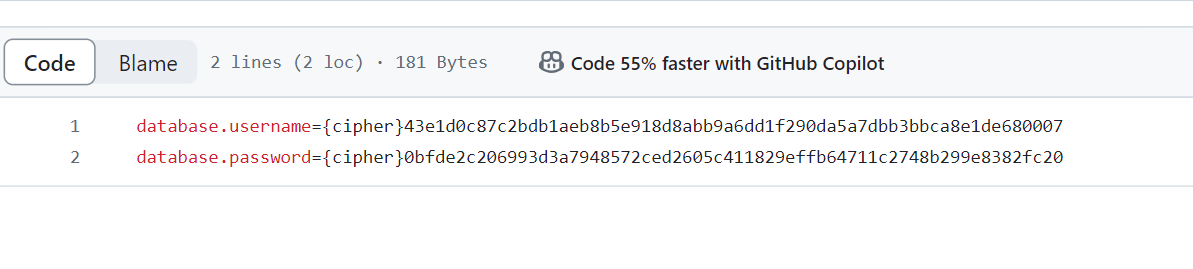


In the properties you need to use {cipher} beside the encrypted data so that the configuration server will decrypt the data and give them to the microservices

account-service.properties



Update all the properties in GIT



Circuit Breaker

It is a design pattern that is used to stop the cascade of failures across the microservices when any remote service is slow/down

It uses three states while communicating with the remote service

1. OPEN: It doesn’t send the request to the remote service
2. HALF\_OPEN: It is to decide whether the circuit should go to open or close state based on the failure rates
3. CLOSE: It sends the request to the remote service

Note: Automatically circuit breaker in open state goes to half open state after 60 seconds

Library we need to use for circuit breaker are:

1. Resilience4J: Gives the circuit breaker configurations
2. Actuator: gives the end points to see the circuit breaker status, failure rates, threshold
3. AOP: Takes care of providing alternate response when the circuit is open without invoking the code i.e., invoking a fallback method

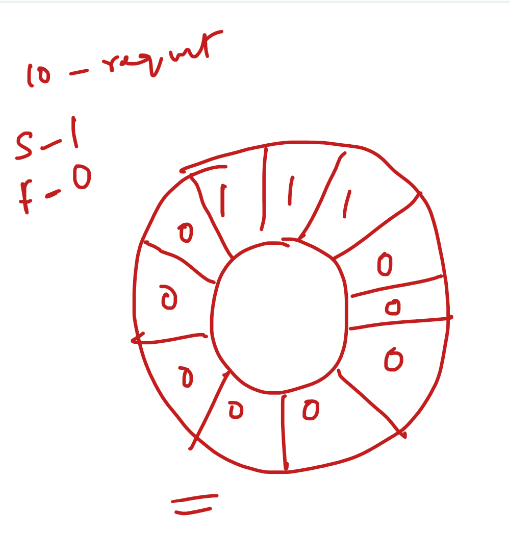
@CircuitBreaker: This annotation is used on the method which makes a remote call, this annotation will have a fallback method configuration so that fallback is called if remote service is unavailable or if circuit breaker state is open

@CircuitBreaker(name = “getAccount”, fallback = “getAccount2”)  
public Wallet getAccountDetails(long acc) { … } // actual method that calls remote service  
  
// fallback method must have the same signature with a Throwable parameter  
public Wallet getAccount2(long acc, Throwable t) { } // fallback method gets called automatically if remote service is down or circuit breaker state is open

In application.properties you will configure the failure rates, threshold, for the getAccount (name of the CircuitBreaker)

ex: resilence4j.circuitbreaker.instance.getAccount.threshold=50 #50% failure is the threshold

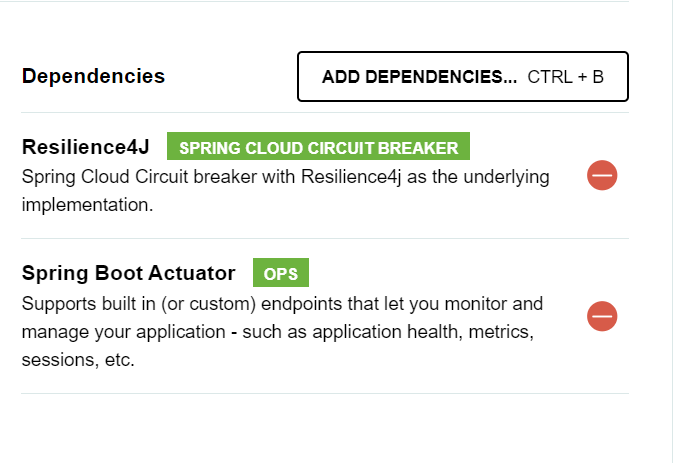
Reslience4j uses RingBit buffer to track the failures



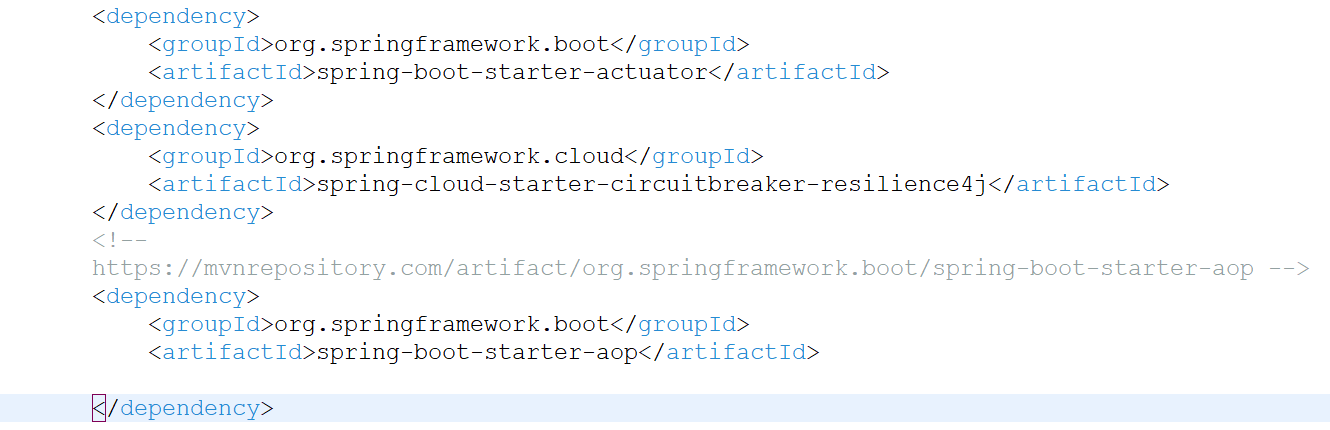
Ring bit buffer is a memory to track the success & failures, success will be entered with 0 & failures will be entered 1, based on these numbers it identifies the failure %.

Note: AOP may not be available in spring initializr, you need to get it from the Maven

