Spring Boot and JWT Authentication

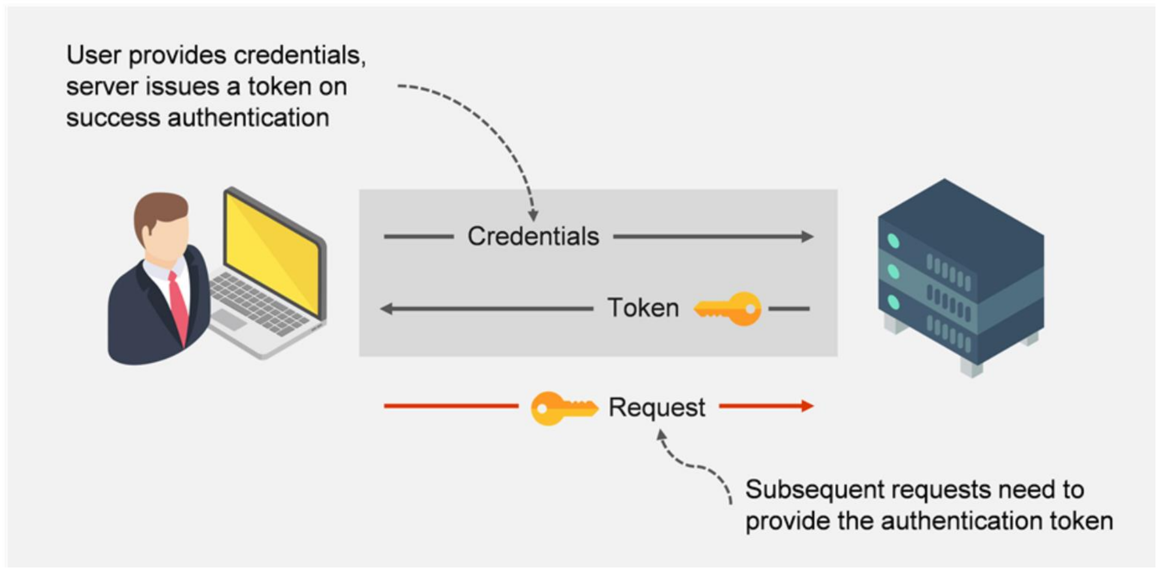
What is a JSON Web Token?

- It is a standard for token-based authentication.
- It works across different programming languages.
- It can be passed around easily.
- JSON Web Tokens is an open, industry-standard RFC 7519 method used for representing claims securely between two parties.

Why Should We Use a JSON Web Token?

- Ease – Ease of client-side processing of the JSON Web Token on multiple platforms.
- Compact - It can be sent through a URL, POST parameter, or inside the HTTP header because of its size. Its transmission is also fast due to its size.
- Security – Securely transmitting information between parties using public/private key pairs.
- Self-Contained – The payload contains all the required information about the user, to avoid querying the database more than once.

Token-based Authentication

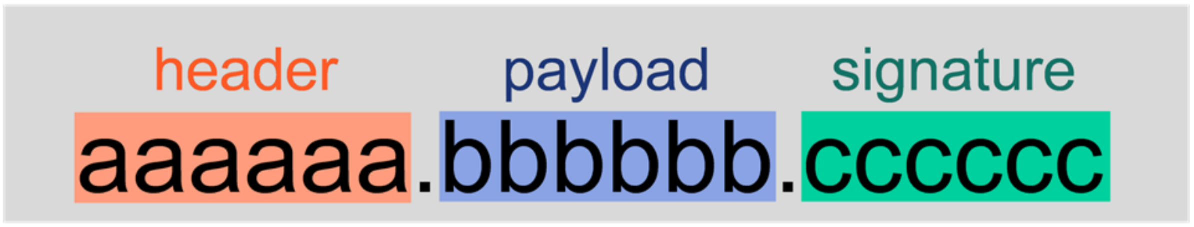

User provides credentials, server issues a token on success authentication
Credentials
Token
Request
Subsequent requests need to provide the authentication token

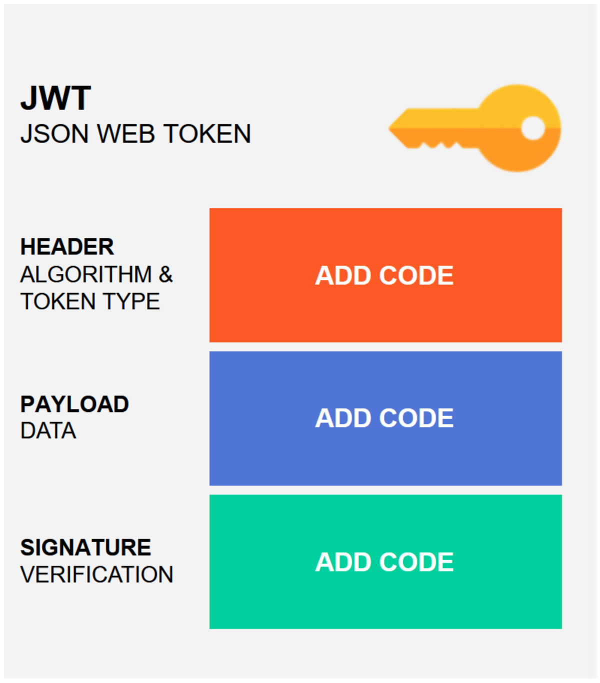
How Does the JWT Work?

- The user first signs into the authentication server using the authentication server's login system (e.g., username and password, Facebook login, Google login, Twitter etc.).
- The authentication server then creates the JWT and sends it to the user.
- When the user makes API calls to the application, the user passes the JWT along with the API call.
- In this setup, the application server would be configured to verify that the incoming JWT are created by the authentication server.
- When the user makes API calls with the attached JWT, the application can use the JWT to verify whether the API call is coming from an authenticated user.

How Does JWT Look Like?

- A JWT is composed of three strings separated by a period/dot.
- The first part is the header, the second is the payload, and the third is the signature.

