Think and Tell

- Do you think a login page is required for each application that we create in this digital era?
- How can you make our application safe from unwanted users?
- How can we make a digital transaction safe?

Apps in Play Store

- Do we always have to log in to the play store to view the apps?
- Do you require authentication when downloading an app or a game from the play store?

Authenticate a Backend Application by Using JASON Web Token (JWT)

Menu

00:03/00:00:38 03/ 38

Slide Note

Learning Objectives

- Explain authentication in Spring Boot
- Explore JWT
- Explain the working of JSON Web Token (JWT)
- Implement JWT for authentication

Menu

00:04 00:00:38 04/ 38

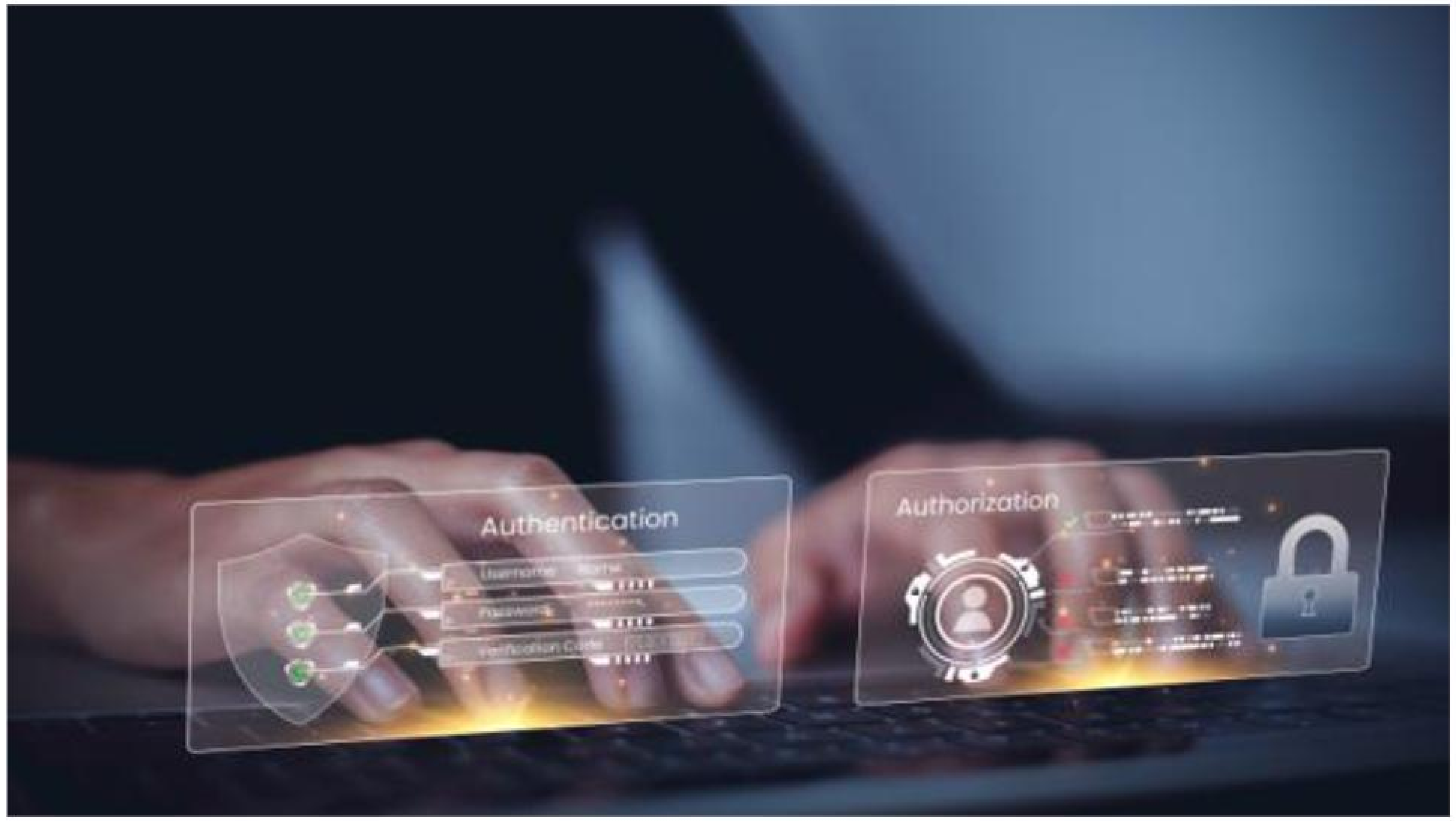
Slide Note

Explain Authentication in Spring Boot

Menu

Slide Note

Authentication and Authorization



- Authentication is a process that examines the user at the login level, where the user is verified.

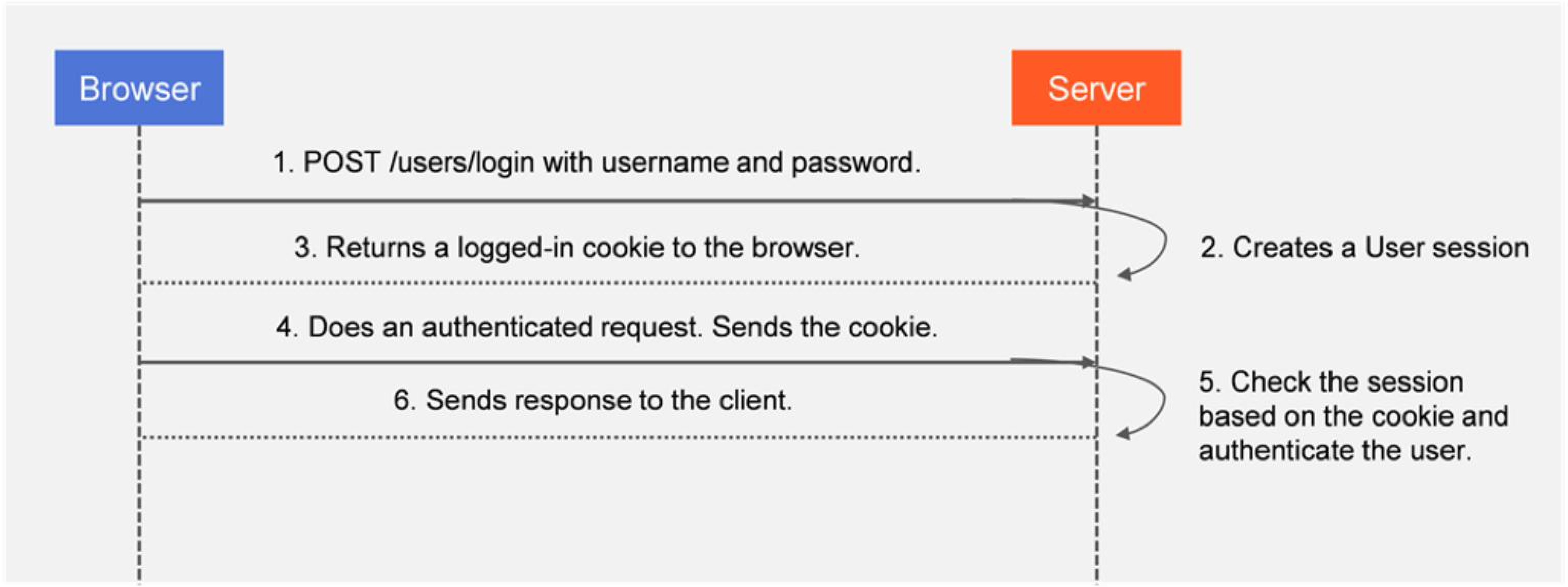
- Authorization is a process where each verified user of a system is given an individual space and resource along with the roles that they already possess in a system.