Note: We need to configure the circuit breaker in the wallet service

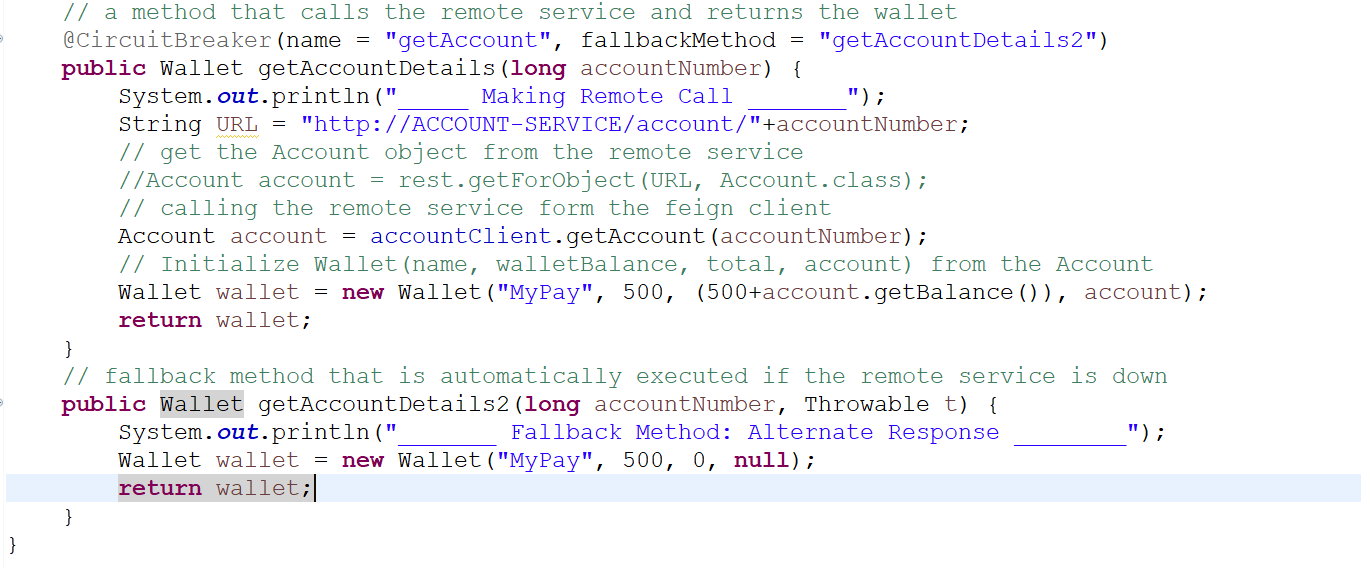


wallet-service/pom.xml

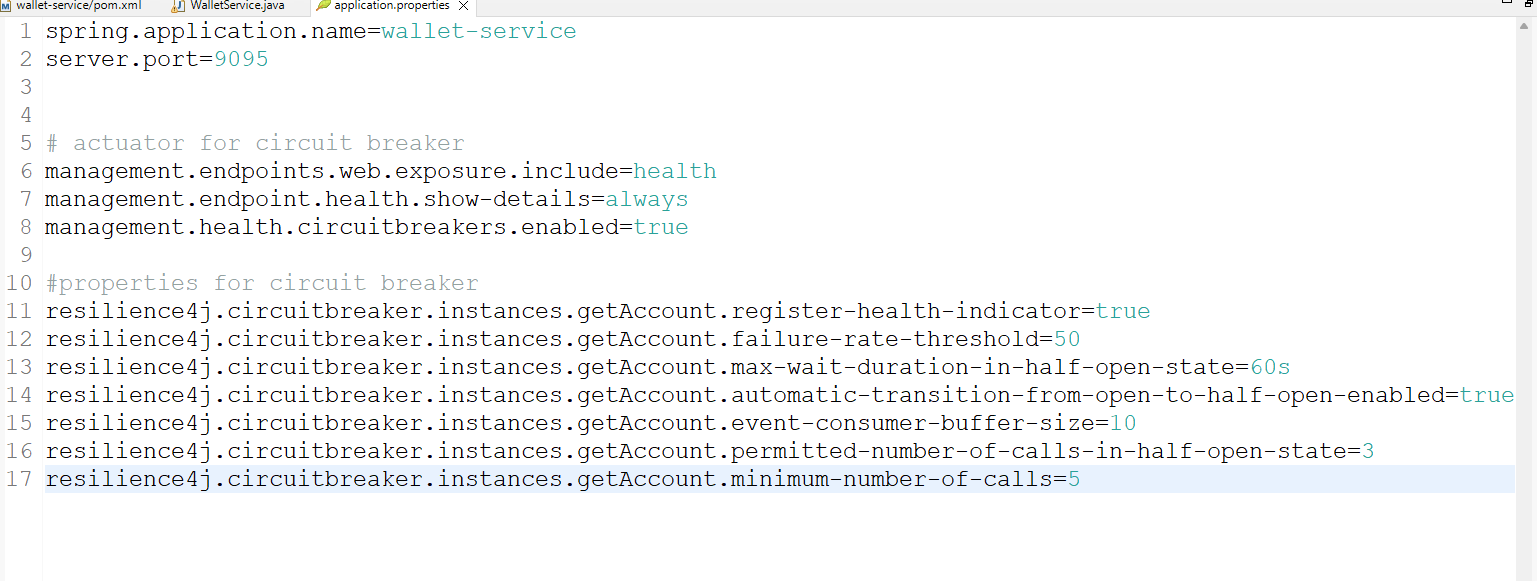


Add @CircuitBreaker in the WalletService method which is making a remote call

WalletService.java



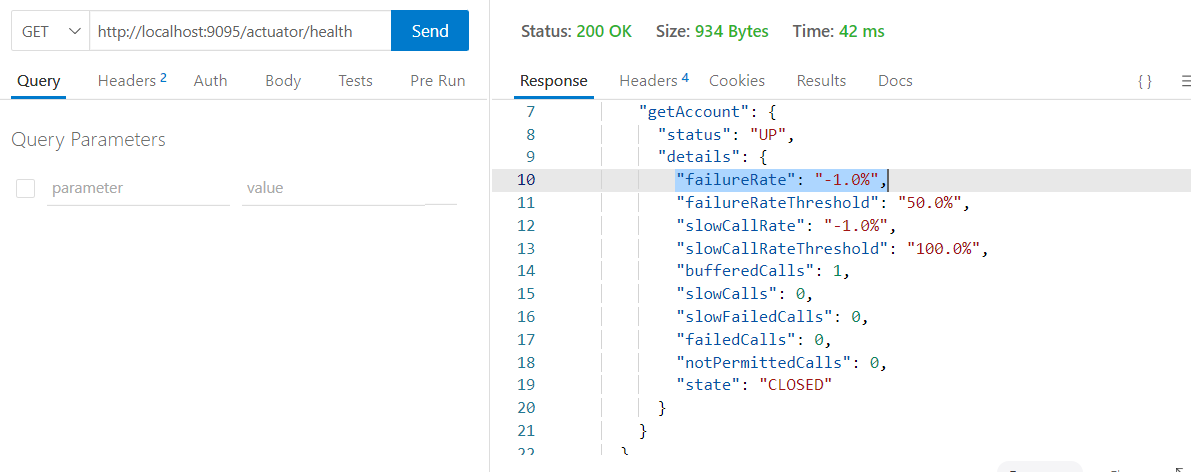
application.properties



Programs to launch

1. Service Discovery
2. Configuration Server
3. Account Microservice
4. Wallet Microservice

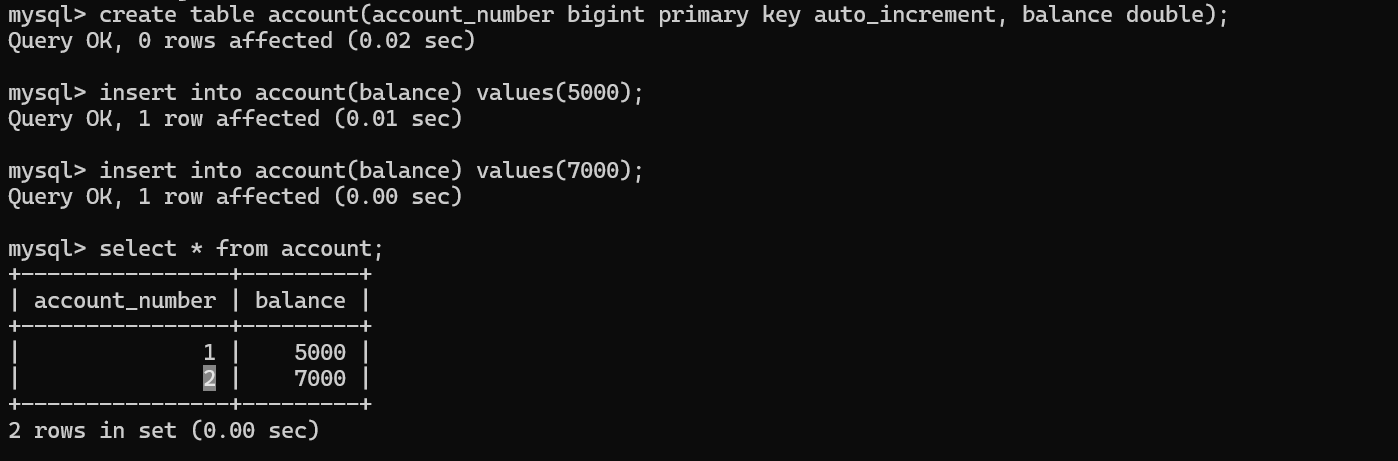
You can see the circuit breaker status in /actuator/health