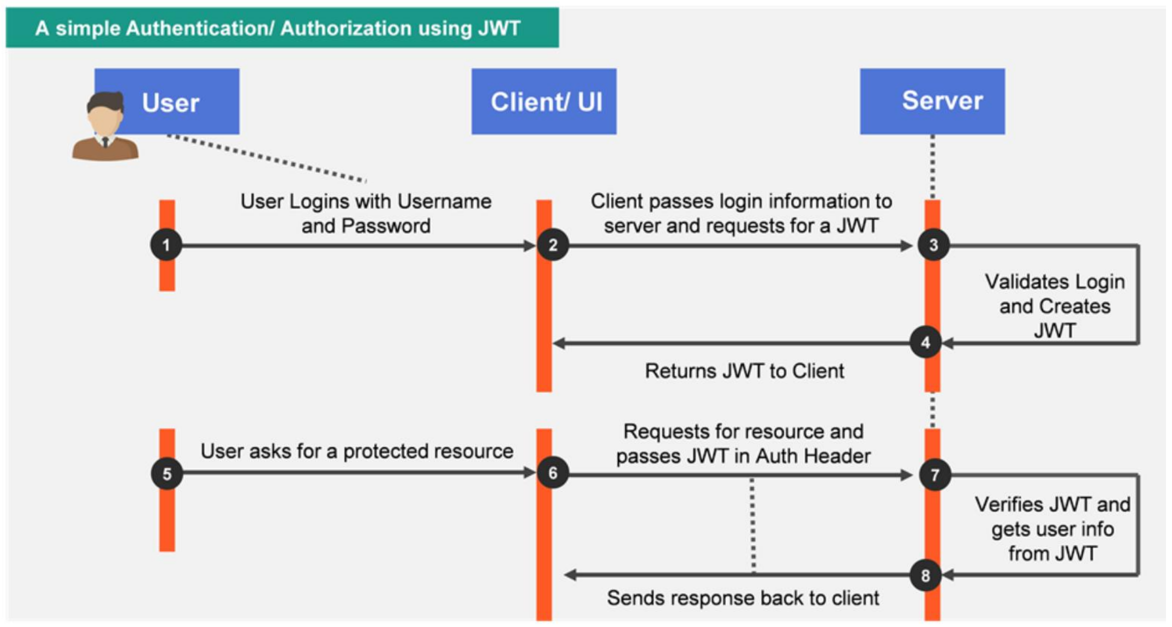




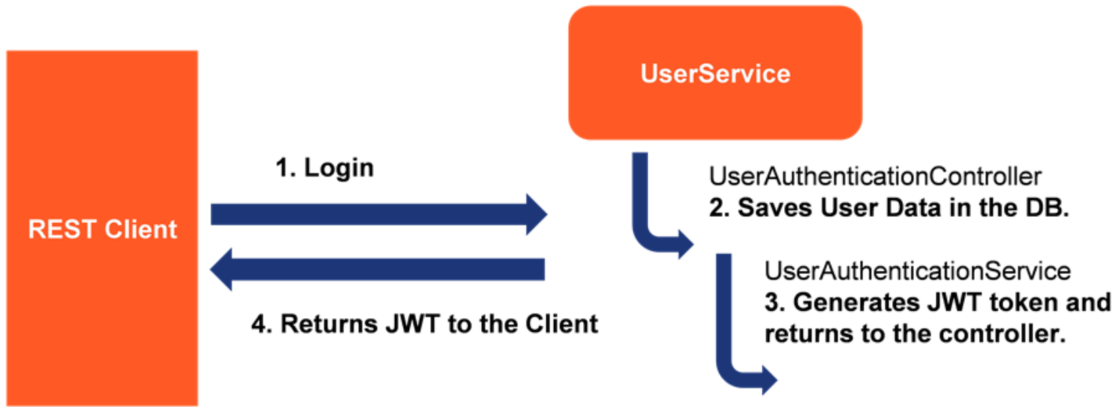
How Does the JWT Look Like?

- Header consists of two parts: the type of token which is JWT, and the signing algorithm being used (HMAC SHA256 in this case).
- Payload - The payload will carry the bulk of JWT, also called the JWT Claims. Claims are statements about an entity(typically the user) and additional data. There are three types of claims: registered, public, and private claims. The payload will carry the bulk of our JWT, also called the JWT Claims.
- Signature - The third and final part of JWT is the signature. This signature is made up of a hash of the following components:
  - the header
  - the payload
  - Secret

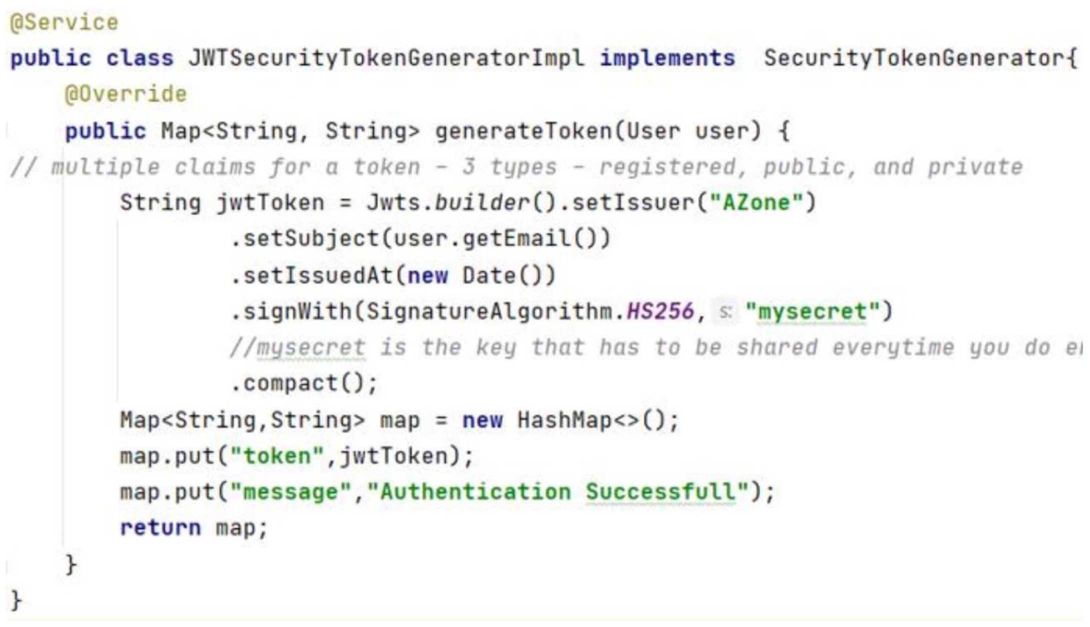
Data Flow of an Application Using JWT


A simple Authentication/ Authorization using JWT
User
Client/ UI
Server
User Logins with Username and Password
Client passes login information to server and requests for a JWT
Validates Login and Creates JWT
Returns JWT to Client
User asks for a protected resource
Requests for resource and passes JWT in Auth Header
Verifies JWT and gets user info from JWT
Sends response back to client
1
2
3
4
5
6
7
8