Slide Note

The predominant authentication process was initially server-based. As we all know, HTTP Protocol is stateless. This means that if we authenticate a user with a username and password, then on the next request, our application won't know who we are. We would have to authenticate again. So, there was a need to ensure that after a user had logged in.

A user's credentials are sent as a POST request to the server. The server authenticates the user. The server responds with a cookie set on the user's browser if the credentials are valid. It includes a SESSION ID to identify the user. The user sessions are stored in memory via files or in the database on the server.

A Look at Session-Based Authentication


Browser
Server
1. POST /users/login with username and password.
2. Creates a User session
3. Returns a logged-in cookie to the browser.
4. Does an authenticated request. Sends the cookie.
5. Check the session based on the cookie and authenticate the user.
6. Sends response to the client.

Menu

Slide Note

Explore JSON Web Token (JWT)

Menu

00:08:00:00:38 08/ 38

Slide Note

What Is a JSON Web Token?

- The JSON Web Token, or JWT, as it is more commonly called, is an open internet standard for securely and compactly transmitting trusted information between the client and server.
- JWT can be used to authenticate or verify an application user.
- It is a standard for token-based authentications.
- JWT works across different programming languages.
- It can be passed around easily between the client and server.
- The tokens contain claims encoded as JSON objects and digitally signed using a private secret or a public/private key pair.
- They are self-contained and verifiable as they are digitally signed.

Slide Note

Why Should You 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 client and server using public/private key pairs.
- Self-Contained – The payload contains all the required information about the user so it can avoid querying the database more than once.

Slide Note

The Uses of JSON Web Tokens

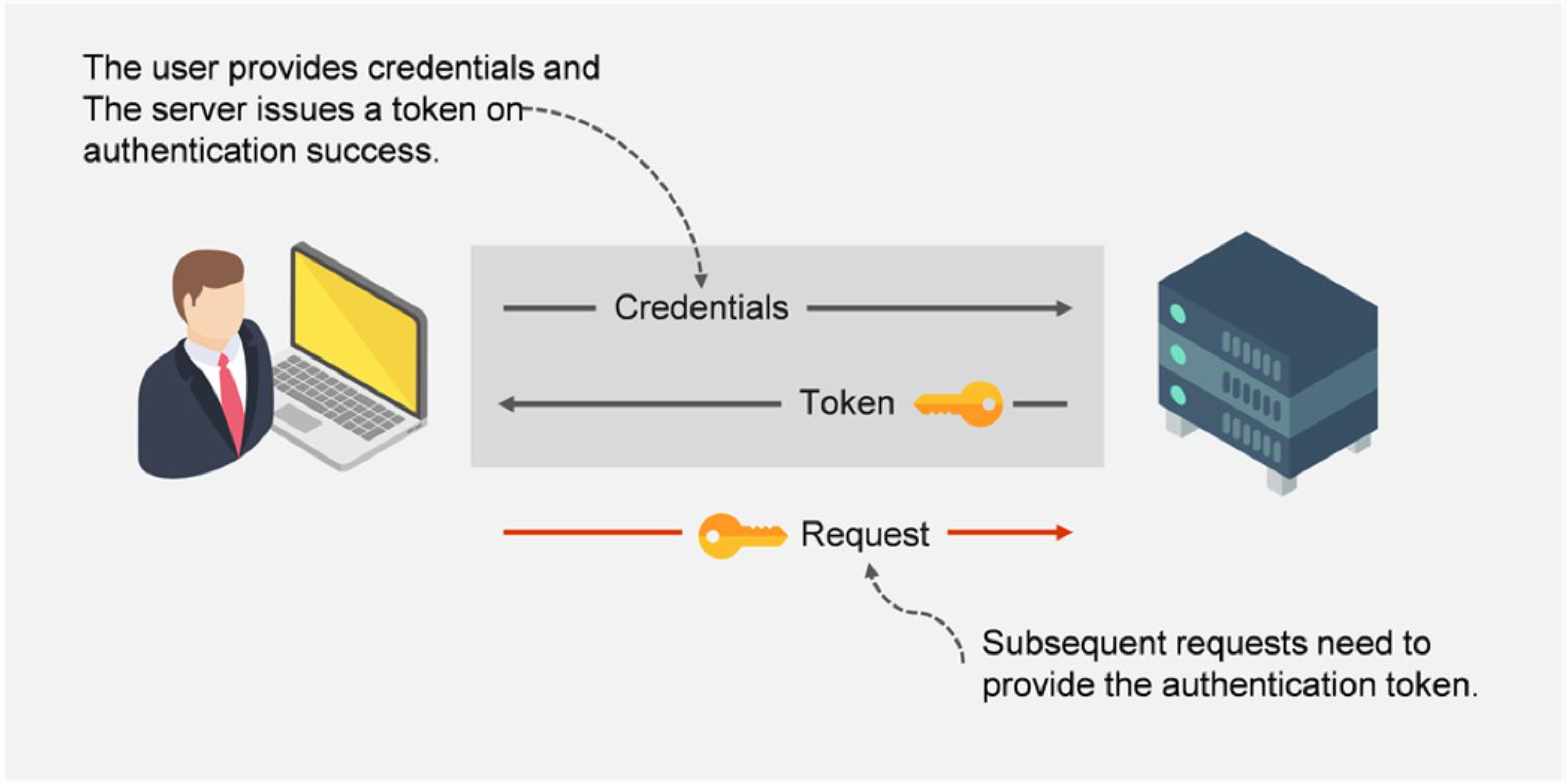
- Authorization
  - Once the user is logged in, each subsequent request will include JWT, allowing the user to access controllers, services, and resources that are permitted with that token.
- Information Exchange
  - JSON web tokens are a good way of securely transmitting information between parties.

Menu

Explain the Working of JSON Web Token (JWT)

Slide Note

Token-based Authentication


The user provides credentials and The server issues a token on authentication success.
Credentials
Token
Request
Subsequent requests need to provide the authentication token.

Menu

Slide Note

The user first logs in the application

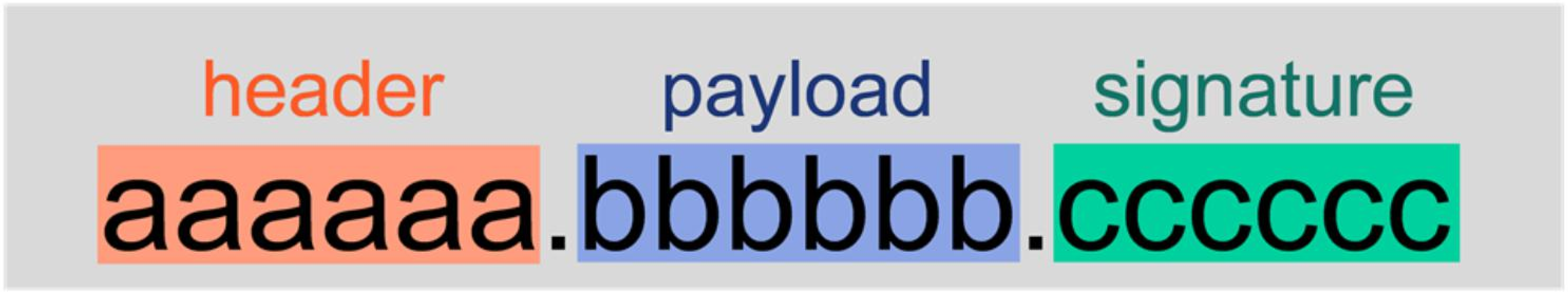
How Does the JWT Work?

