Passwordless Biometric Authentication

Submitted in partial fulfillment of the requirements for the degree of

Master of Technology

in

SOFTWARE ENGINEERING

by

Vinay.R
14MSE0325

Under the guidance of
Prof. Meenatchi.S
SITE
VIT, Vellore.

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

April, 2019

DECLARATION

I hereby declare that the thesis entitled "**Passwordless Biometric Authentication**" submitted by me, for the award of the degree of Master Of Technology in Software Engineering to VIT is a record of bonafide work carried out by me under the supervision of Meenatchi.S.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date : 16-04-2019

Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled "**Passwordless Biometric Authentication**" submitted by Vinay.R & 14MSE0325, SITE, VIT University, for the award of the degree of Master Of Technology in Software Engineering is a record of bonafide work carried out by him under my supervision during the period, 01. 12. 2018 to 16.04.2019, as per the VIT code of academic and research ethics.

The contents of this report has not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place: Vellore

Date: 16-04-2019

Signature of HOD **Signature of Guide**

Internal Examiner **External Examiner**

ACKNOWLEDGEMENTS

It is my pleasure to express with a deep sense of gratitude to Mr. Vivek Dhayalan, Architect, Pramati Technologies for his constant guidance and it is a great opportunity on my part to work with an intellectual and expert in the field.

I would like to express my gratitude to Mr.Vasanth, Senior Manager, Technical,Pramati Technologies for providing an environment to work in and for his inspiration
during the tenure of the course.

I would like to express my gratitude to Prof.Meenatchi.S, Assistant Professor (Selection Grade), SITE, Vellore Institute of Technology, for her constant guidance, continual encouragement, understanding, more than all, She taught me patience in my endeavour. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual in the field of Software Engineering.

In jubilant mood I express ingeniously my whole-hearted thanks to **Dr.S.SreeDharinya**, Head of the Department, **Prof. Swarna priya R.M** and **Prof. Karthikeyan J**, Main Project Coordinators all teaching staff and members working as limbs of our university for their not-self-centred enthusiasm coupled with timely encouragement showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully

Place: Vellore **Name of the student**
Date: 16-04-2019 VINAY.R

Executive Summary

In recent years, there have been so many cases of stolen or hacked passwords. As a result, passwords are becoming a larger part of the problem rather than the solution to keeping users' information safe.

Additionally, more websites require users to create an account, which means users have to juggle multiple passwords and remember which ones go to each account. As a result, users who want to remember their accounts usually choose a password they're comfortable with (say their birthday) or credentials they've used for other accounts.
All of these challenges can be better avoided with passwordless authentication like fingerprint scanning and 2FA.

CONTENTS

List of Abbreviations

API	Application programming interface
REST	Representational state transfer
2FA	Two-Factor authentication
MVC	Model View Controller
SSO	Single-Sign-On

1. INTRODUCTION

1.1.
OBJECTIVE:

Now a days companies are approaching the Customer Login Experience as new technologies like Passwordless, Multi Factor Authentication, and social login options become more prevalent. It is found that companies are eager to embrace and adopt methods for streamlining the login process, suggesting that we should see an increase in passwordless usage in the near future.
As the industry evolves to a possibly passwordless future, the current state of the industry still finds a predominant use of passwords as the primary means of logging in. Before making the leap into passwordless, many companies have implemented social login as an easier way to consolidate secure logins.
The login experience is continually changing based on user demand and the need to protect against today's sophisticated cybercriminal landscape. Passwordless is a signal of the kind of industry change we are all heading towards.

Authentication with Fingerprint:
With this form of authentication, the user is asked to place their finger on a mobile
device. A unique key pair is generated on the device and a new user is created on the server that maps to the key. A session is initiated and the user is logged in.

1.2 MOTIVATION:

To make the customer authentication experience easy and smooth passwordless sign in
came into existence. Password-based authentication opens the door for numerous user errors that negatively affect an organization's security. Under password-based authentication, users can set and use short or easily guessed passwords, commingle personal and business

passwords, or reuse the same password across multiple applications and systems. With passwordless authentication, organizations can avoid all of these vulnerabilities.
Motivation for choosing passwordless Authentication.
Better Security: User controlled passwords are a major vulnerability, users reuse passwords, are able to share them with others. Passwords are the biggest attack vector and are responsible for 81%attacks.

Reduction in Total Cost of Ownership (TCO) : Passwords are expensive, they require constant maintenance from IT staff, removing passwords will reduce support tickets and free IT to deal with real problems.

IT Gains Control and Visibility: Phishing, reuse, and sharing are common issues when relying on passwords, with passwordless.

User Experience : passwordless authentication means no more user memorized secrets, streamline the authentication process.

1.3 BACKGROUND:

Undoubtedly the first things that biometric technology makes our lives easy is through enhancing security. Biometric authentication and identification are changing the way we do and see things. Smart mobile devices are now adding an extra layer of security through fingerprint scanner, voice, or facial recognition. Recently, Samsung has added an iris scanner in their latest smartphone.
The data center of Google uses multimodal biometric verification to keep track of the security. These days, many corporations offer several related but independent services. These services can be configured to leverage Single Sign On To enhance the user experience and security at the same time. SSO is an access control approach in which a user has to present his/her credentials only once to log in to the services, and he/she can seamlessly switch to other related services without having to provide identity credentials until the session lasts. This approach dramatically improves user experience and saves

considerable time and efforts of providing credentials every time a user switches to another service. When single sign-on is implemented using biometrics, it becomes the most user friendly and secure method of authentication.

2. PROJECT DESCRIPTION AND GOALS

OAUTH:

Oauth allows an end user's account information to be used by third-party services, such as Facebook, without exposing the user's password. OAuth acts as an intermediary on behalf of the end user, providing the service with an access token that authorizes specific account information to be shared.

OAuth allows notifying a **resource provider** (e.g. Facebook) that the **resource owner** (e.g. you) grants permission to a **third-party** (e.g. a Facebook Application) access to their **information** (e.g. the list of your friends).
Basically there are three parties involved: oAuth Provider, oAuth Client and Owner.