Flow Diagram – To Generate the Token

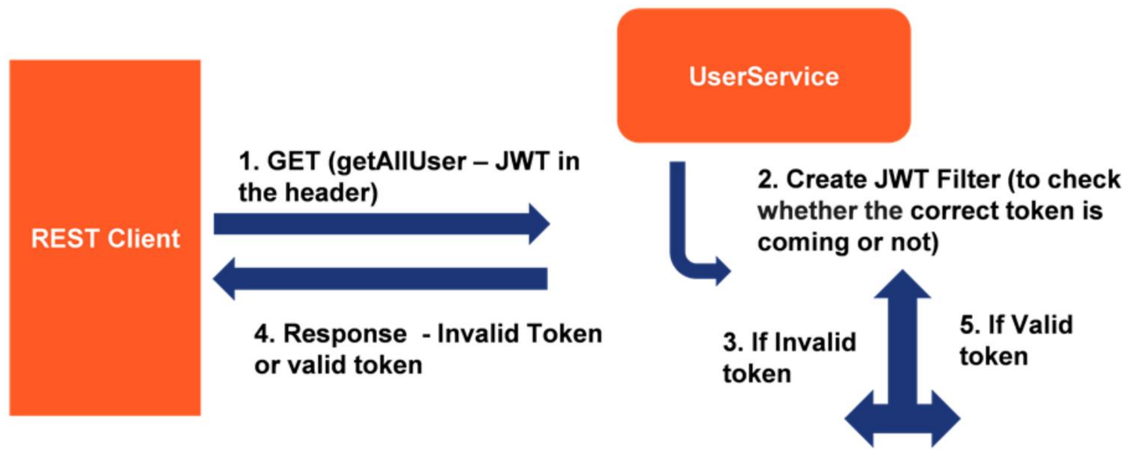

UserService
REST Client
1. Login
4. Returns JWT to the Client
UserAuthenticationController
2. Saves User Data in the DB.
UserAuthenticationService
3. Generates JWT token and returns to the controller.

Generate Token



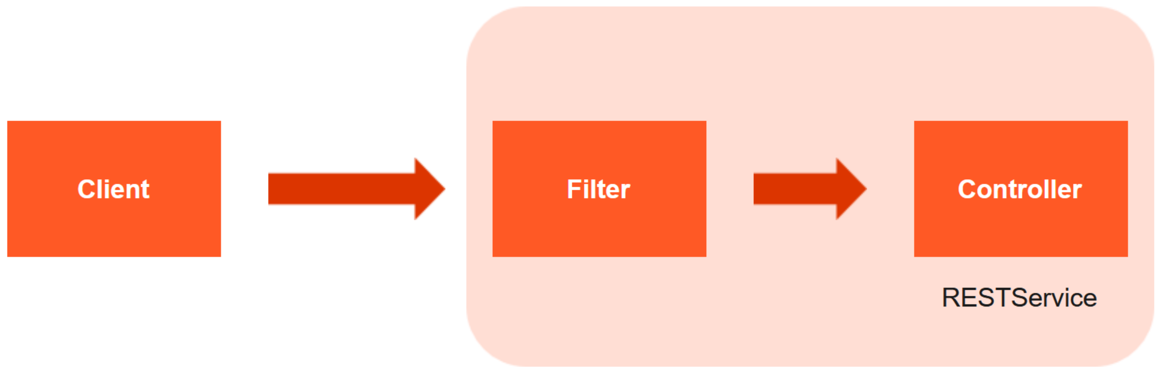
```
@Service
public class JWTSecurityTokenGeneratorImpl implements  SecurityTokenGenerator{
    @Override
    public Map<String, String> generateToken(User user) {
// multiple claims for a token - 3 types - registered, public, and private
        String jwtToken = Jwts.builder().setIssuer("AZone")
                .setSubject(user.getEmail())
                .setIssuedAt(new Date())
                .signWith(SignatureAlgorithm.HS256, s: "mysecret")
                //mysecret is the key that has to be shared everytime you do e
                .compact();
        Map<String,String> map = new HashMap<>();
        map.put("token",jwtToken);
        map.put("message","Authentication Successfull");
        return map;
    }
}
```

Flow Diagram – For the Filter

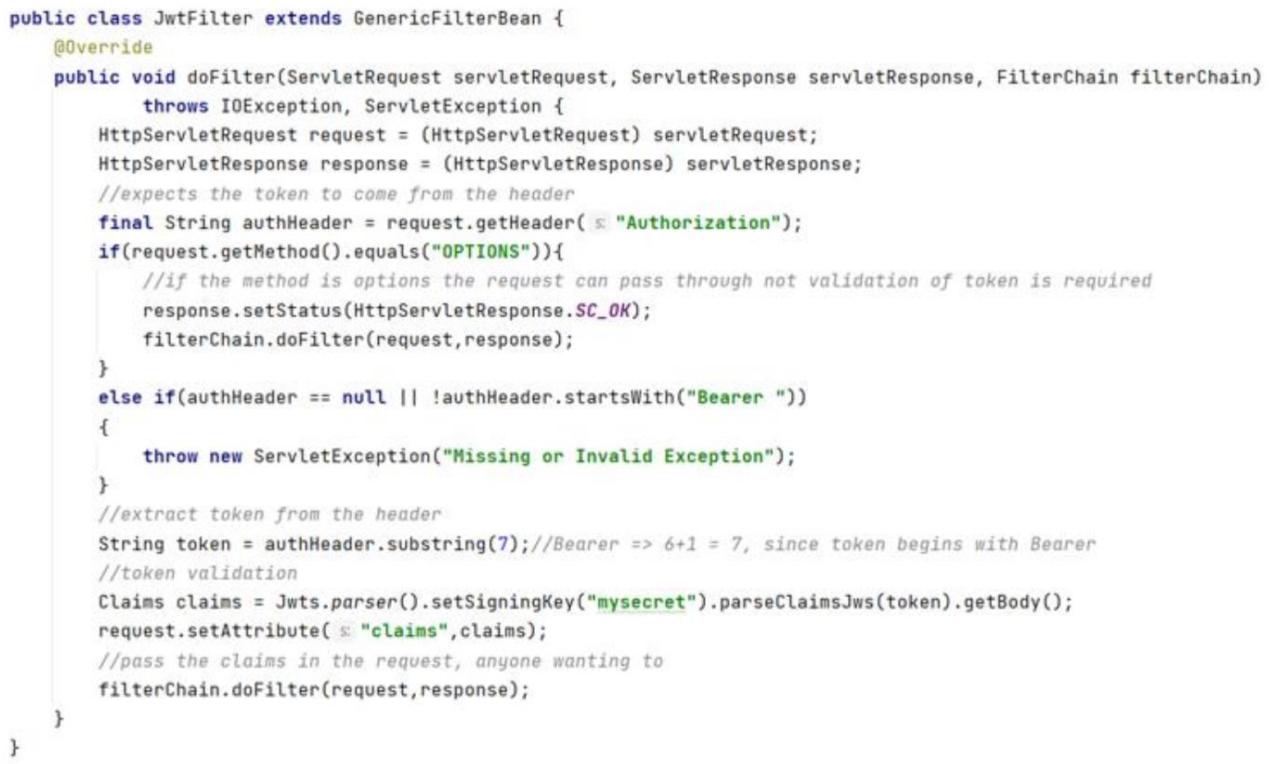

UserService
REST Client
1. GET (getAllUser – JWT in the header)
2. Create JWT Filter (to check whether the correct token is coming or not)
4. Response - Invalid Token or valid token
3. If Invalid token
5. If Valid token