- The user first signs into the application using valid credentials.
- The server authenticates the user and issues a JWT back to the client.
- When the user makes API calls to the application, the client passes the JWT along with the API call.
- The server verifies the validity of the incoming JWT that the client has passed.

Slide Note

The Structure of JWT

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



Menu

Slide Note

Header

- The header consists of two parts:
  - The type of token that is JWT
  - The signing algorithm being used (HMAC SHA256 in this case)
- Then, this JSON is Base64Url encoded to form the first part of the JWT.

```
{
  "alg":"HS256",
  "typ":"JWT"
}
```

HEADER
ALGORITHM &
TOKEN TYPE

Menu

00:16:00:00:38 16/ 38

Slide Note

The payload is the part of the JWT where all the user data is actually added. This data is also referred to as the 'claims' of the JWT.This information is readable by anyone so it is always advised to not put any confidential information in here. This part generally contains user information. This information is present as a JSON object then this JSON object is encoded to BASE64URL. We can put as many claims as we want inside a payload, though unlike header, no claims are mandatory in a payload. The JWT with the payload will look something like this:

The above JWT contains *userId,iss,sub,and exp.* All these play a different role as userId is the ID of the user we are storing, 'iss' tells us about the issuer, 'sub' stands for subject, and 'exp' stands for expiration date.

Payload

- The payload is the component of the JWT where all the user data is added.
- This data is also referred to as the "claims" of the JWT. Claims usually contain user information.
- This information is readable by anyone, so it's best to avoid putting any confidential information here.
- This payload in the diagram shown contains userId, iss, sub, and exp.
  - Here userId is the ID of the user you are storing.
  - Iss tells about issuer. Sub stands for subject.
  - Exp stands for expiration date.

Click on the link for more information on claims.

```
{
    "userId":"abc56-7ef",
    "iss": "https://xyz.domain.com/",
    "sub": "auth/some-hash-here",
    "exp": 134567
}
```

PAYLOAD
DATA

Menu

00:17:00:00:38 17/ 38

Slide Note

In this algorithm, base64url encodes the header and the payload is created in steps 1 and 2. The algorithm then joins the resulting encoded strings together with a period (.) between them. In our pseudo code, this joined string is assigned to *data*. To get the JWT signature, the *data* string is hashed with the secret key using the hashing algorithm specified in the JWT header.

The secret is the signature held by the server. This is how the server will be able to verify existing tokens and sign new ones.

Signature

- The third and final part of JWT is the signature.
- It is used to verify the authenticity of token.
- This signature is made up of the following components:
  - the header
  - the payload
- Secret
  - The secret is the signature held by the server. This is how the server will be able to verify existing tokens and sign new ones.

```
{
    HMACSHA256(
    base64UrlEncode(header) + " . " +
    base64UrlEncode(payload),
    secretkey)
}
```

SIGNATURE VERIFICATION

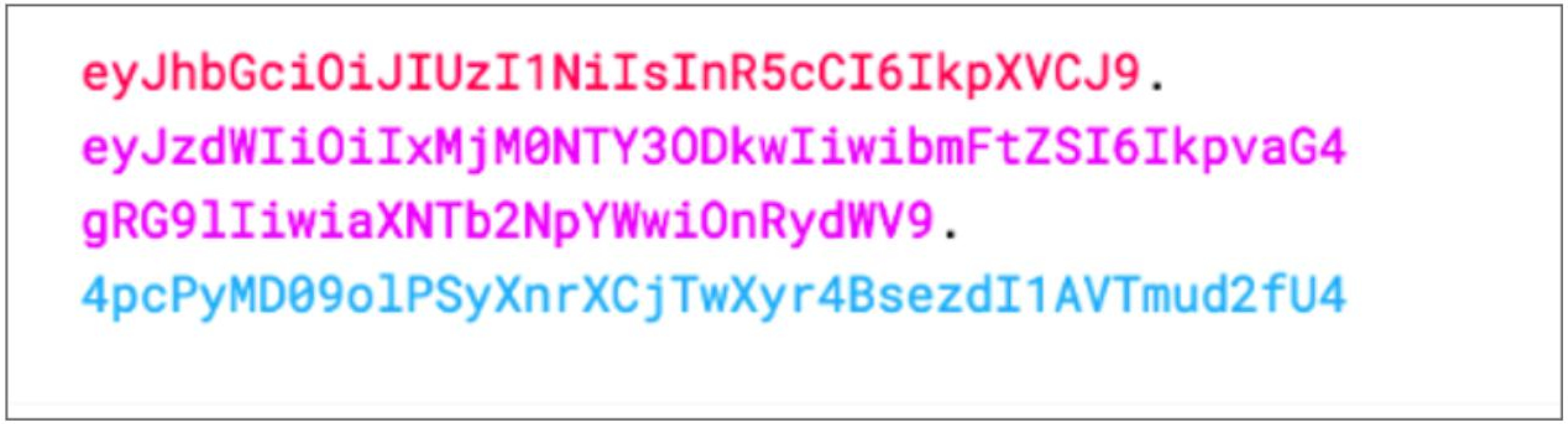
Menu

00:18:00:00:38 18/ 38

Slide Note

A Demo Structure of JWT

- The image below shows a JWT that has a header and payload encoded, and it is signed with a secret key.
- This output is three Base64-URL strings separated by dots that can be easily passed in HTML or HTTP environments.