Stop the Account Service and send few more requests minimum 5, then you can see the OPEN state

1. After 1 min you can see HALF\_OPEN
2. HALF\_OPEN to CLOSE state occurs only if the service is up

Create database table account in mysql & store some records



Activity:

Account Microservice should connect to the MySQL database using Spring Data JPA, however the datasource configurations must be pulled from the GIT and also the username & password must be in encrypted format, when wallet sends the account number it has to send only those numbers present in the database

Steps:

1. Using configuration server create the encrypted data for username & password & store them in your git repository (ensure you are able push the configuration file whose name matches to the spring.application.name of account service)
2. Add Spring Data JPA & MySQL library to the account microservice
3. Create an entity for account and map the account number & balance
4. Create a Repository that extends JpaRepository<Account, Long>
5. From the service layer you must get the Account balance using the account number
6. From the controller call the service layer method and return account number & balance in JSON format

Spring Security

Authentication: Verify the user

Authorization: Permissions for the user to access the resource

OAuth2: It is a token generated for Authentication & Authorization, but it is not a standard token

JWT: JSON Web Token, it follows some standards like Digital signature and encoding formats

/auth/user/userProfile: It can be accessed by the user having USER\_ROLE  
/auth/admin/adminProfile: It can be accessed by the user having ADMIN\_ROLE

A table called userinfo to store the user & admin information’s, it will have following columns  
name, password, email, roles

Steps to perform

1. Launch the application
2. Store the user info like name, password, email, roles

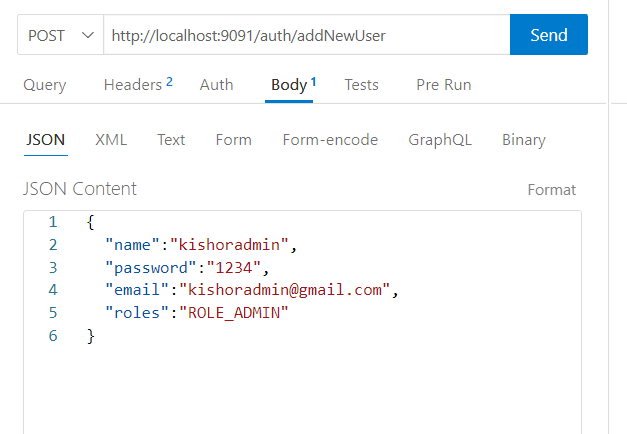
Note: Password will be stored in encrypted format

1. Generate the token by passing username & password

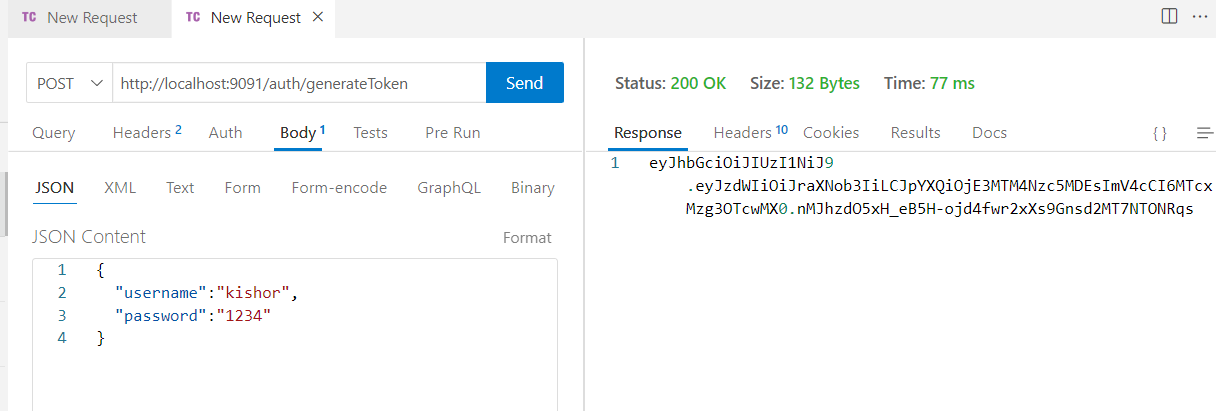
Note: username is name itself

1. Access the resource with & without JWT token and observe the output

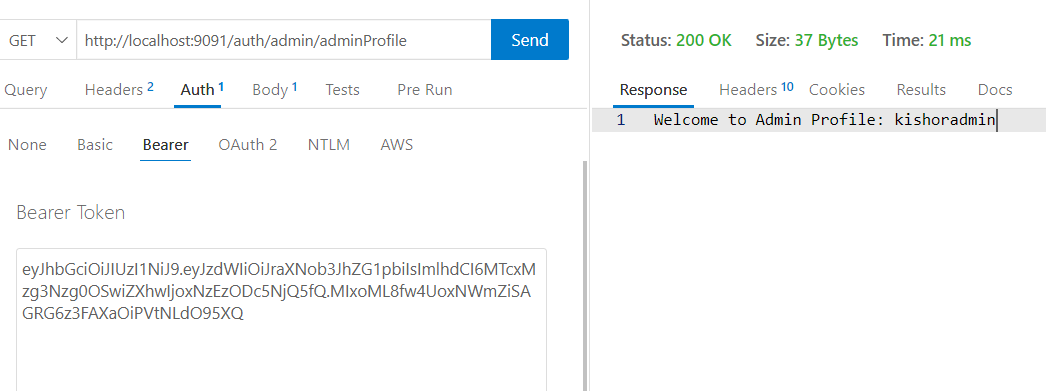
Add User



Generate the token



Access the resource with token



Asynchronous programming using Webflux

Synchronous programs usually blocks the threads while performing CPU intensive tasks like IO operations, DB operations,

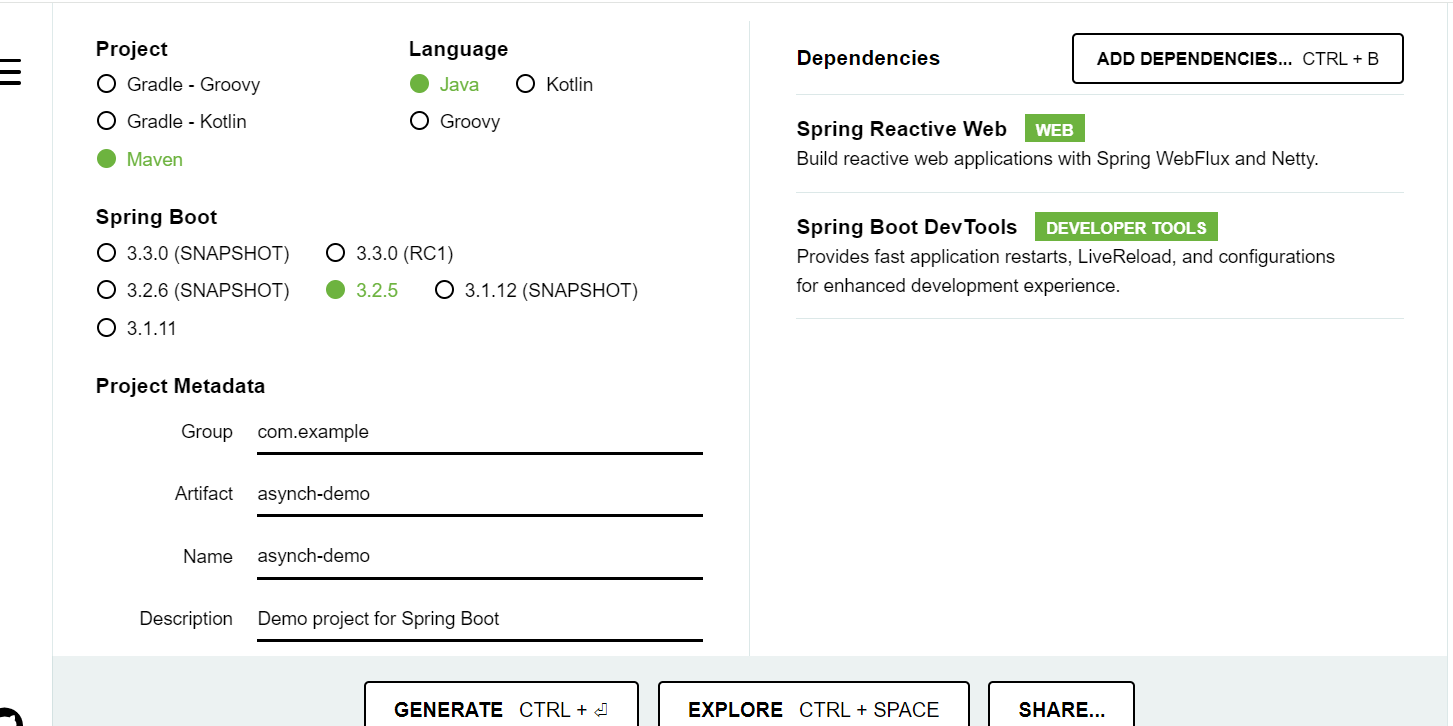
Asynchronous programs doesn’t block the threads while CPU intensive tasks, these threads are reused while performing CPU intensive tasks