- OAuth Client (Application Which wants to access your credential)
- OAuth Provider (eg. facebook, twitter...)
- Owner (the person with facebook,twitter.. account)

JSON Web Token (JWT) a JSON-based open standard (RFC 7519) for creating access tokens that assert some number of claims. For example, a server could generate a token that has the claim "logged in as admin" and provide that to a client. The client could then use that token to prove that it is logged in as admin. The tokens are signed by one party's private key *(usually the server's)*, so that both parties *(the other already being, by some suitable and trustworthy means, in possession of the corresponding public key)* are able to verify that the token is legitimate. The tokens are designed to be compact, URL-safe,and usable especially in a web browser single-sign-on (SSO) context. JWT claims can be typically used to pass identity of authenticated users between an identity provider and a service provider, or any other type of claims as required by business processes.

3.TECHNICAL SPECIFICATION

The technical specifications include

- Eclipse IDE
- Java-For Programming
- Spring Boot
- Postman
- Maven

About Maven

Maven is a build automation tool used primarily for java projects.

Maven addresses two aspects of building software: first, it describes how software is built and second, it describes its dependencies.

Unlike earlier tools like Ant, it uses conventions for the build procedure, and only exceptions need to be written down

Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories such as the Maven Central Repository, and stores them in a local cache.

POSTMAN:

Postman allows you create collections of integration tests to ensure your API is working as expected. Tests are run in a specific order with each test being executed after the last is finished. For each test, an HTTP request is made and assertions written in javascript are then used to verify the integrity of your code. Since the tests and test assertions are written in JavaScript, we have freedom to manipulate the received data in different ways, such as creating local variables or even creating loops to repeatedly run a test

Sample Response in Postman:

Adding Data:

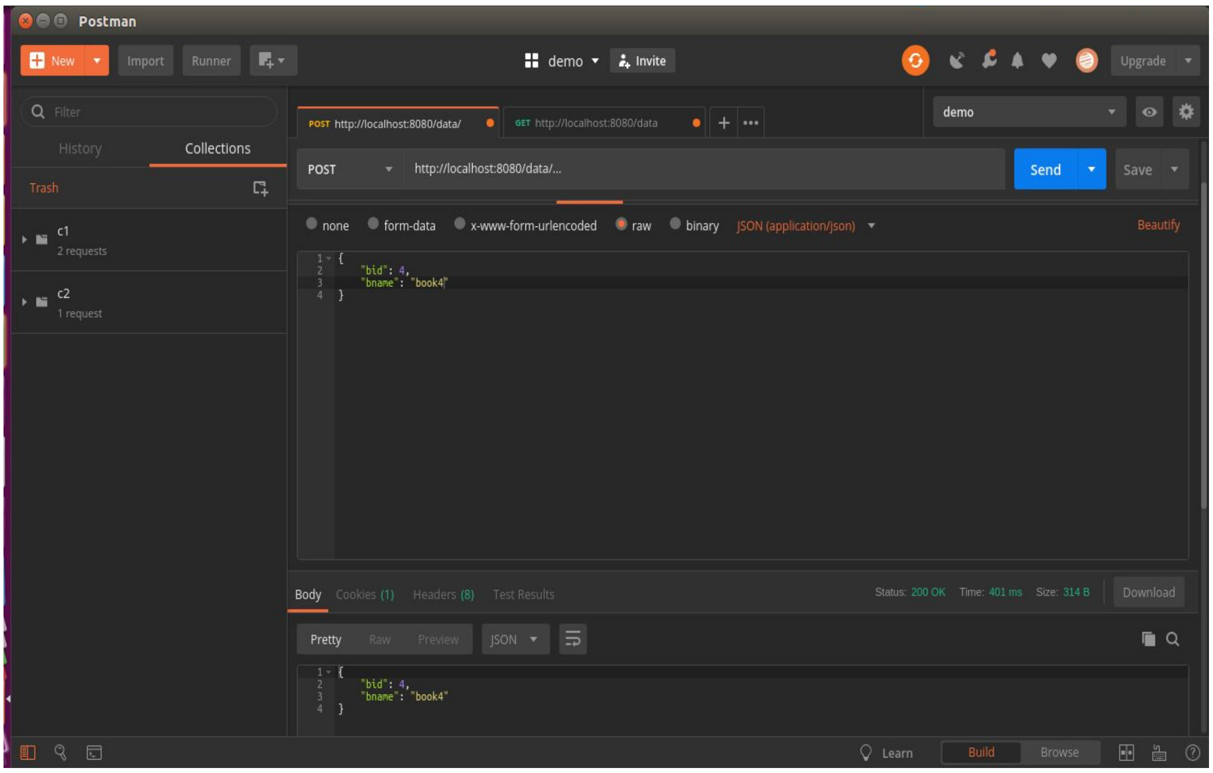
Definition:

POST/data Sample Data

```
 {
     "bid": 4,
     "bname": "book4"
   }
```

RESPONSE

```
{
   "bid": 4,
   "bname": "book4"
}
```



POST http://localhost:8080/data/

GET http://localhost:8080/data

POST http://localhost:8080/data/...

none form-data x-www-form-urlencoded raw binary JSON (application/json)

```
{
    "bid": 4,
    "bname": "book4"
}
```

Status: 200 OK Time: 401 ms Size: 314 B

```
{
    "bid": 4,
    "bname": "book4"
}
```