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Menu

Slide Note

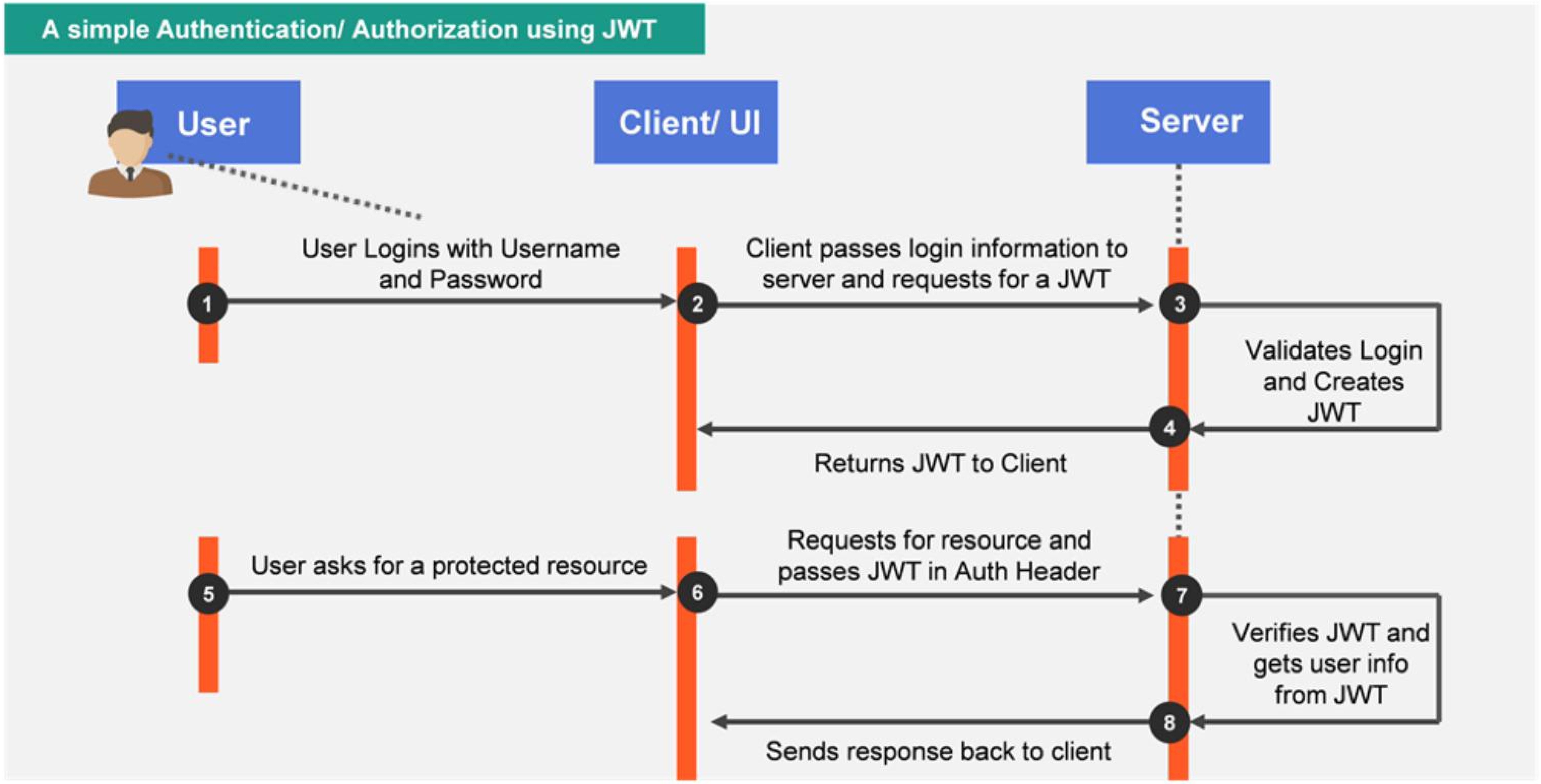
Implement JWT for Authentication

Menu

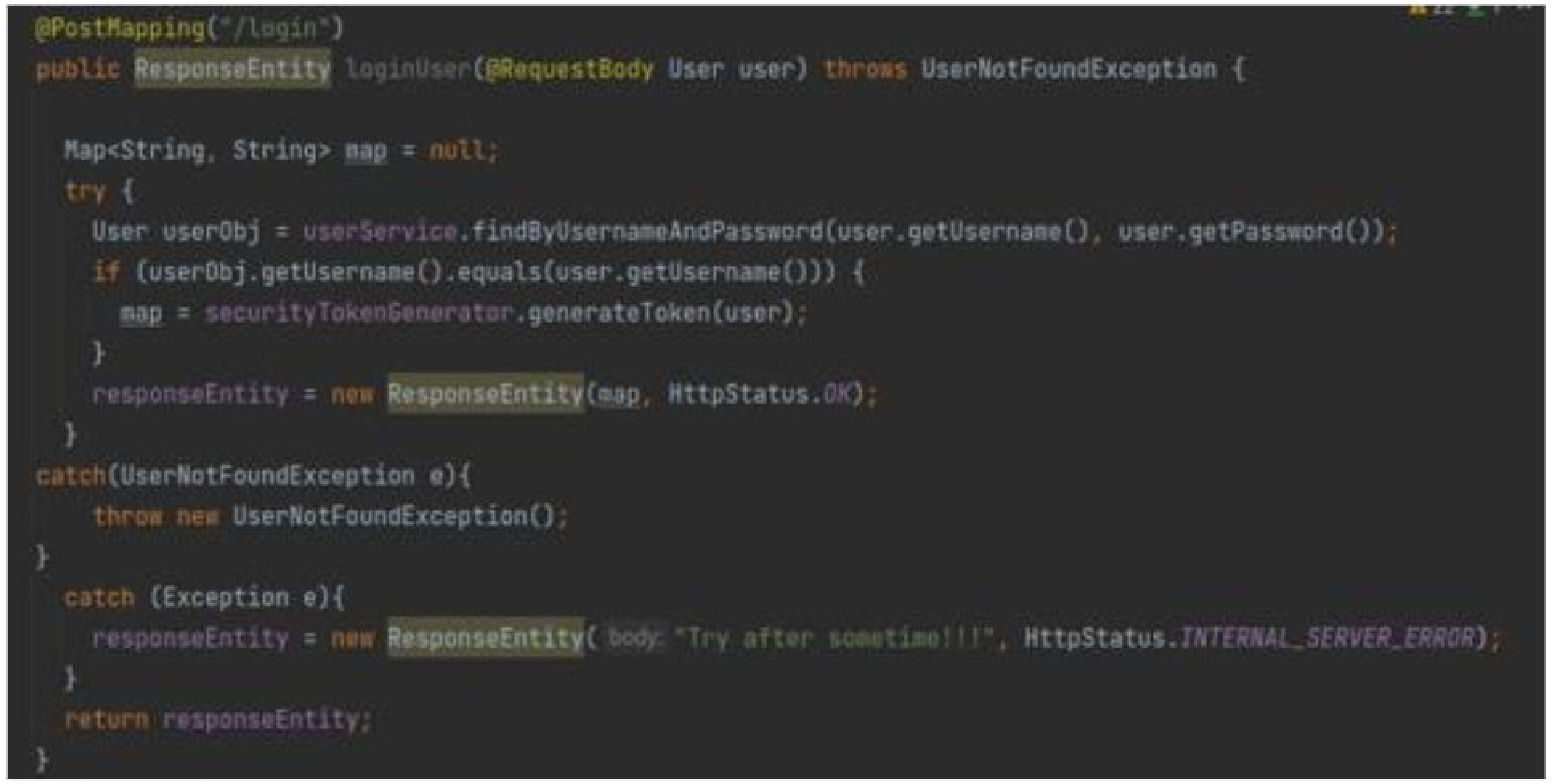
00:20 00:00:38 20/ 38

Slide Note

Data Flow of an Application Using JWT


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

The Controller Layer



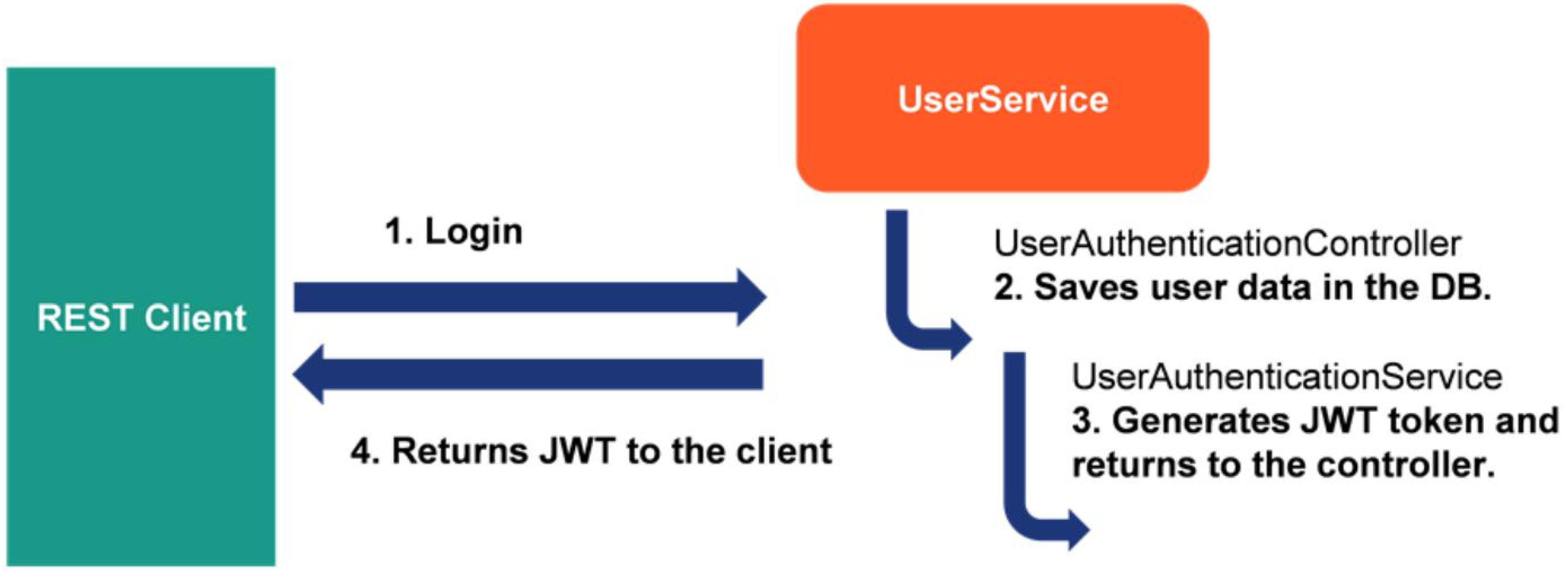
```
@PostMapping("/login")
public ResponseEntity loginUser(@RequestBody User user) throws UserNotFoundException {

  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( body: "Try after sometime!!!", HttpStatus.INTERNAL_SERVER_ERROR);
  }
  return responseEntity;
}
```

- Expose a POST API with mapping as /login.
- On passing the correct username and password, it will generate a JWT.

Slide Note

Flow Diagram – To Generate the Token


REST Client
UserService
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.

Menu

Slide Note

Jwts.builder - Returns a new JwtBuilder instance that can be configured and then used to create JWT compact serialized strings.

setSubject - Sets the JWT Claims sub (subject) value.

setIssuesAt - Sets the JWT Claims iat (issued at) value.