In traditional servers the threads are blocked until the worker threads publishes the data to it

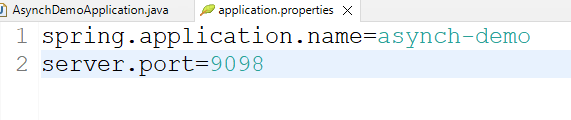
In Modern servers the threads are not blocked, because once the particular thread handles the request it doesn’t wait for worker threads to publish the data, instead it goes back to the thread pool to serve other requests, when the worker threads publish the data any one of the thread in the thread pool produces the result

Library

1. Web Flux - Netty server
2. Devtools

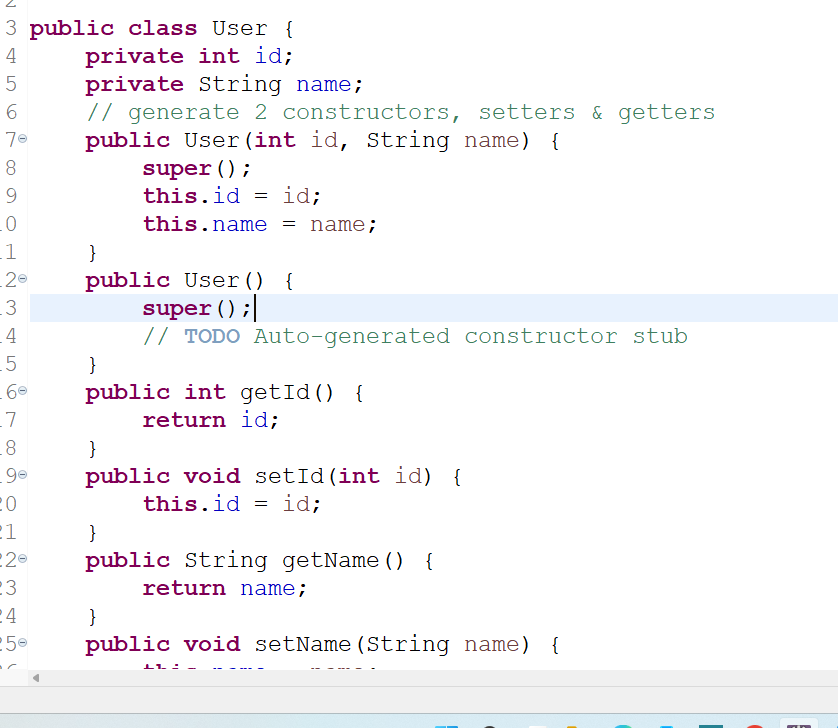


application.properties

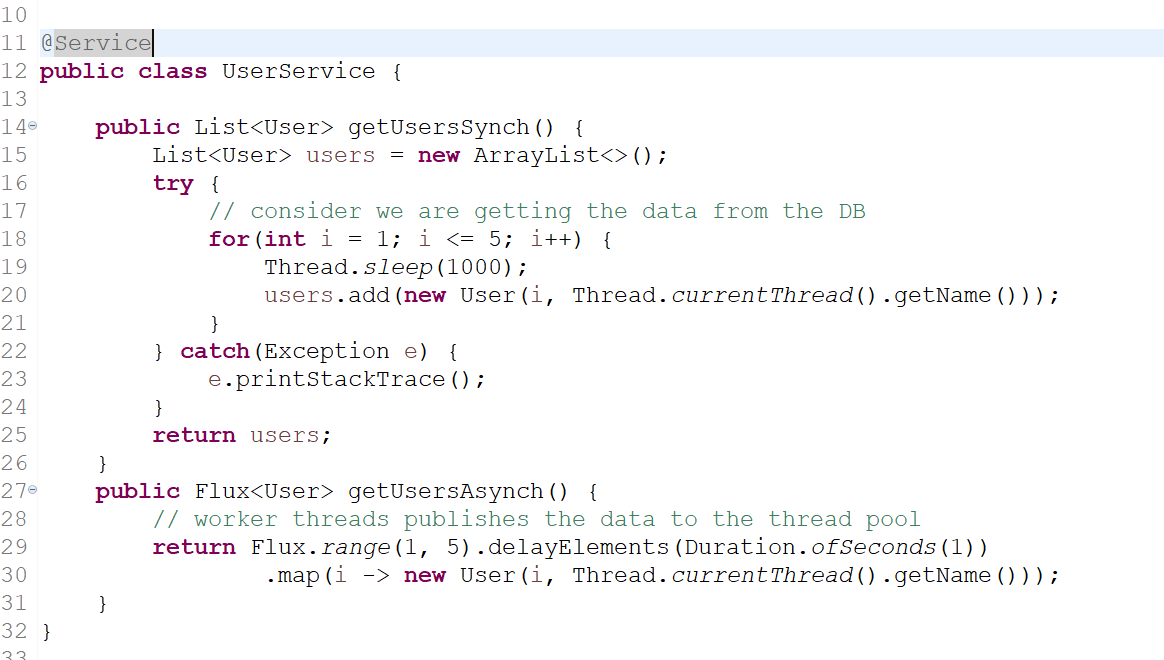


We will create a service layer that performs the operations synchronously & asynchronously, which returns multiple user objects.