Verification of JWT Token by Filter

- **A filter is an object that does the pre-processing and post-processing of a request.**



Filter Code



```java
public class JwtFilter extends GenericFilterBean {
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain)
            throws IOException, ServletException {
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        HttpServletResponse response = (HttpServletResponse) servletResponse;
        //expects the token to come from the header
        final String authHeader = request.getHeader( s: "Authorization");
        if(request.getMethod().equals("OPTIONS")){
            //if the method is options the request can pass through not validation of token is required
            response.setStatus(HttpServletResponse.SC_OK);
            filterChain.doFilter(request,response);
        }
        else if(authHeader == null || !authHeader.startsWith("Bearer "))
        {
            throw new ServletException("Missing or Invalid Exception");
        }
        //extract token from the header
        String token = authHeader.substring(7);//Bearer => 6+1 = 7, since token begins with Bearer
        //token validation
        Claims claims = Jwts.parser().setSigningKey("mysecret").parseClaimsJws(token).getBody();
        request.setAttribute( s: "claims",claims);
        //pass the claims in the request, anyone wanting to
        filterChain.doFilter(request,response);
    }
}
```

Example:

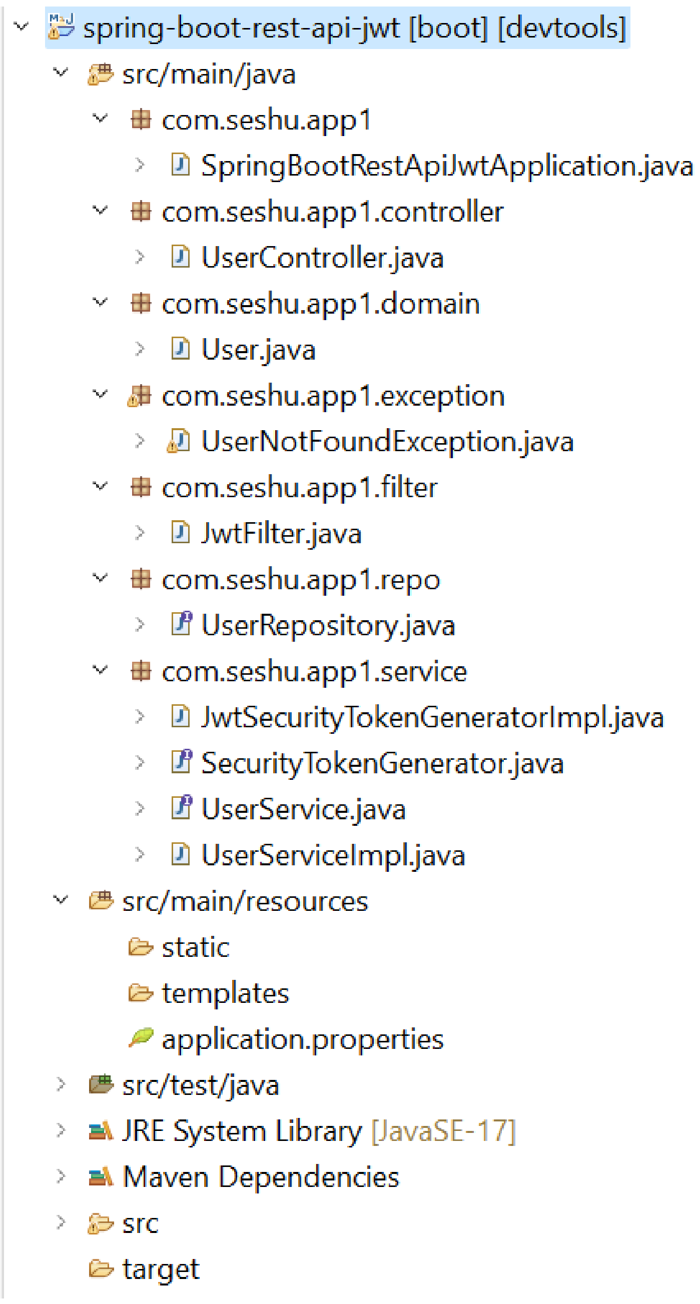
Generate spring boot project with fallowing dependencies;

Spring devtools

Spring data jpa

Mysql Driver

Spring Web



```
spring-boot-rest-api-jwt [boot] [devtools]
  src/main/java
    com.seshu.app1
      SpringBootRestApiJwtApplication.java
    com.seshu.app1.controller
      UserController.java
    com.seshu.app1.domain
      User.java
    com.seshu.app1.exception
      UserNotFoundException.java
    com.seshu.app1.filter
      JwtFilter.java
    com.seshu.app1.repo
      UserRepository.java
    com.seshu.app1.service
      JwtSecurityTokenGeneratorImpl.java
      SecurityTokenGenerator.java
      UserService.java
      UserServiceImpl.java
  src/main/resources
    static
    templates
    application.properties
  src/test/java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
  target
```

pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
     <modelVersion>4.0.0</modelVersion>
     <parent>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-parent</artifactId>
          <version>2.7.1</version>
          <relativePath /> <!-- lookup parent from repository -->
     </parent>
     <groupId>com.seshu</groupId>
     <artifactId>spring-boot-rest-api-jwt</artifactId>
     <version>0.0.1-SNAPSHOT</version>
     <name>spring-boot-rest-api-jwt</name>
     <description>Demo project for Spring Boot</description>
     <properties>
          <java.version>17</java.version>
     </properties>
     <dependencies>
          <dependency>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-starter-web</artifactId>
          </dependency>

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

          <dependency>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-devtools</artifactId>
               <scope>runtime</scope>
               <optional>true</optional>
          </dependency>
          <dependency>
               <groupId>mysql</groupId>
               <artifactId>mysql-connector-java</artifactId>
               <scope>runtime</scope>
          </dependency>
          <dependency>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-starter-test</artifactId>
               <scope>test</scope>
          </dependency>
          <dependency>
               <groupId>io.jsonwebtoken</groupId>
               <artifactId>jjwt</artifactId>
               <version>0.9.1</version>
          </dependency>

     </dependencies>

     <build>
          <plugins>
               <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
               </plugin>
          </plugins>
     </build>
</project>
```

application.properties

```
server.port=8084
spring.datasource.url=jdbc:mysql://localhost:3306/adidb
spring.datasource.username=root
spring.datasource.password=seshu

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

server.error.include-message=always
```

User.java

```
package com.seshu.app1.domain;