signWith - Signs the constructed JWT using the specified algorithm with the specified key, producing a JWS.

Compact - Actually builds the JWT and serializes it to a compact, URL-safe string according to the JWT Compact Serialization rules.

JSON Web Tokens are tokens that are not only unique to a user but also contain whatever information you need for that user, the so called claims.

The most basic claim is the 'subject' (basically a unique user ID).

These claims can then be retrieved from the JWT whenever the client sends the JWT to the server.

JWT's are either encrypted with a secure key (only known to the server) or signed.

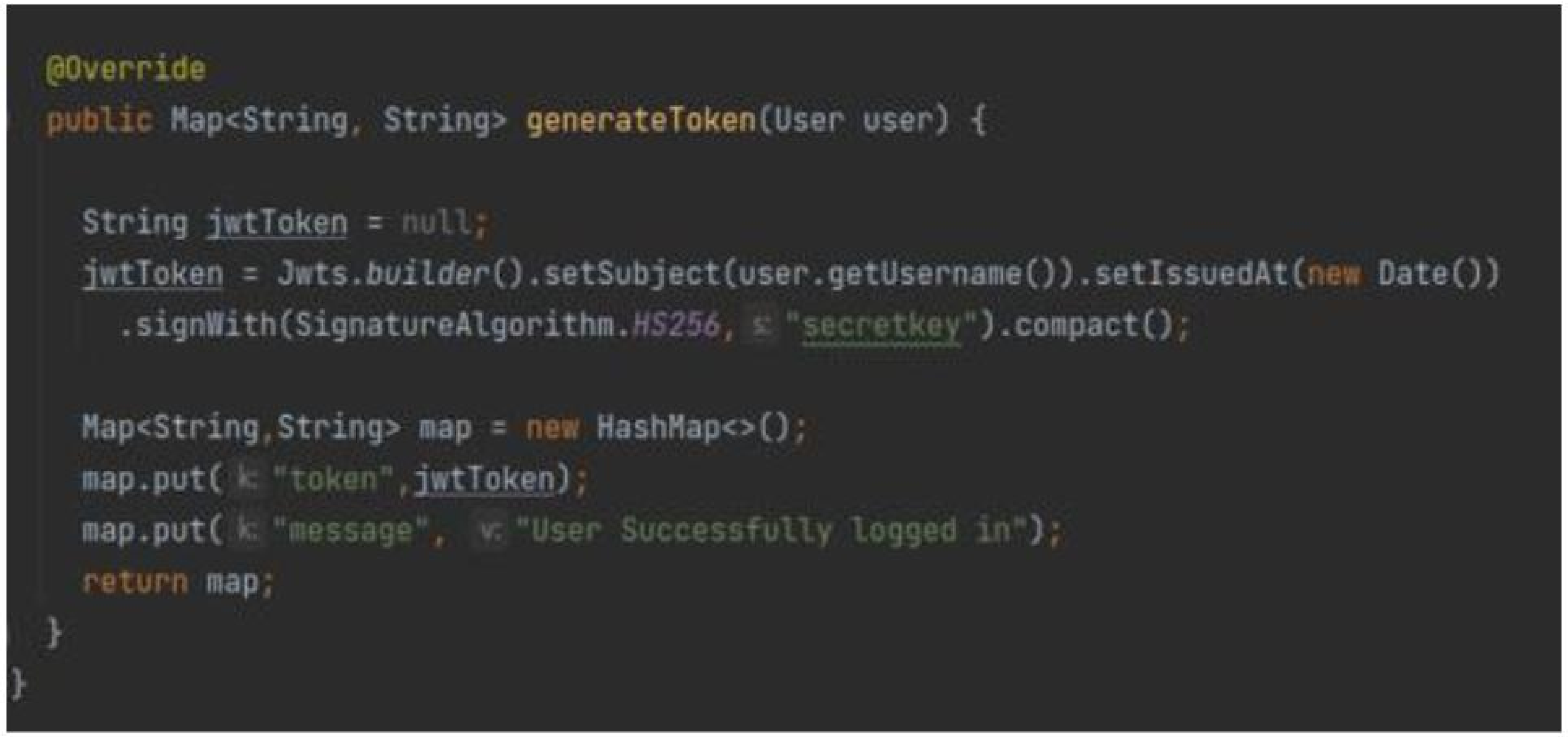
The most common approach when you use JWTs is by signing them. This 'signed' bit of the JWT is called the JWS, JSON Web Signature

This 'token' is then used on subsequent API calls from the client to the server.

The standard approach here is to send an Authorization header with a "Bearer" token.

The final JWT will be a three-part base64-encoded string, signed with the specified signature algorithm, and using the provided key. After this point, the token is ready to be shared with the another party.