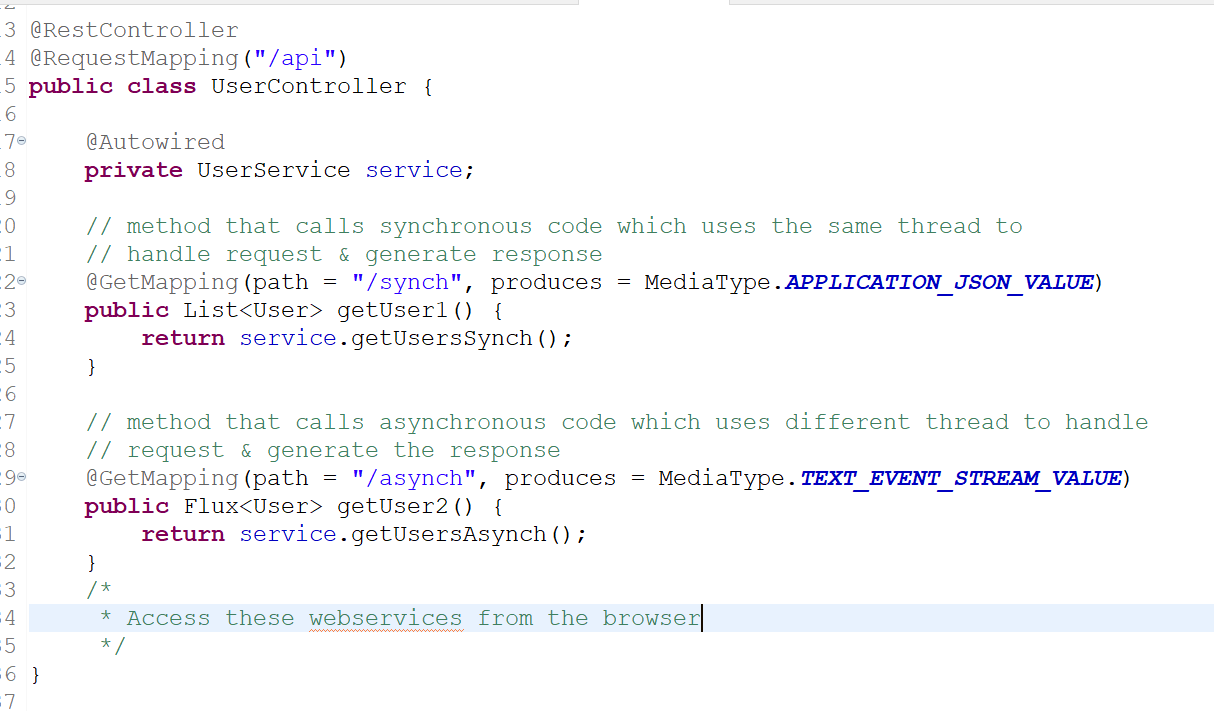
User.java



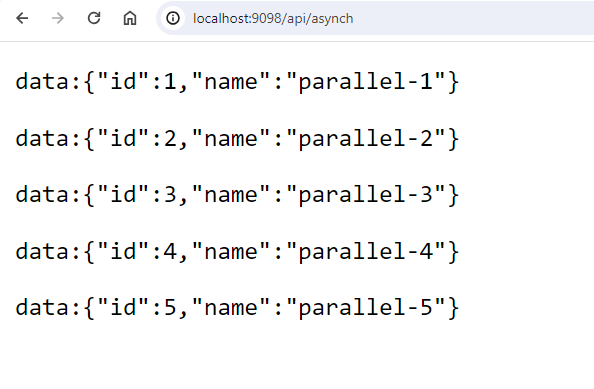
UserService.java



UserController.java



Output:



React.js

It is a Javascript library used to develop User Interfaces for Single Page Applications.

Single Page Applications are those applications where everything happens in one single page.

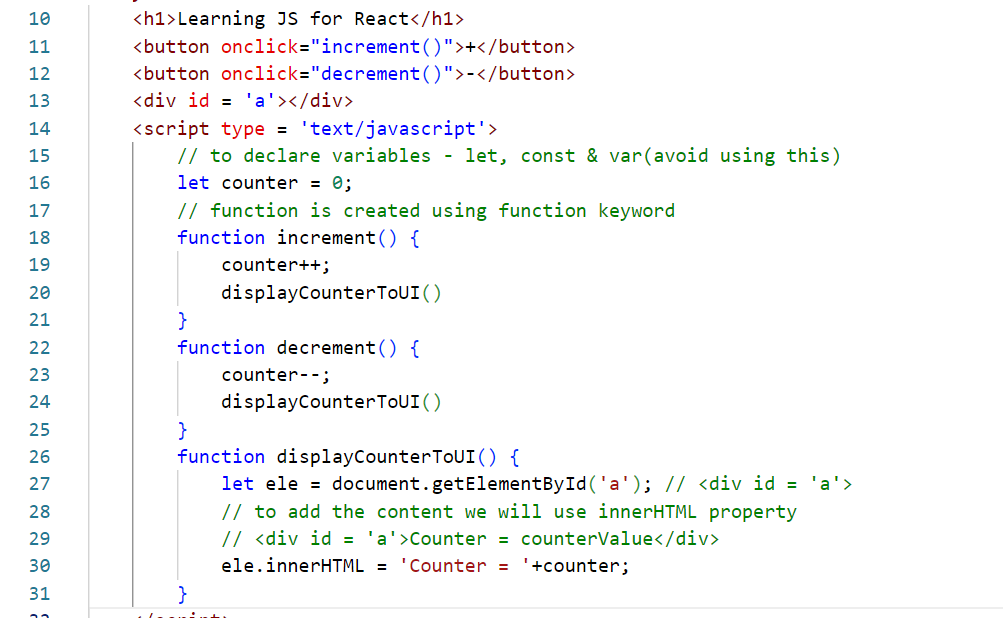
Javascript

It is a programming language used to develop User interfaces, all browsers understand Javascript, it takes care of adding dynamic effect to the web pages, other than Javascript browser understands HTML & CSS

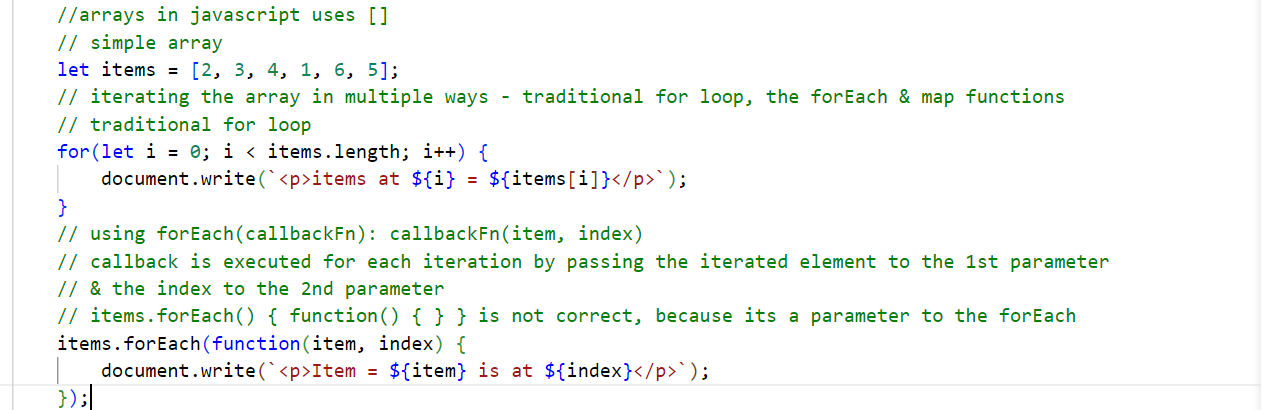
HTML: Displaying the content

CSS: Style the HTML

Javascript: Provide effects to the HTML & CSS at runtime



Arrays & Iterations



For Each: it does an internal iteration

Map: It also does an internal iteration, but it generates a new array after the iteration, so that you can use it for transformation

let array2 = items.map(function(element, index) { return value; } );

Arrow functions: These are used to simplify writing the callback functions, they use => between the method signature & the body, if callback function has only one statement then no need to use { } and return keywords

Suppose there’s a callback function with one line

function(x, y) {   
 document.write(x + “ “ +y);  
}

Arrow function for the above callback doesn’t need {}

(x, y) => document.write(x + “ “+y);