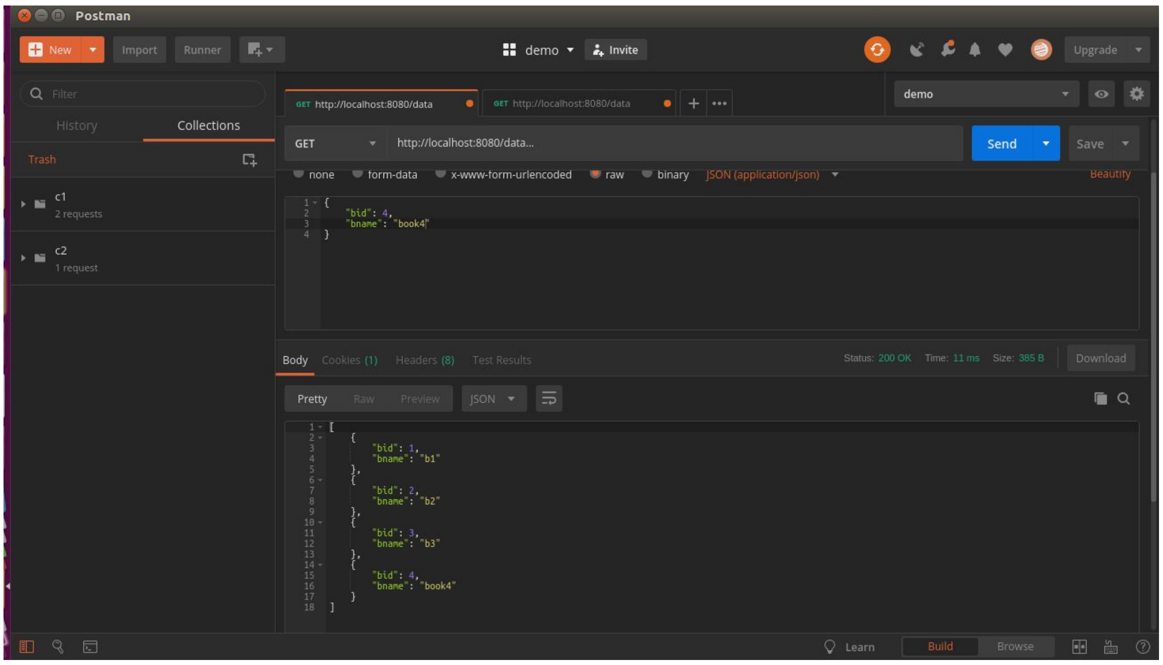
Figure: 3.1 POST response in Postman

Getting complete Data

Definition

GET/data RESPONSE

```
[
   {
      "bid": 1,
      "bname": "b1"
   },
   {
      "bid": 2,
      "bname": "b2"
   },
   {
      "bid": 3,
      "bname": "b3"
   },
]
```



```
{
    "bid": 4,
    "bname": "book4"
}
```

Status: 200 OK   Time: 11 ms   Size: 385 B

```
[
    {
        "bid": 1,
        "bname": "b1"
    },
    {
        "bid": 2,
        "bname": "b2"
    },
    {
        "bid": 3,
        "bname": "b3"
    },
    {
        "bid": 4,
        "bname": "book4"
    }
]
```

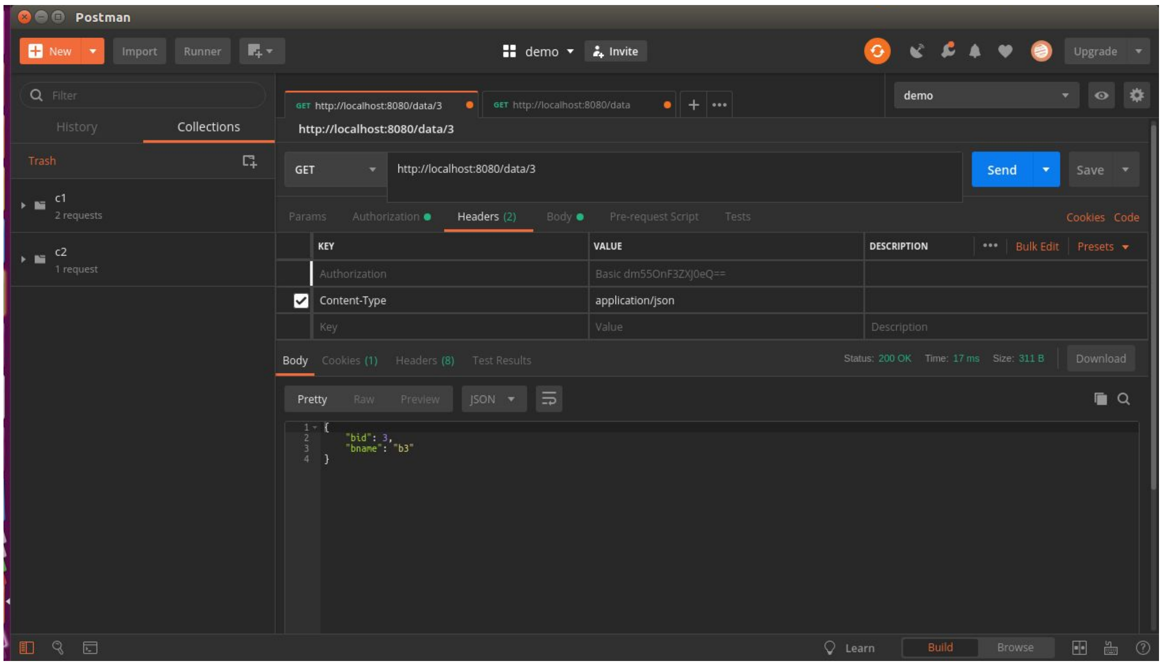
Figure: 3.2 GET response in Postman

Getting Desired Data

Definition

GET/data/{id} SAMPLE RESPONSE

```
{
   "bid": 3,
   "bname": "b3"
}
```



GET http://localhost:8080/data/3

GET http://localhost:8080/data

Environment: demo

http://localhost:8080/data/3

GET http://localhost:8080/data/3

Send | Save

Params | Authorization | Headers (2) | Body | Pre-request Script | Tests | Cookies | Code

	KEY	VALUE	DESCRIPTION
	Authorization	Basic dm55OnF3ZXJ0eQ==	
✓	Content-Type	application/json	
	Key	Value	Description

Body | Cookies (1) | Headers (8) | Test Results

Status: 200 OK Time: 17 ms Size: 311 B Download

Pretty | Raw | Preview | JSON

```
{
    "bid": 3,
    "bname": "b3"
}
```

Figure: 3.3 GET specific response with id in Postman.