Generate a Token



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

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

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

- The final JWT is a three-part base64-encoded string, signed with the specified signature algorithm and using the provided key.
- Now the token is ready to be passed between client and server.
- The generated token should be used to access privileged REST endpoints.

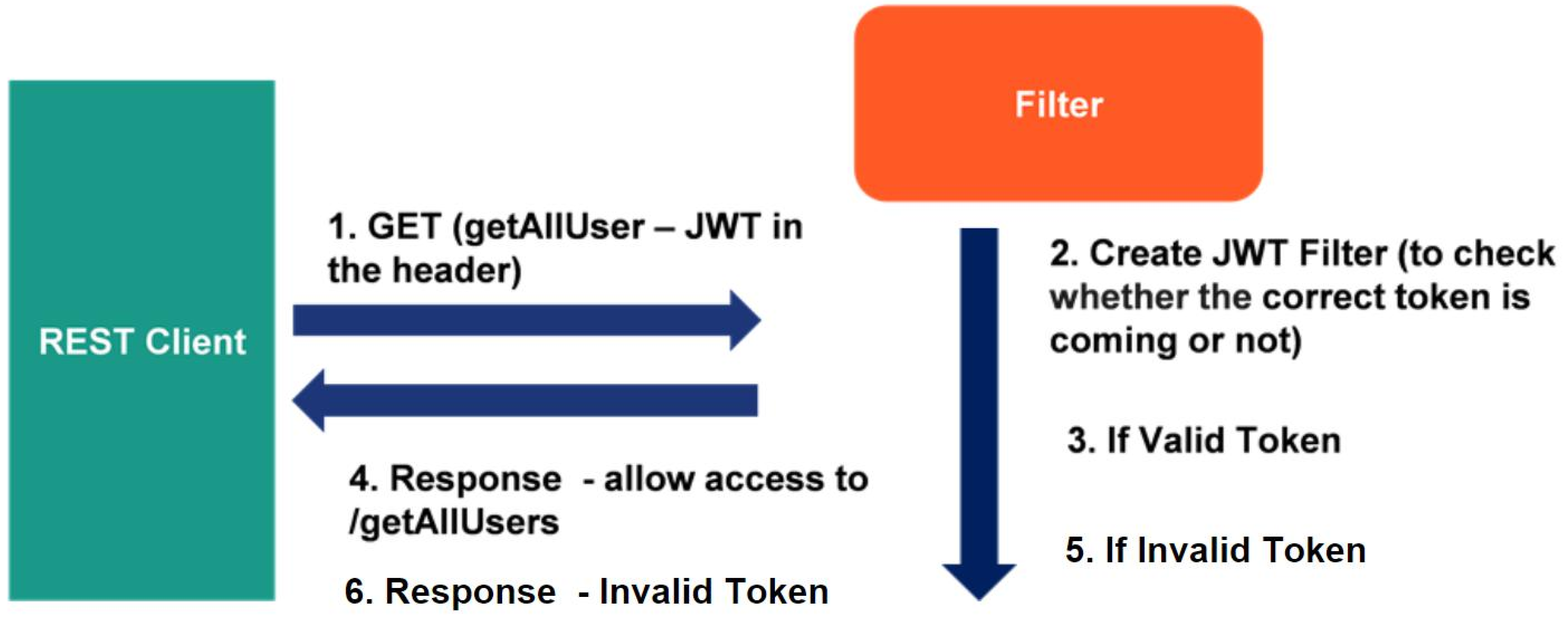
Menu

00:24 00:00:38 24/ 38

Slide Note

Access Privileged REST Endpoints

The GET mapping endpoint /`getAllUsers()` is a privileged endpoint, i.e., only requests with a JWT can access the endpoints.


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

Menu

Slide Note

A **filter** is an object that is invoked at the preprocessing and postprocessing of a request.

The filter used here is GenericFilterBean, which is a class.

It implements the filter interface.

Usage of Filter

• Recording all incoming requests

• Logs the IP addresses of the computers from which the requests originate

• Conversion

• Data compression

• Encryption and decryption

• Input validation, etc.

Advantage of Filter

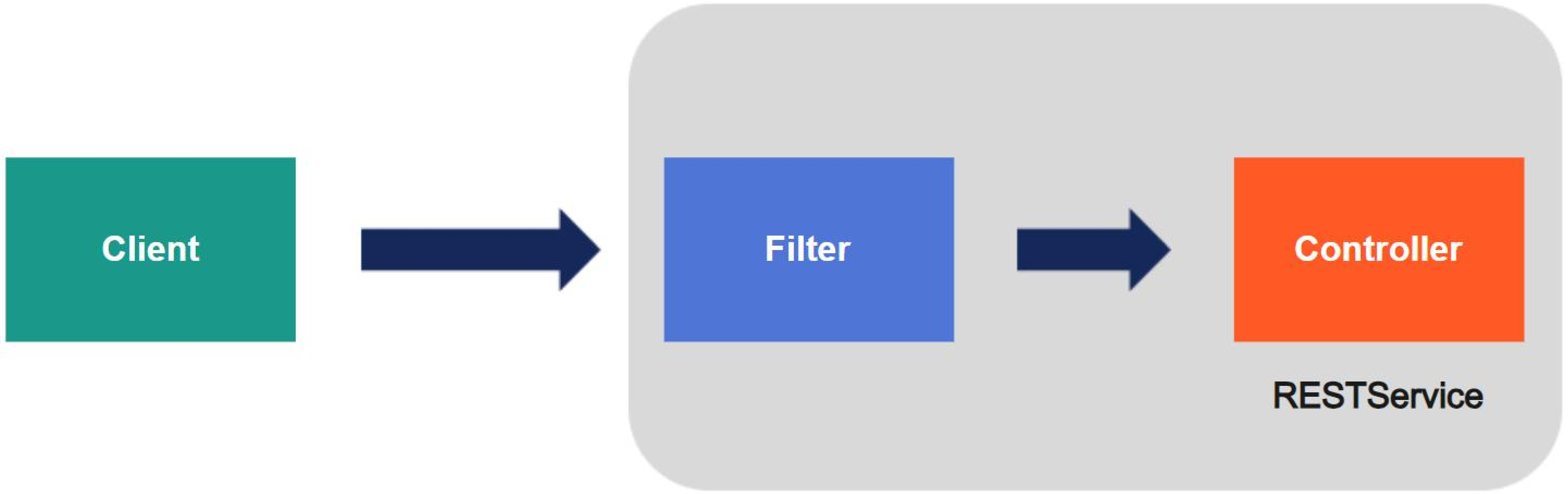
• Filter is pluggable.

• One filter don't have dependency onto another resource.