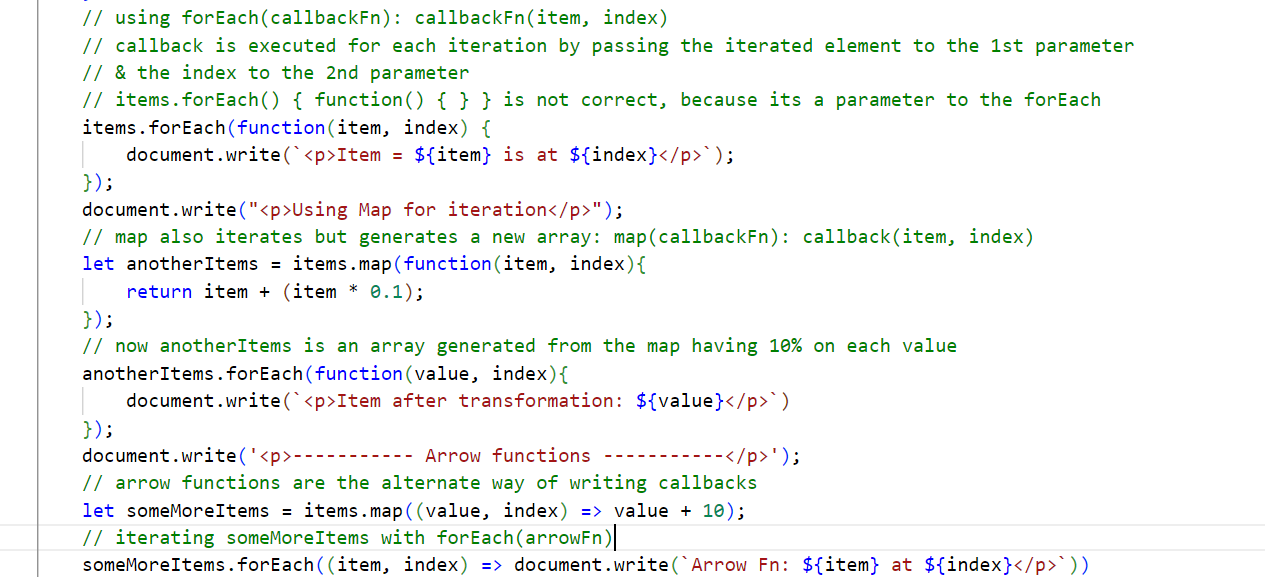
Suppose there’s a callback function with one line & returns a value

function(x, y) {   
 return (x + y);  
}

Arrow function for the above callback doesn’t need return & {}

(x, y) => (x + y);

However you must use { } & return if there are more than one line statement, though return is optional for some callbacks



String padding: whenever you want to add some extra strings in the beginning or end of a string you can use methods like  
- padStart()  
- padEnd()

998877XXXX: padding at the end

XXXX998833: padding in the beginning

Optional chain

It is used when you are accessing a property of an object which may or may not be present, earlier developers used if conditions to check for the property in the object, but it makes the code lengthier, which you can also check using ?.

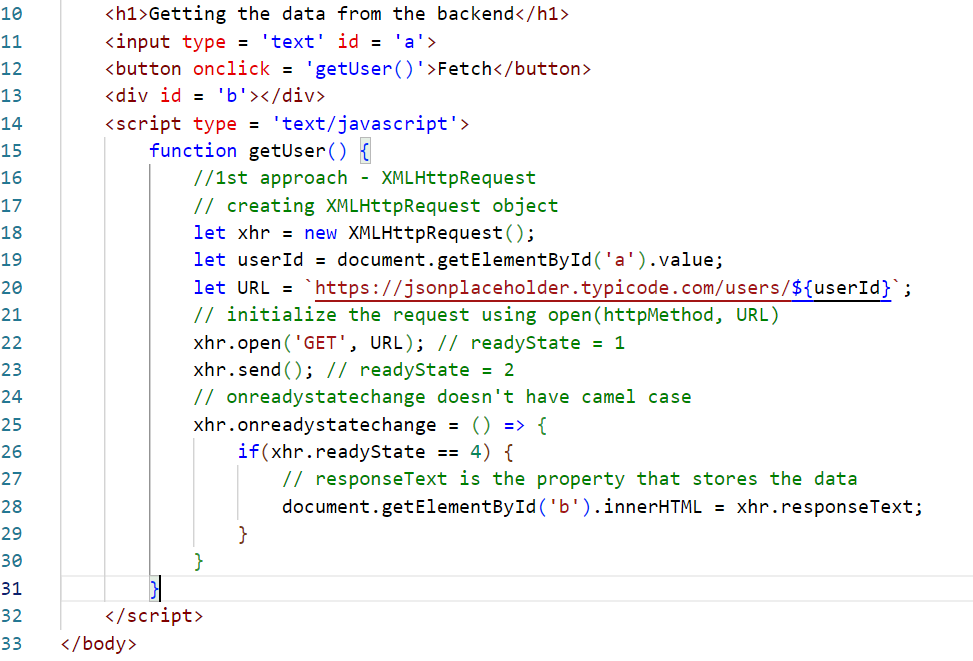
let arrays = [ { id: 200, name : “Sid”, address : { state: “KA”, city : “BLR” } },   
 {id: 300, name: “Chinmay”},   
 { id: 300, name : “Vivek”, address : { state: “MH”, city : “MBI” }}  
];   
forEach -> item.name, if(item.address != ‘undefined’) { item.address.state } // older approach

// newer approach is to use ?.  
forEach -> item.name, item.address?.state

How to access backend API’s from Javascript

Javascript started with using XMLHttpRequest then it used Promise and later async/await to access the backend API’s

XMLHttpRequest: The first approach used by Javascript to access the backend

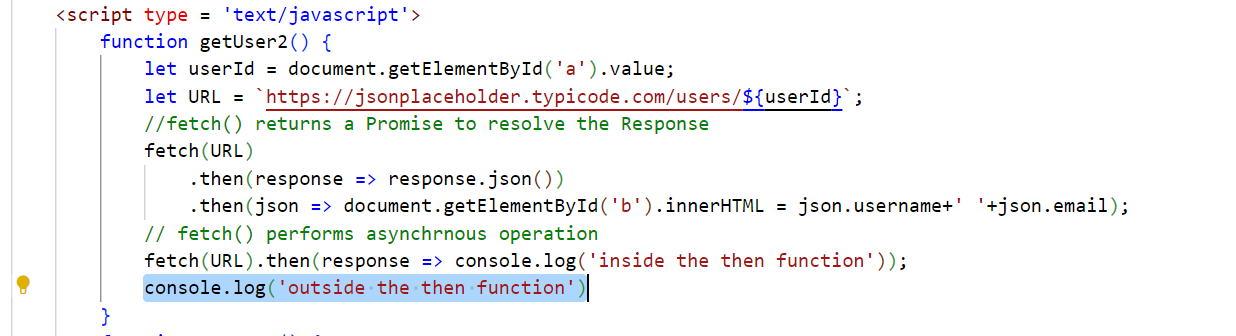


Promise: It is also used to make backend calls but it handles success & errors in 2 methods using .then(callbackFn) and .catch(callbackFn)

Javascript has introduced fetch() function to make HTTP calls that returns Promise, whether the Promise is fulfilled or rejected that can be identified using the callbackFn of .then or .catch

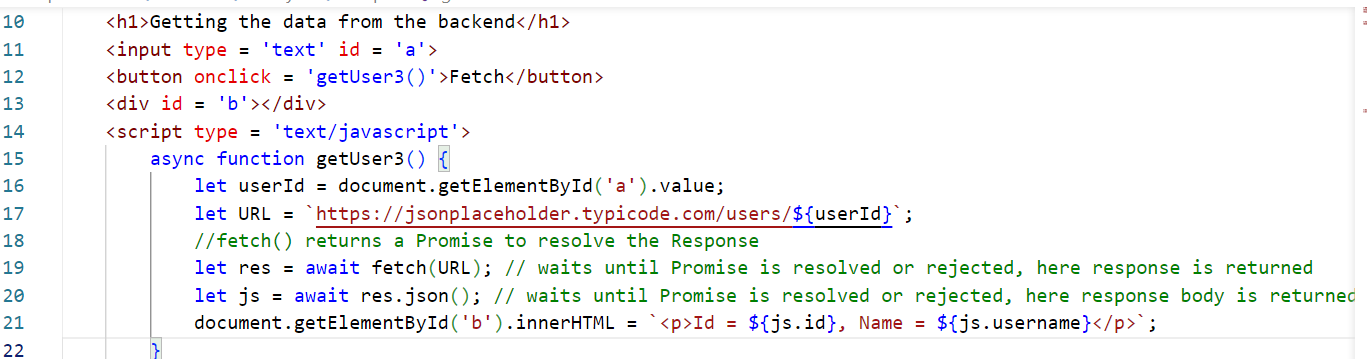
.then(callbackFn): The callback of then is called if the Promise is success/fulfilled

.catch(callbackFn): The callback of catch is called if the Promise is rejected



In Promise you must use .then() or .catch() to know the result

async/await: This also works like Promise, but it waits for Promise to resolve/reject and then goes to the next line



React.js

It provides rich Javascript library to create single page applications.