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

@Entity
@Table(name = "user_details")
public class User {
	@Id
	@GeneratedValue
	private int userId;
	private String username;
	private String password;
	private String address;

	public User() {
	}

	public User(int userId, String username, String password, String 
address) {
		this.userId = userId;
		this.username = username;
		this.password = password;
		this.address = address;
	}

	public int getUserId() {
		return userId;
	}

	public void setUserId(int userId) {
		this.userId = userId;
	}

	public String getUsername() {
		return username;
	}

	public void setUsername(String username) {
		this.username = username;
	}

	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}

	public String getAddress() {
		return address;
	}

	public void setAddress(String address) {
		this.address = address;
	}

	@Override
	public String toString() {
		return "User{" + "userId=" + userId + ", username='" + username
+ '\'' + ", password='" + password + '\''
				+ ", address='" + address + '\'' + '}';
	}
}
```

UserRepository.java

```
package com.seshu.app1.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import com.seshu.app1.domain.User;

public interface UserRepository extends JpaRepository<User,Integer> {
  public User findByUsernameAndPassword(String username , String
password);
}
```

UserNotFoundException.java

```
package com.seshu.app1.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(code = HttpStatus.NOT_FOUND , reason = "User Not Found")
public class UserNotFoundException extends Exception {
}
```

UserService.java

```
package com.seshu.app1.service;

import java.util.List;

import com.seshu.app1.domain.User;
import com.seshu.app1.exception.UserNotFoundException;


public interface UserService {
  public User saveUser(User user);
  public User findByUsernameAndPassword(String username , String
password) throws UserNotFoundException;
  List<User> getAllUsers();
}
```

UserServiceImpl.java

```
package com.seshu.app1.service;

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

import com.seshu.app1.domain.User;
import com.seshu.app1.exception.UserNotFoundException;
import com.seshu.app1.repo.UserRepository;

import java.util.List;

@Service
public class UserServiceImpl implements UserService {
	private UserRepository userRepository;

	@Autowired
	public UserServiceImpl(UserRepository userRepository) {
		this.userRepository = userRepository;

	}

	@Override
	public User saveUser(User user) {
		return userRepository.save(user);
	}

	@Override
	public User findByUsernameAndPassword(String username, String
password) throws UserNotFoundException {
		User user = userRepository.findByUsernameAndPassword(username,
password);
		if (user == null) {
			throw new UserNotFoundException();
		}
		return user;

	}

	@Override
	public List<User> getAllUsers() {
		return userRepository.findAll();
	}
}
```

SecurityTokenGenerator.java

```
package com.seshu.app1.service;


import java.util.Map;

import com.seshu.app1.domain.User;

public interface SecurityTokenGenerator {

  Map<String,String> generateToken(User user);
}
```

JwtSecurityTokenGeneratorImpl.java

```
package com.seshu.app1.service;

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.stereotype.Service;

import com.seshu.app1.domain.User;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;

@Service
public class JwtSecurityTokenGeneratorImpl implements
SecurityTokenGenerator {

     @Override
     public Map<String, String> generateToken(User user) {

          String jwtToken = null;
          jwtToken =
Jwts.builder().setSubject(user.getUsername()).setIssuedAt(new Date())
                    .signWith(SignatureAlgorithm.HS256,
"secretkey").compact();

          Map<String, String> map = new HashMap<>();
          map.put("token", jwtToken);
          map.put("message", "User Successfully logged in");
          return map;
     }
}
```

JwtFilter.java

```
package com.seshu.app1.filter;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import org.springframework.web.filter.GenericFilterBean;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class JwtFilter extends GenericFilterBean {
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException,
ServletException {

        final HttpServletRequest request = (HttpServletRequest)
servletRequest;
        final HttpServletResponse response = (HttpServletResponse)
servletResponse;
        final String authHeader = request.getHeader("authorization");
        if ("OPTIONS".equals(request.getMethod())) {
            response.setStatus(HttpServletResponse.SC_OK);
            filterChain.doFilter(request, response);
        } else {
            if (authHeader == null || !authHeader.startsWith("Bearer "))
{
                throw new ServletException("Missing or invalid
Authorization header");
            }
            final String token = authHeader.substring(7);
            final Claims claims =
Jwts.parser().setSigningKey("secretkey").parseClaimsJws(token).getBody();
            request.setAttribute("claims", claims);
            filterChain.doFilter(request, response);
        }
    }
}
```

UserController.java

```
package com.seshu.app1.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import com.seshu.app1.domain.User;
import com.seshu.app1.exception.UserNotFoundException;
import com.seshu.app1.service.SecurityTokenGenerator;
import com.seshu.app1.service.UserService;

import javax.servlet.http.HttpServletRequest;
import java.util.List;
import java.util.Map;

@RestController
public class UserController {

     private ResponseEntity<?> responseEntity;
     @Autowired
     private UserService userService;
     @Autowired
     private SecurityTokenGenerator securityTokenGenerator;

     // Should only give username and password
     @PostMapping("/login")
     public ResponseEntity<?> loginUser(@RequestBody User user) throws
UserNotFoundException {

          Map<String, String> map = null;
          try {
               User userObj =
userService.findByUsernameAndPassword(user.getUsername(),
user.getPassword());
               if (userObj.getUsername().equals(user.getUsername())) {
                    map = securityTokenGenerator.generateToken(user);
               }
               responseEntity = new ResponseEntity<>(map, HttpStatus.OK);
          } catch (UserNotFoundException e) {
               throw new UserNotFoundException();
          } catch (Exception e) {
               responseEntity = new ResponseEntity<>("Try after
sometime!!!", HttpStatus.INTERNAL_SERVER_ERROR);
          }
          return responseEntity;
     }
```

```
	// first step - register the user
	@PostMapping("/register")
	public ResponseEntity<?> saveUser(@RequestBody User user) {

		userService.saveUser(user);
		return responseEntity = new ResponseEntity<>("User Created",
HttpStatus.CREATED);
	}

	@GetMapping("/api/v1/userservice/users")
	public ResponseEntity<?> getAllUsers(HttpServletRequest request) {

		List<User> list = userService.getAllUsers();
		responseEntity = new ResponseEntity<>(list, HttpStatus.OK);
		return responseEntity;
	}
}
```

SpringBootRestApiJwtApplication.java