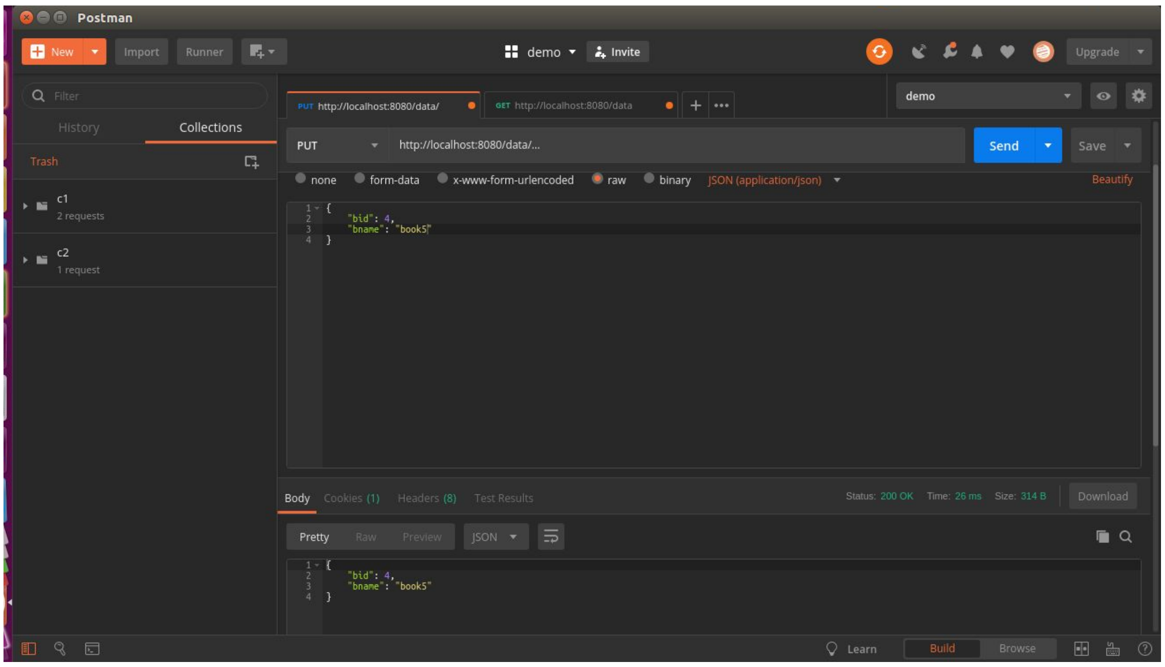
Updating the Data

Definition

PUT/data RESPONSE

```
{
   "bid": 2,
   "bname": "changed"

}
```



PUT http://localhost:8080/data/...

none, form-data, x-www-form-urlencoded, [✓ raw], binary — JSON (application/json)

```
{
    "bid": 4,
    "bname": "book5"
}
```

Body Cookies (1) Headers (8) Test Results   Status: 200 OK Time: 26 ms Size: 314 B

JSON

```
{
    "bid": 4,
    "bname": "book5"
}
```

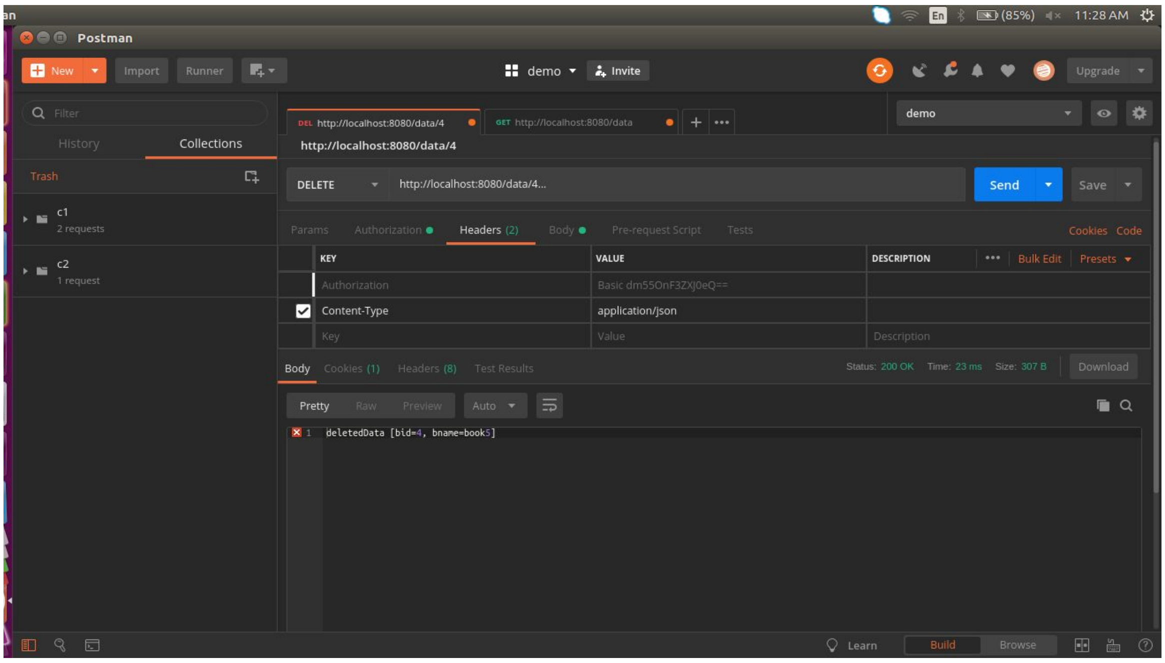
Figure: 3.4 PUT response in Postman(Update)

Deleting specific entry:

Definition:

DELETE/data/{id} RESPONSE

deletedData [bid=4, bname=book5]



DELETE http://localhost:8080/data/4

	KEY	VALUE	DESCRIPTION
	Authorization	Basic dm55OnF3ZXJ0eQ==	
✓	Content-Type	application/json	

Status: 200 OK Time: 23 ms Size: 307 B

```
deletedData [bid=4, bname=book5]
```

Figure: 3.5 DELETE with id response in Postman

SPRING BOOT:

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.

FEATURES OF SPRING BOOT:

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks and externalized configuration
- Absolutely no code generation and no requirement for XML configuration

4.DESIGN APPROACH AND DETAILS

MVC ARCHITECTURE:

The MVC architectural pattern has existed for a long time in software engineering. All most all the languages use MVC with slight variation, but conceptually it remains the same. MVC stands for Model, View and Controller. MVC separates the application into three components - Model, View and Controller.