It uses HTML & JSX to create UI components

Components: These are the UI’s which user can see

JSX: Javascript XML it simplifies writing the HTML code in the Javascript, it supports all the Javascript features

It uses three libraries mainly

1. React
2. ReactDOM
3. Babel

React: It is used to create components

ReactDOM: It maps the components to the Real DOM by using Virtual DOM

Babel: It helps you to write JSX & converts to Javascript so that browser can understand

Virtual DOM (VDOM): It updates the Real DOM without refreshing the entire DOM tree by making changes only to the content that needs the change

React Tool Kit

This tool kit helps you to create an industry standard project with auto-reload features

You must create project using react tool kit using the following commands

npx create-react-app project-name

npm start >> package.json >> runs a script file for start

How to share the data from one component to another component in React.js

We can share the data in two ways

1. props: read-only data
2. state: read & write both

Every component will have an inbuilt property called props to receive the value from another component

If App wants to share an object to User: App -> object -> User

Then App component can share like this:-

<User name = “Alex” age = “25” /> Here props = {name = “Alex”, age = “25” }  
<User name = “Raj” age = 35” /> Here props = {name = “Raj”, age = “35” }

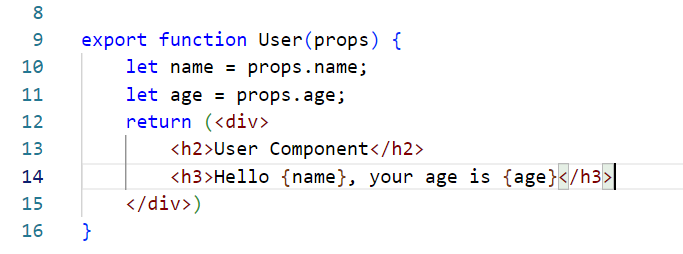
In User component you can read name & age through props

function User(props) {   
 name = props.name;  
 age = props.age  
 Display name & age  
}

A component can also share a complex objects or array to the props

let object = {id:100, name:”Raj”, salary:35000}  
<Employee obj = {object} />  
  
function Employee(props) {  
 let e = props.obj; // e = obj = object = {id:100, name:”Raj”, salary:35000}  
}

Simple.js



How to iterate array of elements in React.js

You will use List & Keys: List is a collection of iterated item inside an element & key is an unique id for each iterated item

In React.js you need to call the map function and create element for each iteration

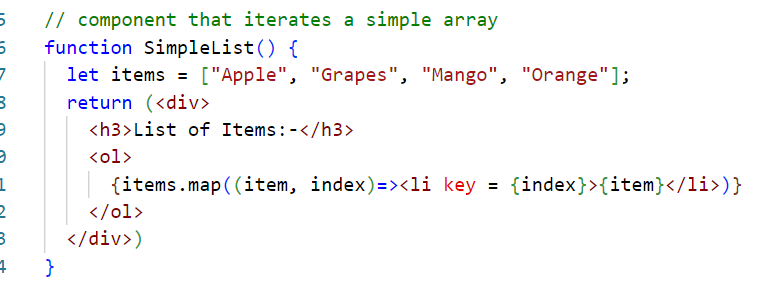
let fruits = [“Orange”, “Mango”, “Grapes”, “Apple”];

let list = fruits.map( (fruit, index) => <li key = {index}>{fruit}</li>);

<ol> { list } </ol> This results in

<ol>  
 <li key = “0”>Orange</li>  
 <li key = “1”>Mango</li>  
 <li key = “2”>Grapes</li>  
 <li key = “3”>Apple</li>  
</ol>

You can also iterate inside the HTML element as below  
<ol>  
 { fruits.map((fruit, index) => <li key = {index}>{fruit}</li>) }  
</ol>





Adding styles in the React.js

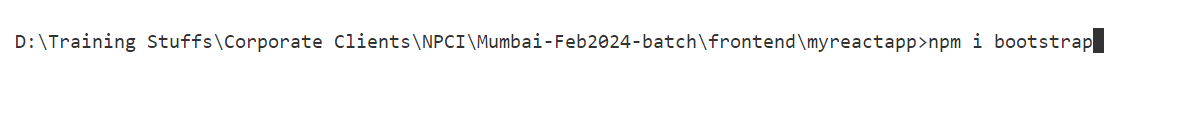
You can add an online CDN link in a global stylesheet index.css or you can download the css library and add its path in the index.css(recommended, because it will part of the project)

npm install bootstrap >> This downloads the bootstrap inside node\_modules

index.css

@import url(path of bootstrap in node\_modules)

Installing bootstrap

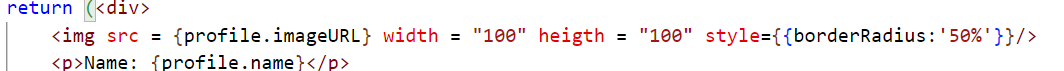


Note: bootstrap provides 10000+ inbuilt classes, in React.js we must use className as an attribute instead of class, because class is a keyword in Javascript

<p class = ‘text-danger’> is incorrect inside the JSX, we must use

<p className = ‘text-danger’>

You can also add your own styles using style attribute



States & Events

States: These are the data which components can modify, you can create states using an inbuilt React hook method useState

import { useState } from ‘react’;

let [name, setName] = useState(“”); // initial value of name is “”

let [age, setAge] = useStage(“”); // initial value of age is “”

name & age will store the data

setName & setAge is used to modify the data

ex: setName(value); setAge(value);

Events:

When you want to perform some action you can handle events, but in React.js these events are named as below:-

onClick, onChange, onBlur, onFocus, onMouseOver, and so on, it uses camel case

For these events you need to attach a callback which is executed when that event occur, it can be a reusable callback or inline callback.

handleClick = ( e ) => { some code }

<button onClick = {handleClick}>MyButton</button>

<button onClick = {( e ) => { some code } } >MyButton2</button>

Note: e is the event that is generated, you can access the element that generated the event

ex: e.target.value, e.prefentDefault

axios:

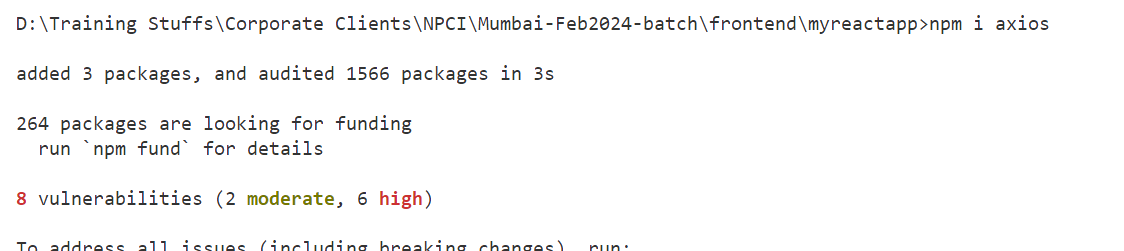
It is used to make HTTP calls to the backend, it provides promise based http methods

axios.get(URL).then(callbackFn).catch(callbackFn)  
axios.post(URL, data).then(callbackFn).catch(callbackFn)  
axios.delete(URL, data).then(callbackFn).catch(callbackFn)  
axios.put(URL, data).then(callbackFn).catch(callbackFn)

Note: By default backend services restrict access for the external application running on a different domain, if React/any application tries to access a backend service that restricts access then it gets CORS (Cross Origin Resource Sharing) error, if it doesn’t restrict then backend can share data to the applications.

Note: JSON placeholder doesn’t restrict

Install axios using npm install axios



Import axios in the JS file

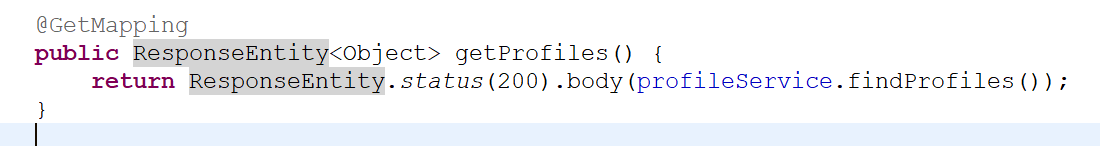
import axios from ‘axios’;

You can make HTTP calls with the URL

axios.get(URL).then(..).catch(…)   
axios.post(URL, data).then(..).catch(..)