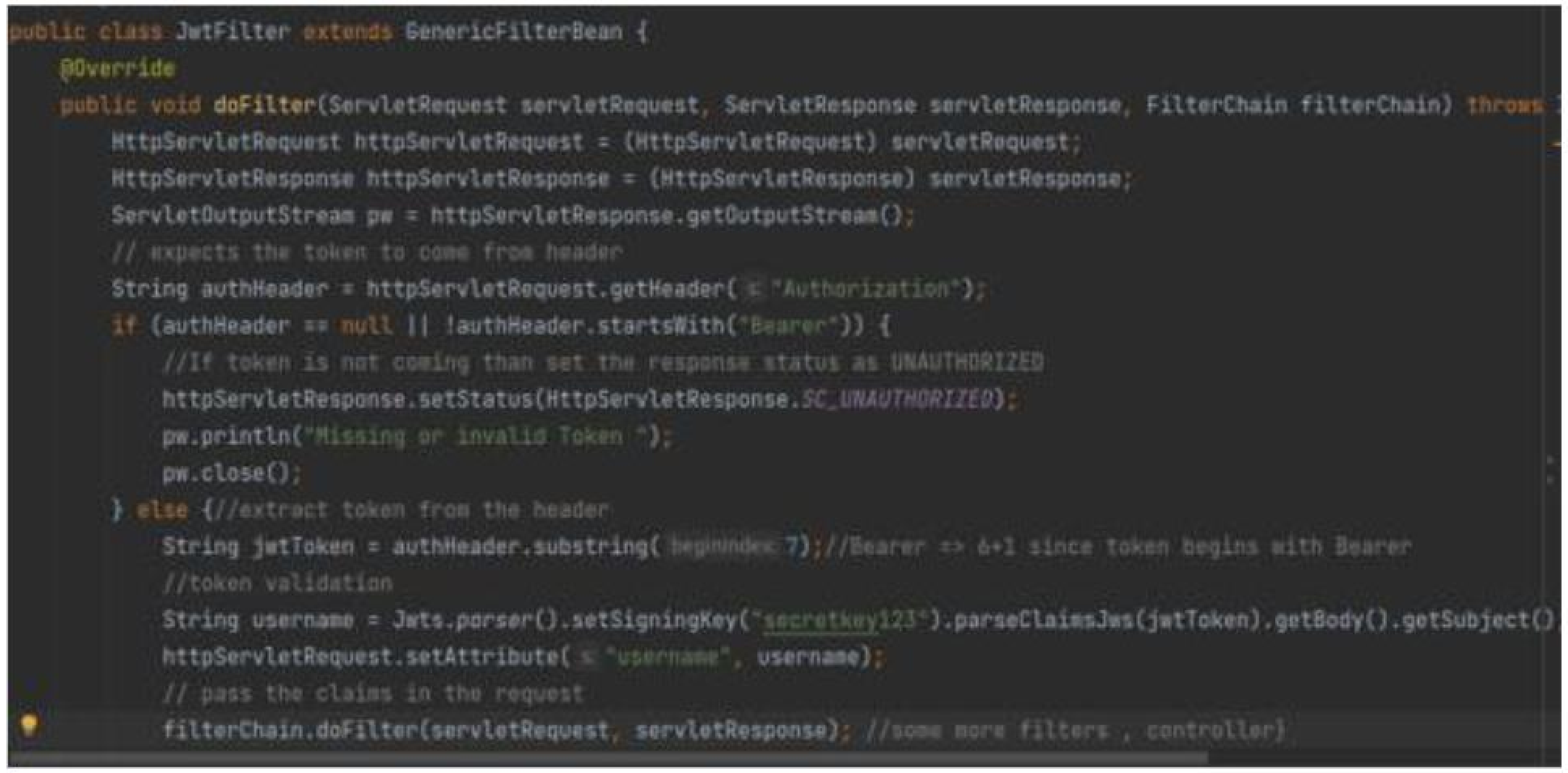
• Less maintenance.

Verification of JWT Token by Filter

- A filter is an object that intercepts the incoming request and performs the pre-processing and postprocessing of the request.



GenericFilterBean

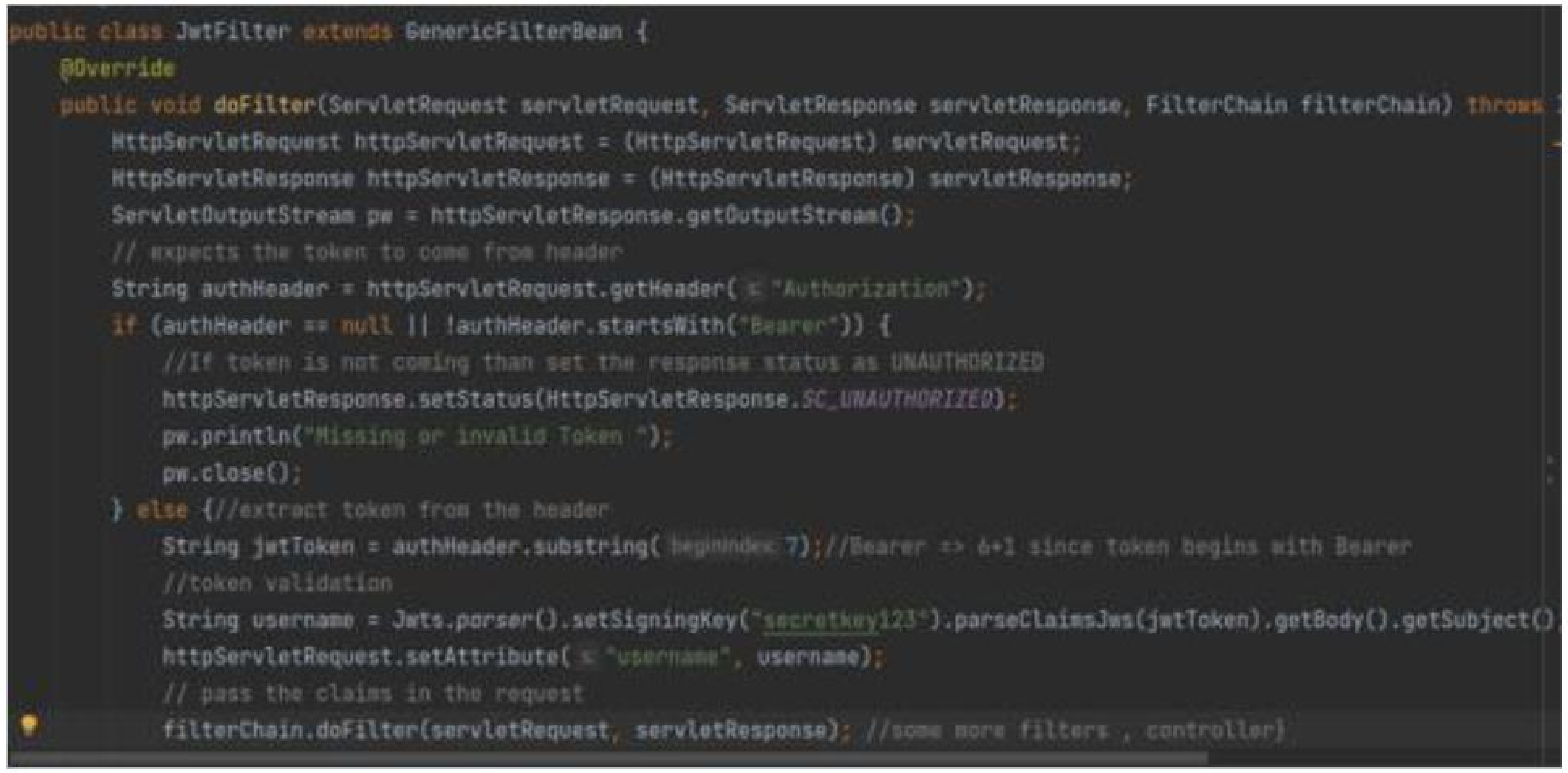


```java
public class JwtFilter extends GenericFilterBean {
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws 
        HttpServletRequest httpServletRequest = (HttpServletRequest) servletRequest;
        HttpServletResponse httpServletResponse = (HttpServletResponse) servletResponse;
        ServletOutputStream pw = httpServletResponse.getOutputStream();
        // expects the token to come from header
        String authHeader = httpServletRequest.getHeader( s: "Authorization");
        if (authHeader == null || !authHeader.startsWith("Bearer")) {
            //If token is not coming than set the response status as UNAUTHORIZED
            httpServletResponse.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            pw.println("Missing or invalid Token ");
            pw.close();
        } else {//extract token from the header
            String jwtToken = authHeader.substring( beginIndex: 7);//Bearer => 6+1 since token begins with Bearer
            //token validation
            String username = Jwts.parser().setSigningKey("secretkey123").parseClaimsJws(jwtToken).getBody().getSubject()
            httpServletRequest.setAttribute( s: "username", username);
            // pass the claims in the request
            filterChain.doFilter(servletRequest, servletResponse); //some more filters , controller)
```