Model: Model represents the shape of the data and business logic. It maintains the data of the application. Model objects retrieve and store model state in a database.

<u>Model is a data and business logic.</u>

View: View is a user interface. View display data using model to the user and also enables them to modify the data.

<u>View is a User Interface.</u>

Controller: Controller handles the user request. Typically, users interact with View, which in-turn raises appropriate URL request, this request will be handled by a controller. The controller renders the appropriate view with the model data as a response.

<u>Controller is a request handler.</u>

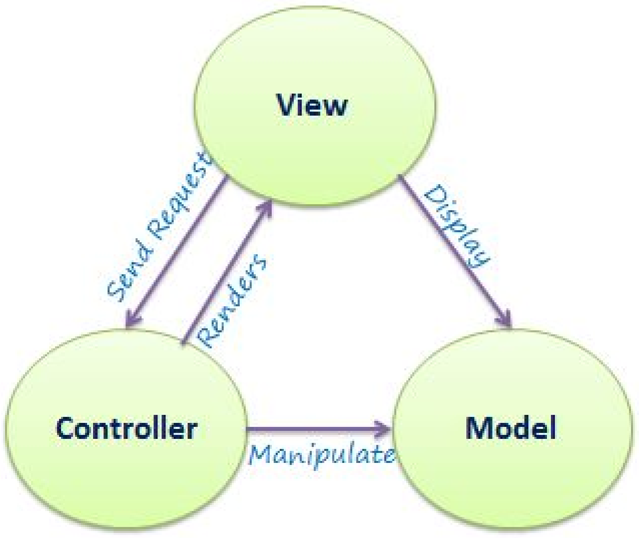


Figure: 4.1 MVC Architecture

4.1 Design Approach / Materials & Methods

SPRING MVC:

A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

Model - A model contains the data of the application. A data can be a single object or a collection of objects.

Controller - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.

View - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.

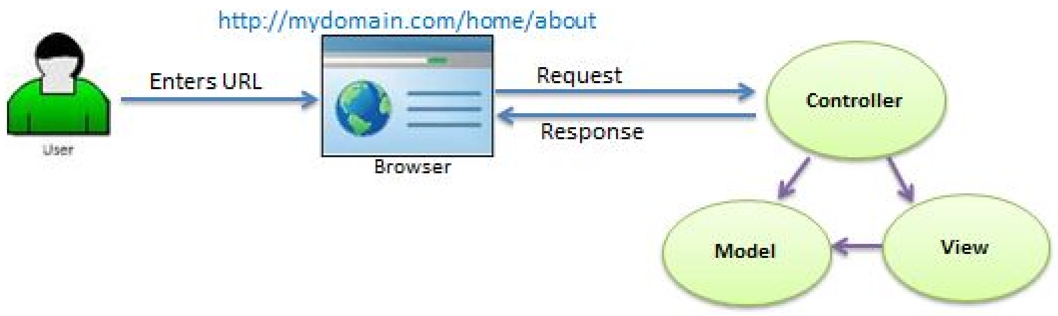


Figure: 4.1.1 Request Handling in MVC

4.2 Codes and Standards

Sample Code:

MODEL:

```
package com.practice.demo;

import javax.persistence.Entity;

import javax.persistence.Id;

@Entity

public class Data {
        @Id

        private int bid;

        private String bname;



        public int getBid() {
```

```
		return bid;
	}
	public void setBid(int bid) {
		this.bid = bid;
	}
	public String getBname() {
		return bname;
	}
	public void setBname(String bname) {
		this.bname = bname;
	}
	@Override
	public String toString() {
		return "Data [bid=" + bid + ", bname=" + bname + "]";
}
}
```

VIEW:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
```

```
<form action="data">
<input type="text" name="bid"><br>
<input type="text" name="bname"><br>
<input type="submit"><br>
</form>
<form action="data">
<input type="text" name="bid"><br>
<input type="submit">
</form>
</body>
</html>
```

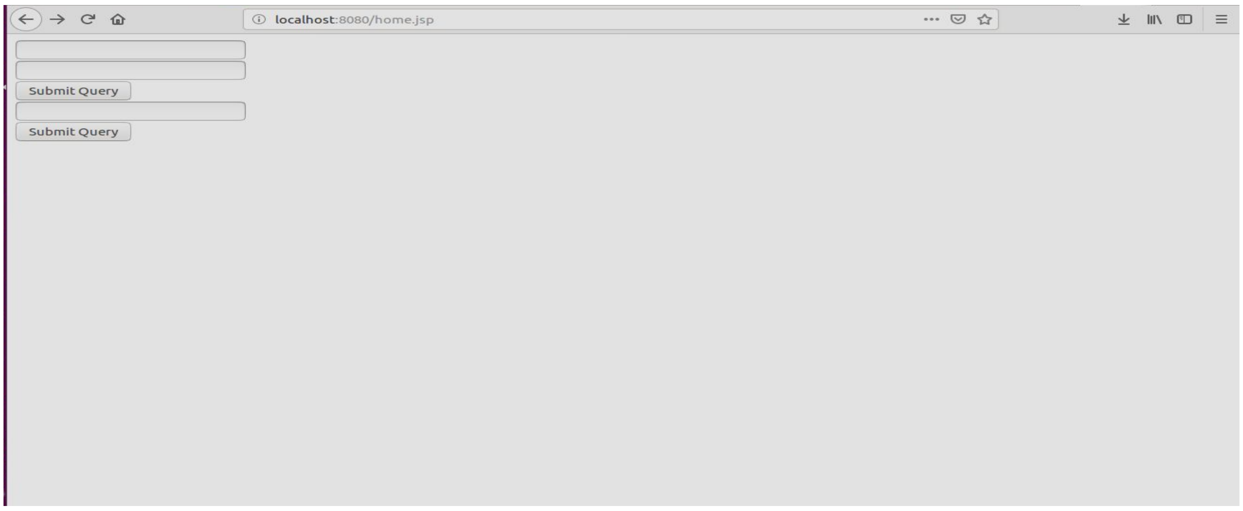


Figure:4.2.1 Sample View Page

CONTROLLER:

```
package com.practice.demo;
@RestController
public class DataController {
 @RequestMapping("/")
 //Displays home
 public String home() {
```

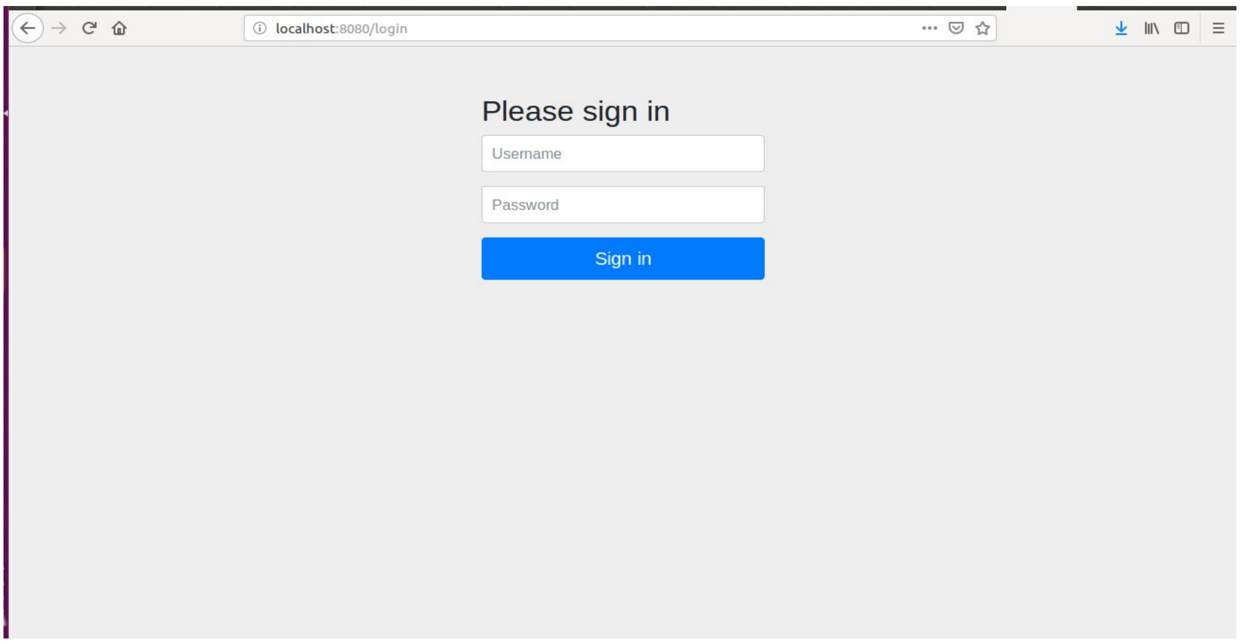
```
 return "welcome";
}
@Autowired
DataRepo datarepo;
//@PostMapping is a composed annotation that acts as a shortcut for
@RequestMapping(method = RequestMethod.POST)
//To add new entry to the data
@PostMapping(path = "/data")
public Data addData(@RequestBody Data entry) {
 datarepo.save(entry);
 return entry;
}

//To get the complete data
@GetMapping("/data")
@ResponseBody
public List < Data > getData() {
 return datarepo.findAll();
}
//To get specific details of data
//"id" to be given to get desired data
@RequestMapping("/data/{bid}")
@ResponseBody
public Optional < Data > getidData(@PathVariable int bid) {
 return datarepo.findById(bid);
}
//To delete specific entry
//"id" of the entry to be deleted is given
@DeleteMapping("/data/{bid}")
public String deleteData(@PathVariable int bid) {
 Data element = datarepo.getOne(bid);
 datarepo.delete(element);
 return "deleted" + element;
}
//To Update existing data
@PutMapping("/data")
public Data update(@RequestBody Data data) {
 datarepo.save(data);
 return data;
}
```

Security With Spring:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
 @Bean
 @Override
 protected UserDetailsService userDetailsService() {
 List < UserDetails > users = new ArrayList < > ();
users.add(User.withDefaultPasswordEncoder().username("vny").password("qwerty").roles("
USER").build());
```

```
 return new InMemoryUserDetailsManager(users);
 }
@Override
protected void configure(HttpSecurity http) throws Exception {
 http.authorizeRequests().antMatchers("/h2-console/**")
  .permitAll().and()
  .authorizeRequests()
  .antMatchers("/**").authenticated()
  .anyRequest().permitAll()
  .and()
  .formLogin()
  .and()
  .httpBasic();
 http.csrf().disable();
 http.headers().frameOptions().disable();
 }
}
```


localhost:8080/login

Please sign in

Figure:4.2.2 Security Login with Spring

The Passwordless authentication is carried out by the biometric authentication in the mobile devices in the application and using rest apis to check for authentication and the details of it cannot be disclosed as per industry standards

4.3 Constraints, Alternatives and Tradeoffs.

Constraint:

Conventional Programming

In software engineering, a monolithic application describes a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform.

Alternative:

Modular programming:

Modularity refers to the concept of making multiple modules first and then linking and combining them to form a complete system. Modularity enables re-usability and minimizes duplication. ... Modular programming is an extensively used concept based on modularity. Modularity is also a feature of object oriented programming.

Modular programming is the process of subdividing a computer program into separate sub-programs.

A module is a separate software component. It can often be used in a variety of applications and functions with other components of the system. Similar functions are grouped in the same unit of programming code and separate functions are developed as separate units of code so that the code can be reused by other applications.

Object-oriented programming (OOP) is compatible with the modular programming concept to a large extent. Modular programming enables multiple programmers to divide up the work and debug pieces of the program independently.

The benefits of using modular programming include:

1. Less code has to be written.

2. A single procedure can be developed for reuse, eliminating the need to retype the code many times.
3. Programs can be designed more easily because a small team deals with only a small part of the entire code.
4. Modular programming allows many programmers to collaborate on the same application.
5. The code is stored across multiple files.
6. Code is short, simple and easy to understand.
7. Errors can easily be identified, as they are localized to a subroutine or function.
8. The same code can be used in many applications.
9. The scoping of variables can easily be controlled

For example, let's consider the following modules of Docusign which is an eSignature API lets you eSign documents, request signatures, automate your forms and data, and much more. You can integrate the eSignature REST and SOAP APIs into any app, website, or embedded system that can make https requests.