```
package com.seshu.app1;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootRestApiJwtApplication {

	public static void main(String[] args) {
		SpringApplication.run(SpringBootRestApiJwtApplication.class,
args);
	}
}
```

Run Starter class

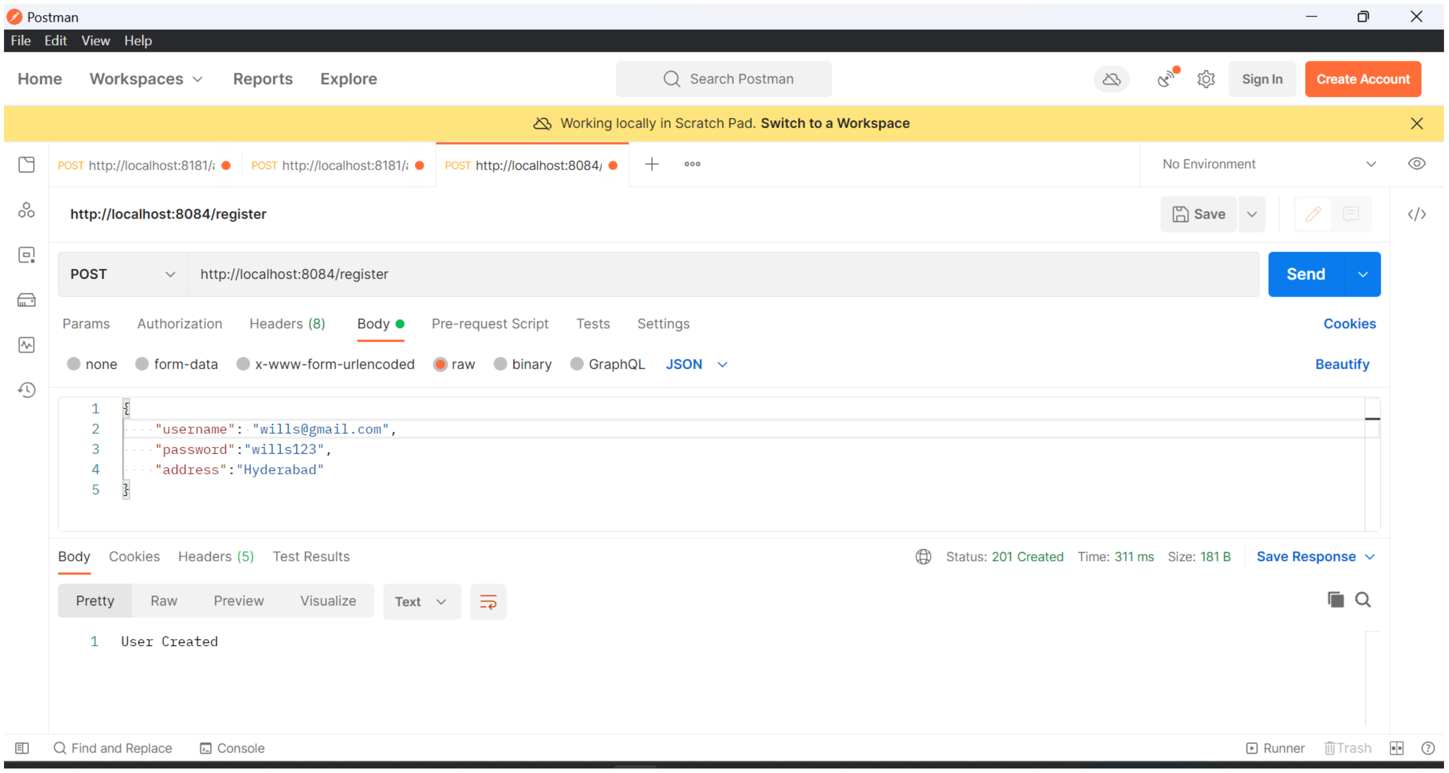
Test application using Postman client tool

Adding new user

http://localhost:8084/register

Request Body

```
{
    "username": "wills@gmail.com",
    "password":"wills123",
    "address":"Hyderabad"
}
```


Postman
File Edit View Help
Home Workspaces Reports Explore
Search Postman
Sign In
Create Account
Working locally in Scratch Pad. Switch to a Workspace
POST http://localhost:8181/
POST http://localhost:8181/
POST http://localhost:8084/
No Environment
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (5) Test Results
Save Response
Pretty Raw Preview Visualize Text
Find and Replace Console Runner Trash

http://localhost:8084/register

POST http://localhost:8084/register

```
{
····"username":·"wills@gmail.com",
····"password":"wills123",
····"address":"Hyderabad"
}
```

Status: 201 Created Time: 311 ms Size: 181 B

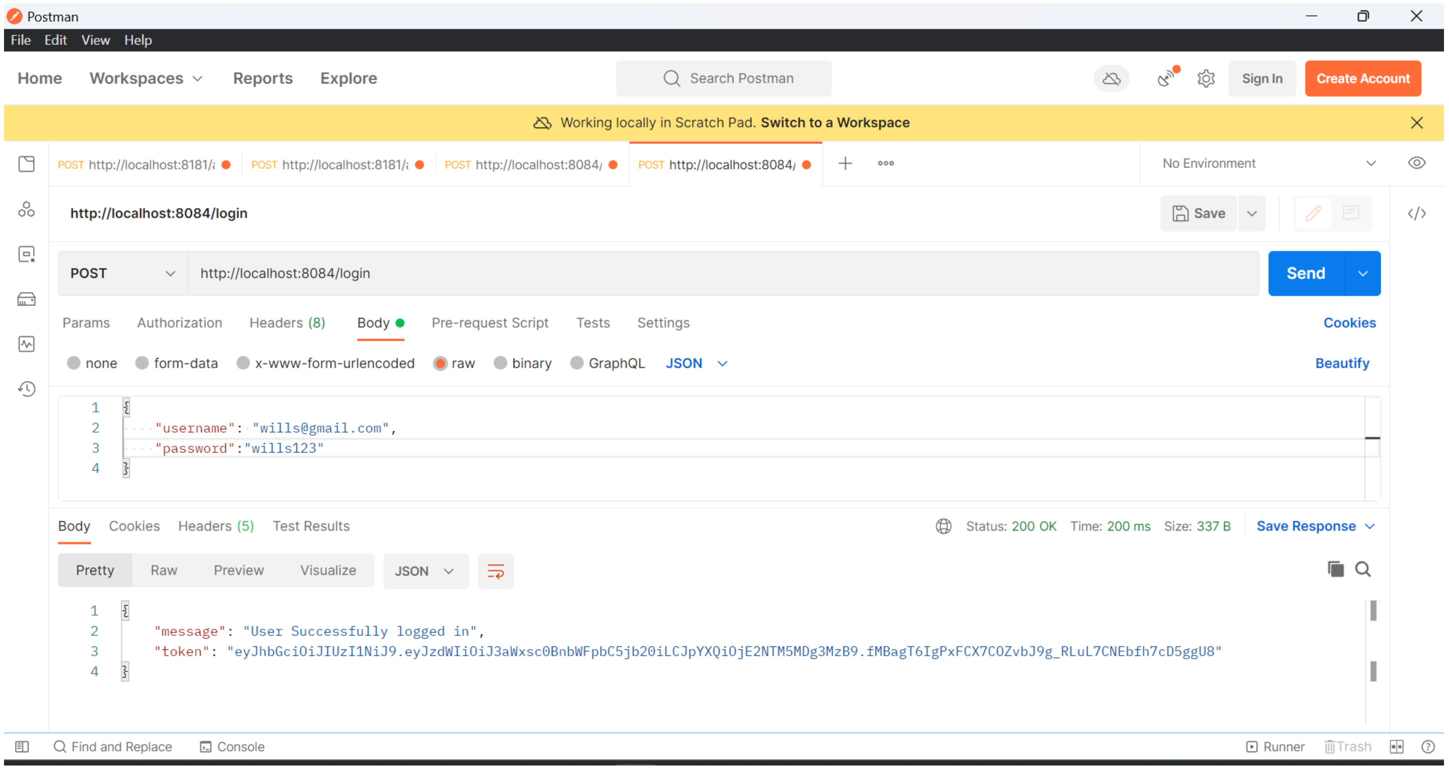
```
User Created
```

Sending login request and generate token

http://localhost:8084/login

Request Body