Backend program must provide access using @CrossOrigin annotation

Find all profiles



Routers: It helps you to navigate from one component to another component

You must install Router using npm install react-router-dom

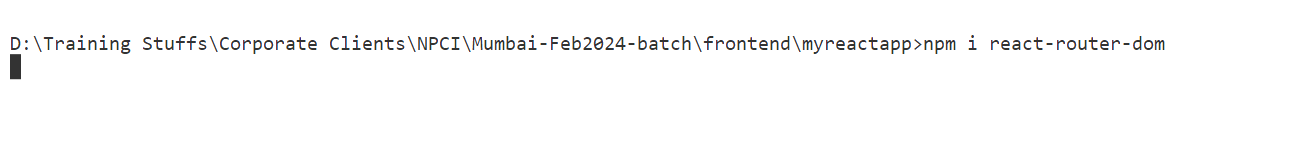
React Router DOM gives some inbuilt components & hook functions as below

1. <BrowserRouter>: A router where all the components must be inside this
2. <Link>: To create links
3. <Routes>: Keeps collection of component which shows any one based on the Link you click
4. <Route>: part of <Routes> which will have the component configured with a link
5. useParams: hook function to read path parameters
6. useNavigate: hook function to programmatically navigate

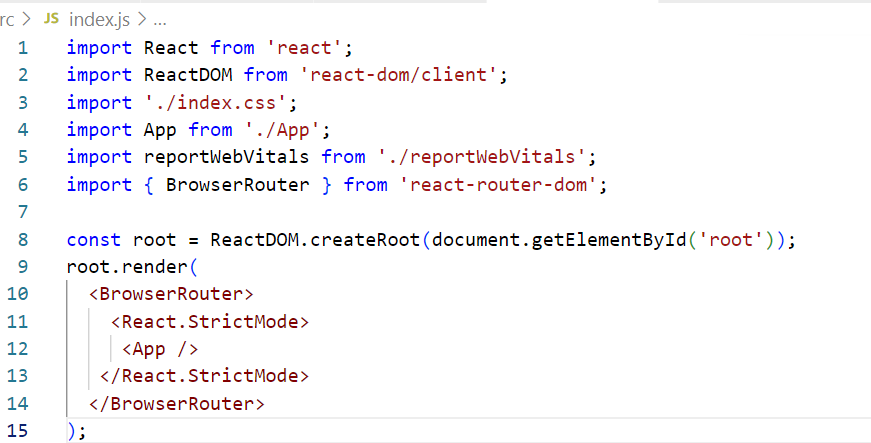
Keep all the components inside the router  
<BrowserRouter>  
 <App />  
</BrowserRouter>  
Create link for each component using <Link>  
<Link to = “/register”>Registration</Link>, <Link to = “/list”>Profiles</Link>  
Configuring the routes for each component

<Routes>  
 <Route path = “/register” element = { <ProfileRegistration /> } />  
 <Route path = “/list” element = { <ProfileList />} />  
</Routes>

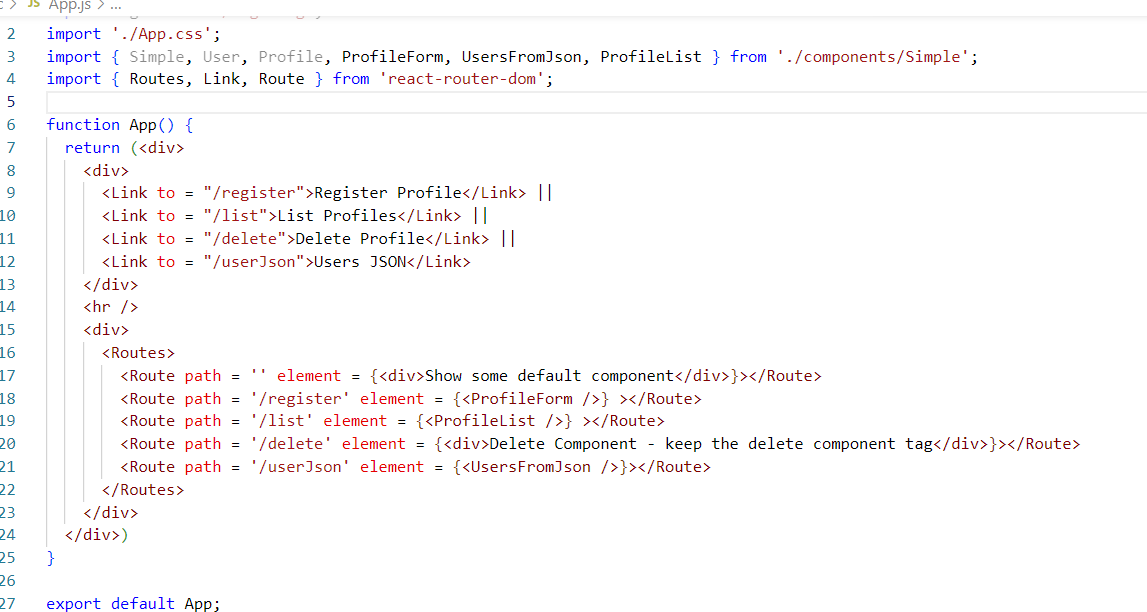
Installing the react-router-dom



Place root component inside <BrowserRouter> i.e., in index.js



Create Links, Routes & Route in the App.js



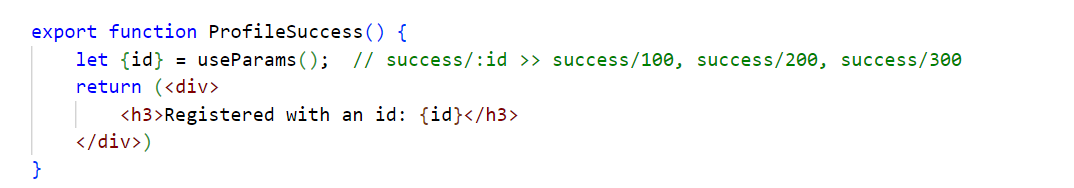
Programmatic navigation

import { useNavigate } from ‘react-router-dom’;

let navigate = useNavigate();

navigate(“/success/”+id);

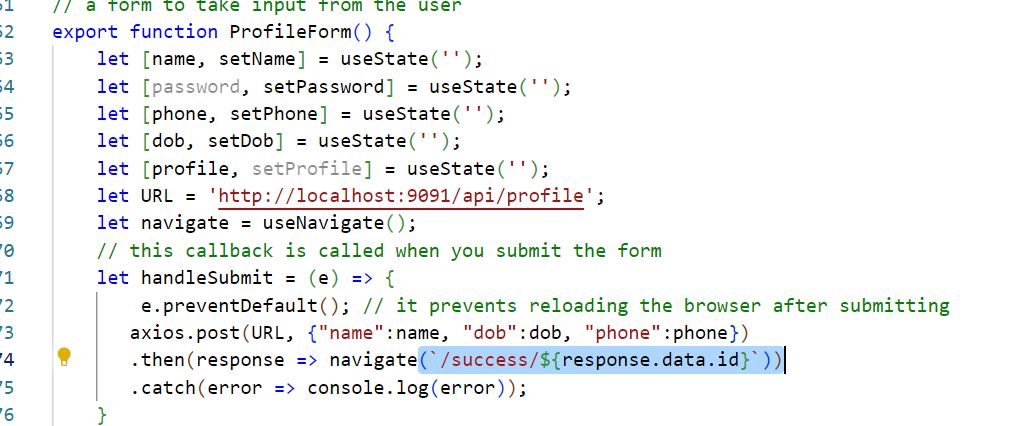
You can have a success component that opens after registration



Here there must be a component with /success/:id  
<Route path = “/success/:id” element = {<ProfileSuccess /> } >

Now ProfileSuccess open on success/200, success/300 or etc.

ProfileRegistration navigates to the success/:id



Configure the <Route> for /success/:id