URL Build Module:

```
public class UrlBuild {

 public String client_id = "5af20934-a3a7-45de-a002-e89416aabce8";

 public String secret_key = "411f4c83-8848-41d4-80f4-86a6ccefe311";

 public String ReturnAuth() throws MalformedURLException, URISyntaxException {

  String auth_url = "https://account-d.docusign.com/oauth/auth";

  String callback_url = "https://www.docusign.com";

  URIBuilder b = new URIBuilder(auth_url);

  b.addParameter("response_type", "code");

  b.addParameter("scope", "signature");

  b.addParameter("client_id", client_id);

  b.addParameter("state", "a39fh23hnf23");
  b.addParameter("redirect_uri", callback_url);
  //Building url
  URL url = b.build().toURL();
  System.setProperty("webdriver.chrome.driver", "//home/vinayr/Desktop//chromedriver");
  WebDriver driver = new ChromeDriver();
  driver.get(url.toString());
  driver.findElement(By.id("username")).sendKeys("krohithvarma1997@gmail.com");
  driver.findElement(By.xpath("//button[@type='submit']")).click();
  driver.findElement(By.id("password")).sendKeys("Mappingman123@");
  driver.findElement(By.xpath("//button[@type='submit']")).click();
  String ur=driver.getCurrentUrl();
  String auth_code = ur.split("\\=")[1];
  //getting authcode
  return auth_code;
 }
}
```

Request Module:

```
public class Requests {
 private String returnStatus() throws UnirestException, MalformedURLException,
URISyntaxException {
  UrlBuild authourizationcode = new UrlBuild();
  String auth_code = authourizationcode.ReturnAuth();
  //combine and encoding client and secret key into base 64
  String encodeBytes = Base64.getEncoder().encodeToString((authourizationcode.client_id +
":" + authourizationcode.secret_key).getBytes());););
```

```
  //Getting access token
  HttpResponse < JsonNode > response =
Unirest.post("https://account-d.docusign.com/oauth/token")
   .header("Authorization", "Basic " + encodeBytes)
   .header("Content-Type", "application/x-www-form-urlencoded")
   .body("grant_type=authorization_code&code=" + auth_code)
   .asJson();
  JSONObject tokenResponse = response.getBody().getObject();
  //System.out.println(tokenResponse);
  String access_token = tokenResponse.getString("access_token");
  return access_token;
 }

 public String[] returnUser() throws UnirestException, MalformedURLException,
URISyntaxException {
  Requests accesstoken = new Requests();
  String token = accesstoken.returnStatus();
  HttpResponse < JsonNode > user_response =
Unirest.get("https://account-d.docusign.com/oauth/userinfo")
   .header("Authorization", "Bearer " + token)
   .asJson();
  JSONObject user_info = user_response.getBody().getObject();
   //System.out.println(user_info);
  /*getting user info i.e
  baseuri
  account id*/
  JSONArray accounts = user_info.getJSONArray("accounts");
  String base_uri = accounts.getJSONObject(0).getString("base_uri");
  String account_id = accounts.getJSONObject(0).getString("account_id");
  String userdata[] = new String[3];
  userdata[0] = base_uri;
  userdata[1] = account_id;
  userdata[2] = token;
  return userdata;
 }
}
```

Sending the File Module:

```
public class FileSend{
private static Scanner input;


public static void main(String[] args) throws URISyntaxException, UnirestException,
IOException {
String auth_url="https://account-d.docusign.com/oauth/auth";
String client_id="5af20934-a3a7-45de-a002-e89416aabce8";
String callback_url="https://www.docusign.com";
String secret_key="411f4c83-8848-41d4-80f4-86a6ccefe311";
  URIBuilder b = new URIBuilder(auth_url);
b.addParameter("response_type", "code");
b.addParameter("scope", "signature");
```

```
b.addParameter("client_id", client_id);

b.addParameter("state", "a39fh23hnf23");

b.addParameter("redirect_uri", callback_url);


java.net.URL url = b.build().toURL();

        try{

            Desktop.getDesktop().browse(url.toURI());

        }

        catch(Exception E){

            System.err.println("Exp : "+E.getMessage());

        }


    input = new Scanner(System.in);

    //String authorization_response

    System.out.println("enter url");


     String authorization_response =input.nextLine();

    // System.out.println("authorization_response = " + authorization_response);

     String auth_code = authorization_response.split("\\=")[1];

     //getting authcode

     System.out.println(auth_code);


    // BASE64_COMBINATION_OF_INTEGRATOR_AND_SECRET_KEYS
```

```
     String encodeBytes =
Base64.getEncoder().encodeToString((client_id+":"+secret_key).getBytes());
  System.out.println("encoded value is " + encodeBytes);
  //String  grant_type="authorization_code";
  //Getting access token
  HttpResponse<JsonNode> response =
Unirest.post("https://account-d.docusign.com/oauth/token")
                        .header("Authorization", "Basic "+encodeBytes)
                        .header("Content-Type", "application/x-www-form-urlencoded")
                        .body("grant_type=authorization_code&code="+auth_code)
                        .asJson();
    JSONObject tokenResponse=response.getBody().getObject();
    //System.out.println(tokenResponse);
    String access_token=tokenResponse.getString("access_token");
    //System.out.println(access_token);


    //Getting userinfo i.e baseuri and accountid
    HttpResponse<JsonNode> user_response =
Unirest.get("https://account-d.docusign.com/oauth/userinfo")
                        .header("Authorization", "Bearer "+access_token)
                        .asJson();


      JSONObject user_info=user_response.getBody().getObject();
     // System.out.println(user_info);
```

```
   JSONArray accounts = user_info.getJSONArray("accounts");
    String base_uri=accounts.getJSONObject(0).getString("base_uri");
    String account_id=accounts.getJSONObject(0).getString("account_id");
    System.out.println(base_uri);
    System.out.println(account_id);


  System.out.println("enter recipient email: ");
         String mail=input.next();
         System.out.println("enter recipient name: ");
         String name=input.next();
         System.out.println("enter the full path of the file you want to send: ");
         String file_path=input.next();
   // path="/home/vinayr/Desktop/demo.pdf";
                  File f =new File(file_path);
                  String file_name = f.getName();
            String fname = file_name.substring(0, file_name.lastIndexOf('.'));
            System.out.println(fname);
            String ext = FilenameUtils.getExtension(file_path);
           System.out.println(ext);
           byte[] fileContent = FileUtils.readFileToByteArray(f);
                  String encodedString = Base64.getEncoder().encodeToString(fileContent);
 // System.out.println(encodedString);
                  HttpResponse<JsonNode> acc_response =
Unirest.post(base_uri+"/restapi/v2/accounts/"+account_id+"/envelopes")
```

```
                                .header("Authorization", "Bearer "+access_token)

                                .header("Content-Type", "application/json")

                                .body("{\n  \"documents\": [\n    {\n      \"documentBase64\":
\""+encodedString+"\",\n      \"documentId\": \"1\",\n      \"fileExtension\": \""+ext+"\",\n
\"name\": \""+fname+"\"\n    }\n  ],\n  \"emailSubject\": \"Please sign the sample file\",\n
\"recipients\": {\n    \"signers\": [\n      {\n        \"email\": \""+mail+"\",\n
\"name\":\""+name+"\",\n        \"recipientId\": \"1\",\n        \"routingOrder\": \"1\",\n
\"tabs\": {\n          \"signHereTabs\": [\n            {\n              \"xPosition\": \"450\",\n
\"yPosition\": \"650\",\n              \"documentId\": \"1\",\n              \"pageNumber\": \"1\"\n
}\n          ]\n        }\n      }\n    ]\n  },\n  \"status\": \"sent\"\n  }")

                                .asJson();


JSONObject env_info=acc_response.getBody().getObject()

 System.out.println(env_info);


 //System.out.println(acc_response);

}

}
```