```
{
    "username": "wills@gmail.com",
    "password":"wills123"
}
```



POST http://localhost:8084/login

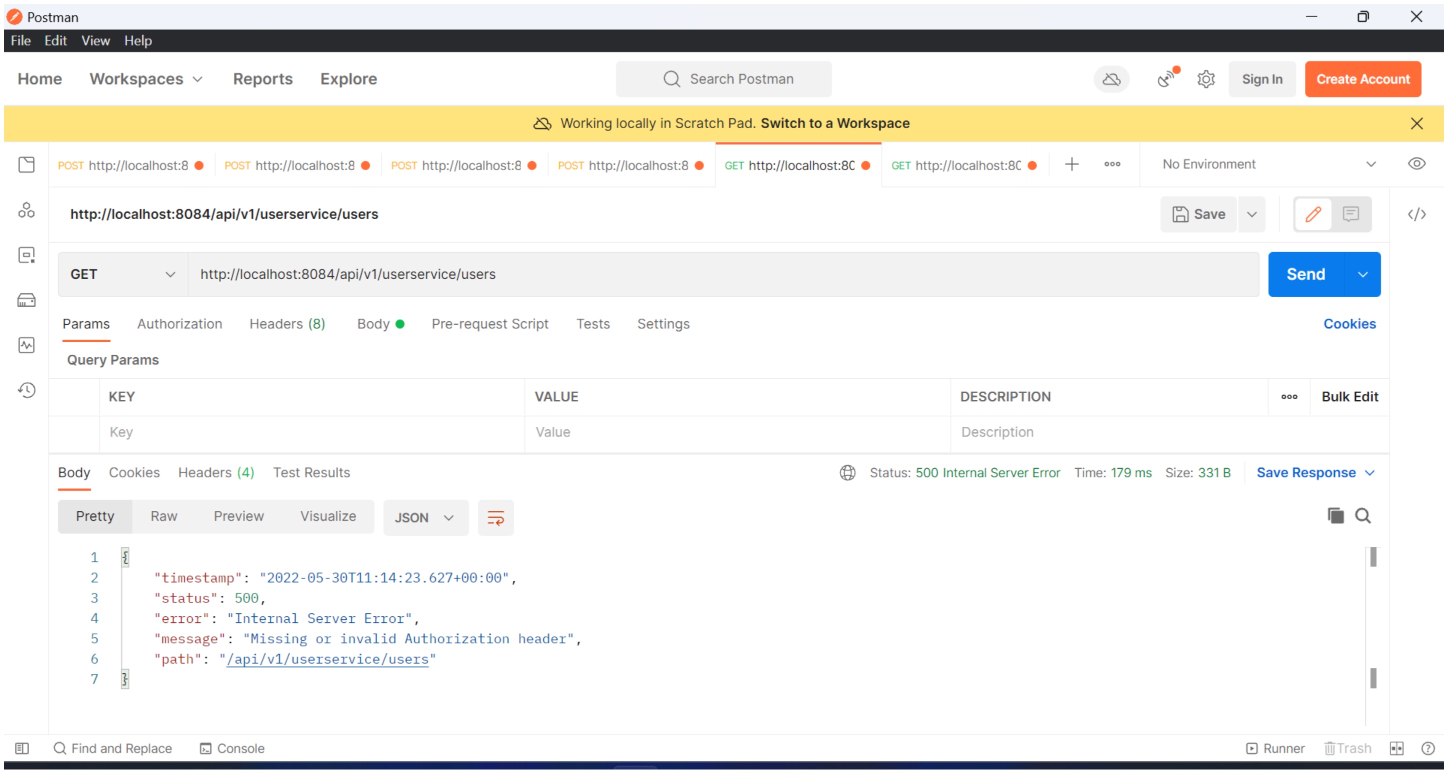
```
{
    "username": "wills@gmail.com",
    "password":"wills123"
}
```

Status: 200 OK Time: 200 ms Size: 337 B

```
{
    "message": "User Successfully logged in",
    "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ3aWxsc0BnbWFpbC5jb20iLCJpYXQiOjE2NTM5MDg3MzB9.fMBagT6IgPxFCX7COZvbJ9g_RLuL7CNEbfh7cD5ggU8"
}
```

Sending request to access users without passing token

http://localhost:8084/api/v1/userservice/users


Postman
File Edit View Help
Home Workspaces Reports Explore
Search Postman
Sign In
Create Account
Working locally in Scratch Pad. Switch to a Workspace
No Environment
http://localhost:8084/api/v1/userservice/users
Save
GET
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
Query Params
Body Cookies Headers (4) Test Results
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console Runner Trash

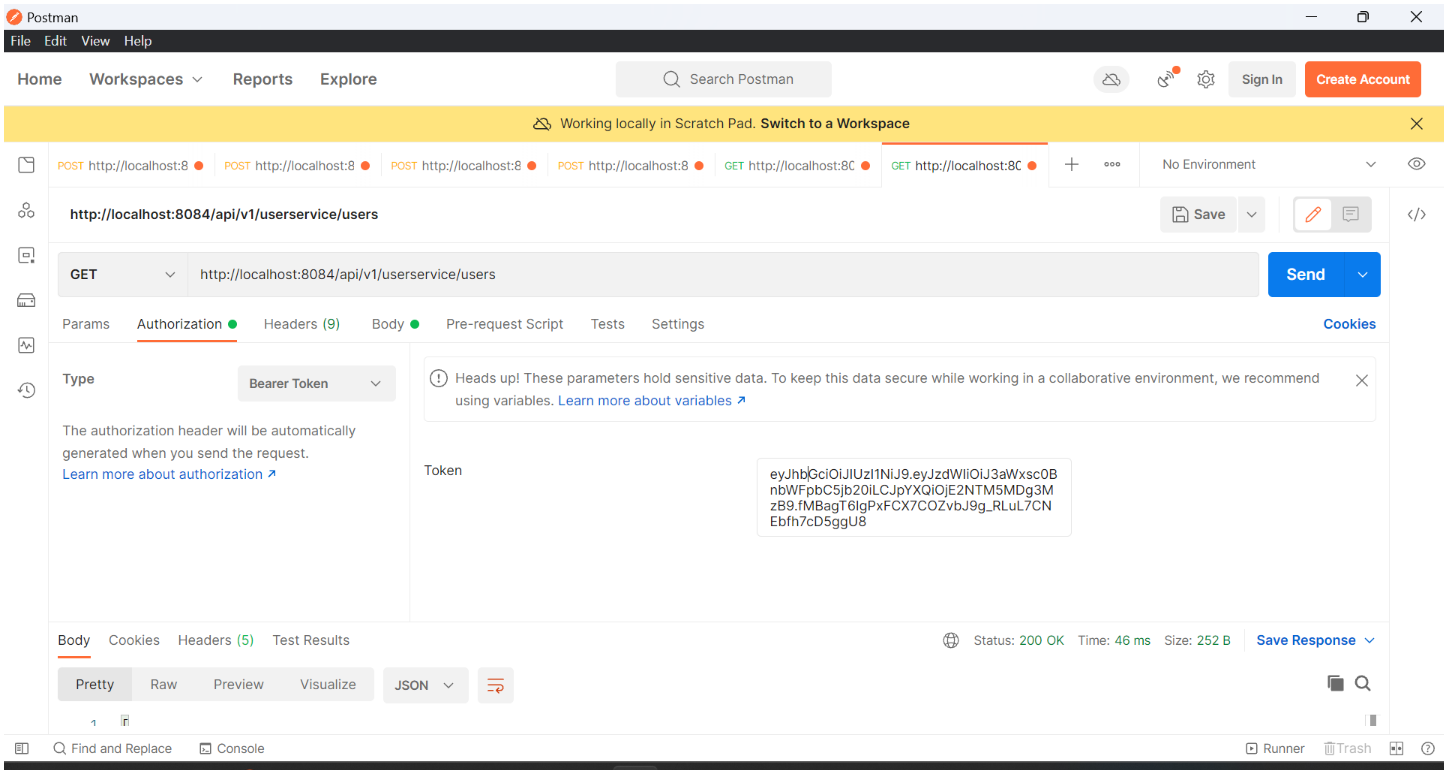
GET http://localhost:8084/api/v1/userservice/users

KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 500 Internal Server Error Time: 179 ms Size: 331 B