Module for User Interaction:

```
public class UserInteraction {

 private static Scanner input;

 public static void main(String[] args) throws UnirestException, URISyntaxException,
IOException {

  Requests userinfo = new Requests();

  String data[] = userinfo.returnUser();

  input = new Scanner(System.in);
  System.out.println("enter recipient email: ");
  String mail = input.next();
  System.out.println("enter recipient name: ");
  String name = input.next();
  System.out.println("enter the full path of the file you want to send: ");
  String file_path = input.next();
  //Provide the complete path of file to be signed
  // eg:"/home/vinayr/Desktop/demo.pdf";
  File f = new File(file_path);
  String file_name = f.getName();
  String fname = file_name.substring(0, file_name.lastIndexOf('.'));
  System.out.println(fname);
  String ext = FilenameUtils.getExtension(file_path);
  System.out.println(ext);
  //converting the file into base64
  byte[] fileContent = FileUtils.readFileToByteArray(f);
  String encodedString = Base64.getEncoder().encodeToString(fileContent);
  // System.out.println(encodedString);
  HttpResponse < JsonNode > acc_response = Unirest.post(data[0] + "/restapi/v2/accounts/" +
data[1] + "/envelopes")
   .header("Authorization", "Bearer " + data[2])
   .header("Content-Type", "application/json")
   .body("{\n  \"documents\": [\n    {\n      \"documentBase64\": \"" + encodedString + "\",\n
\"documentId\": \"1\",\n      \"fileExtension\": \"" + ext + "\",\n      \"name\": \"" + fname +
"\"\n    }\n  ],\n  \"emailSubject\": \"Please sign the sample file\",\n  \"recipients\": {\n
\"signers\": [\n      {\n        \"email\": \"" + mail + "\",\n        \"name\":\"" + name + "\",\n
\"recipientId\": \"1\",\n        \"routingOrder\": \"1\",\n        \"tabs\": {\n          \"signHereTabs\":
[\n            {\n              \"xPosition\": \"450\",\n              \"yPosition\": \"650\",\n
\"documentId\": \"1\",\n              \"pageNumber\": \"1\"\n            }\n          ]\n        }\n      }\n
]\n  },\n  \"status\": \"sent\"\n  }")
   .asJson();
  JSONObject env_info = acc_response.getBody().getObject();
  System.out.println(env_info);
  //System.out.println(acc_response);
 }
}
```

5 SCHEDULE, TASKS, AND MILESTONES

TASKS IN REST API

An **API** is an application programming interface. It is a set of rules that allow programs to talk to each other. The developer creates the API on the server and allows the client to talk to it.

REST determines how the API looks like. It stands for "Representational State Transfer". It is a set of rules that developers follow when they create their API. One of these rules states that you should be able to get a piece of data (called a resource) when you link to a specific URL.

Each URL is called a **request** while the data sent back to you is called a **response**.

The Anatomy Of A Request

It's important to know that a request is made up of four things:

1. The endpoint
2. The method
3. The headers

4. The data (or body)

The endpoint (or route) is the url you request for. It follows this structure:

The **path** determines the resource you're requesting. Think of it like an automatic answering machine that asks you to press 1 for a service, press 2 for another service, 3 for yet another service and so on.

6. SUMMARY:

Nearly all data breaches start with compromised passwords. Whether it be through sophisticated phishing, brute force attacks, social engineering, or any other kind of credential harvesting, the password is the first, and sometimes only, line of defense against cyberattacks.

When passwords are cracked, the floodgates are opened, and all types of malicious programs and activities can get into an organization's systems. The password, then, is like sticking your bubble gum over the hole in a dam in order to prevent a torrent of water from rushing through. Obviously, as a method of protecting personal accounts and company databases, username-password combos are far from ideal.

So will we see the password go extinct in our lifetimes? It's hard to say, but there is definitely a growing trend to move away from the internet's oldest security strategy, and we can reasonably speculate that fewer and fewer online platforms will require one in the future.

7.References:

[1] Matt Raible, “Secure a Spring Microservices Architecture with Spring Security”
Ref:https://developer.okta.com/blog/2018/02/13/secure-spring-microservices-with-oauth

[2] Sovan Mishra ,” Guide to Creating a Project With Maven ”
Ref:https://dzone.com/articles/understanding-maven

[3] Hussein Terek“Developing a REST API using Spring Boot Java-based framework”
Ref:https://dzone.com/articles/introducing-spring-boot