```
{
    "timestamp": "2022-05-30T11:14:23.627+00:00",
    "status": 500,
    "error": "Internal Server Error",
    "message": "Missing or invalid Authorization header",
    "path": "/api/v1/userservice/users"
}
```

http://localhost:8084/api/v1/userservice/users



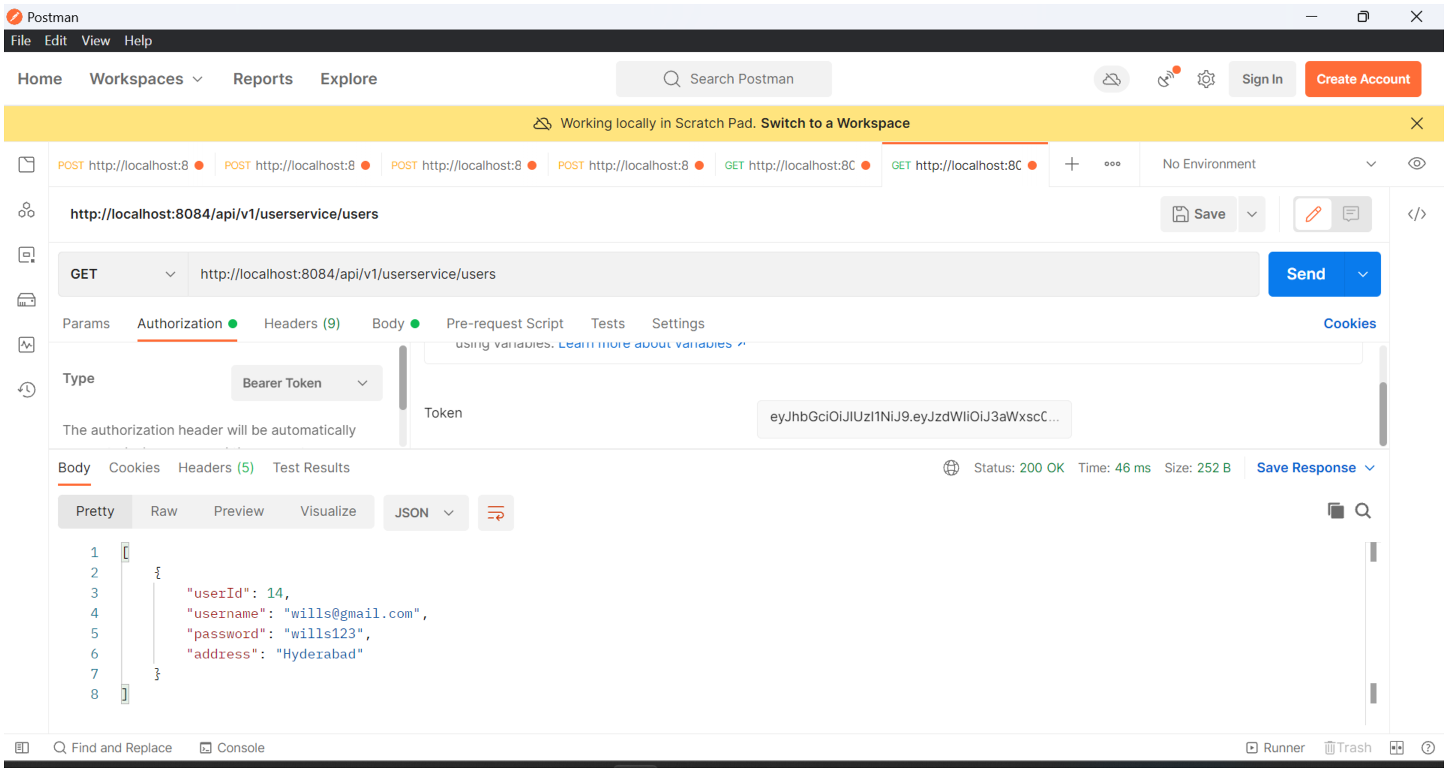
GET http://localhost:8084/api/v1/userservice/users

Authorization — Type: Bearer Token

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables ↗

Token: eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ3aWxsc0BnbWFpbC5jb20iLCJpYXQiOjE2NTM5MDg3MzB9.fMBagT6lgPxFCX7COZvbJ9g_RLuL7CNEbfh7cD5ggU8

Status: 200 OK Time: 46 ms Size: 252 B



GET http://localhost:8084/api/v1/userservice/users

Authorization — Type: Bearer Token

Token: eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ3aWxsc...

Status: 200 OK Time: 46 ms Size: 252 B

```
[
    {
        "userId": 14,
        "username": "wills@gmail.com",
        "password": "wills123",
        "address": "Hyderabad"
    }
]
```