- You will be using the filter to protect the 'secure' endpoints.
- To get the Spring security support, we can use the GenericFilterBean abstract class.
- The filter will be configured inside the main class.
- The filter is responsible for checking if the correct authorization header is coming from the client.
- The JWT parser is used to check the token signature with the same key we used to sign it.

Slide Note

GenericFilterBean (contd.)



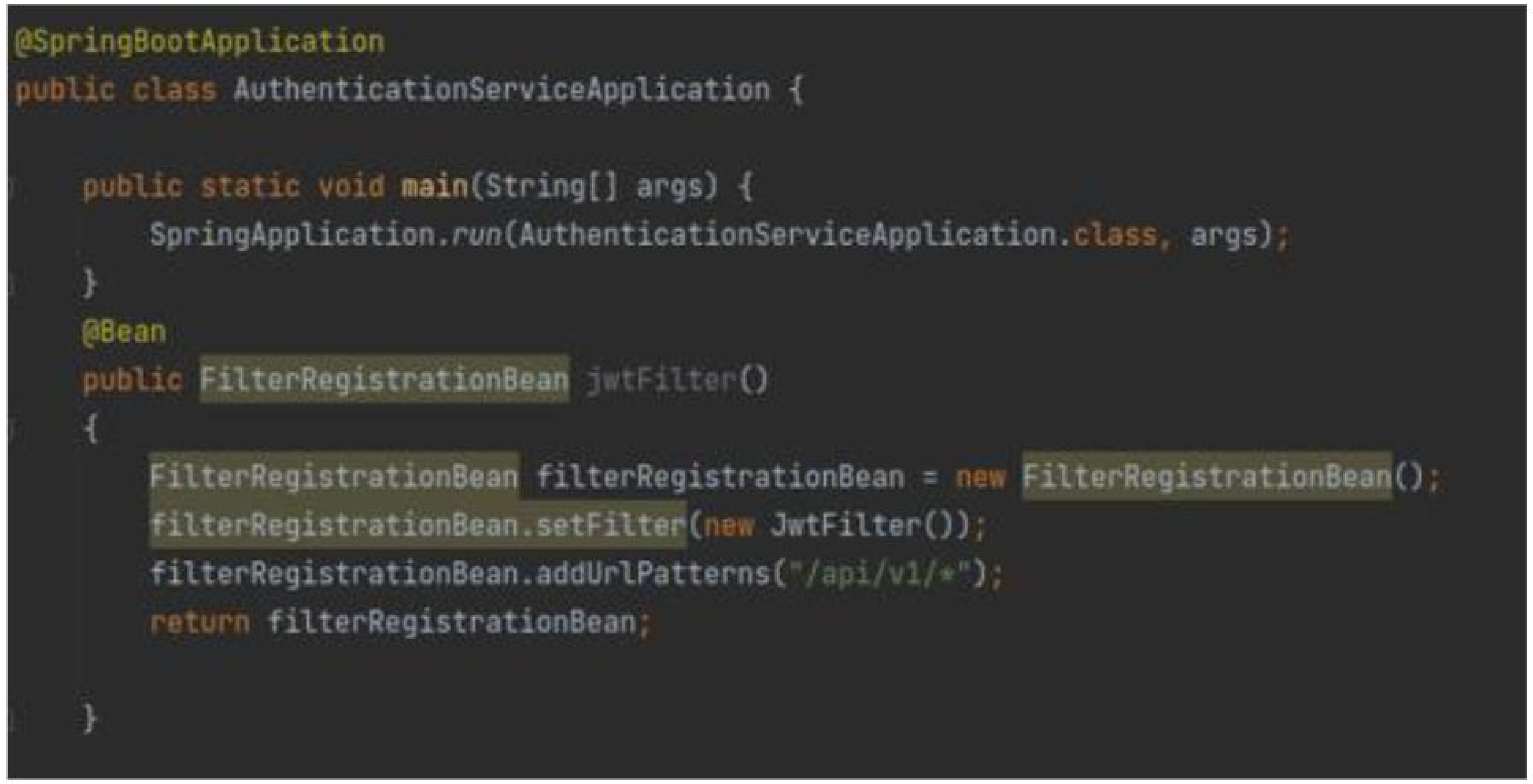
```
public class JwtFilter extends GenericFilterBean {
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws
        HttpServletRequest httpServletRequest = (HttpServletRequest) servletRequest;
        HttpServletResponse httpServletResponse = (HttpServletResponse) servletResponse;
        ServletOutputStream pw = httpServletResponse.getOutputStream();
        // expects the token to come from header
        String authHeader = httpServletRequest.getHeader( s: "Authorization");
        if (authHeader == null || !authHeader.startsWith("Bearer")) {
            //If token is not coming than set the response status as UNAUTHORIZED
            httpServletResponse.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            pw.println("Missing or invalid Token ");
            pw.close();
        } else {//extract token from the header
            String jwtToken = authHeader.substring( beginIndex: 7);//Bearer => 6+1 since token begins with Bearer
            //token validation
            String username = Jwts.parser().setSigningKey("secretkey123").parseClaimsJws(jwtToken).getBody().getSubject()
            httpServletRequest.setAttribute( s: "username", username);
            // pass the claims in the request
            filterChain.doFilter(servletRequest, servletResponse); //some more filters , controller)
```

- If the key is valid, we then store the "Claims" that contain some user information (email, role) in the request object so it can be used by API endpoints down the line.
- The filter method doFilter will help intercept all requests that come from the client.
- The request and response objects will be the same objects that originated from client; the filter will just pass the objects to the appropriate controllers.
- The request header will contain the JWT token.

Menu

Slide Note

Registering a Filter



```
@SpringBootApplication
public class AuthenticationServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(AuthenticationServiceApplication.class, args);
    }
    @Bean
    public FilterRegistrationBean jwtFilter()
    {
        FilterRegistrationBean filterRegistrationBean = new FilterRegistrationBean();
        filterRegistrationBean.setFilter(new JwtFilter());
        filterRegistrationBean.addUrlPatterns("/api/v1/*");
        return filterRegistrationBean;

    }
```

- FilterRegistrationBean is a class.
- It is used to register a filter programmatically in a Spring Boot application.
- Here, the Spring Boot filter is only applied for the URL - "/api/v1/*". This means the request coming to this URL will be first processed by the filter and then it will come to controller.

Menu

Slide Note

Quick Check

Which of the following are parts of a JSON Web Token?

1. Header
2. Payload
3. Signature
4. Footer

Menu

Slide Note

Quick Check: Solution

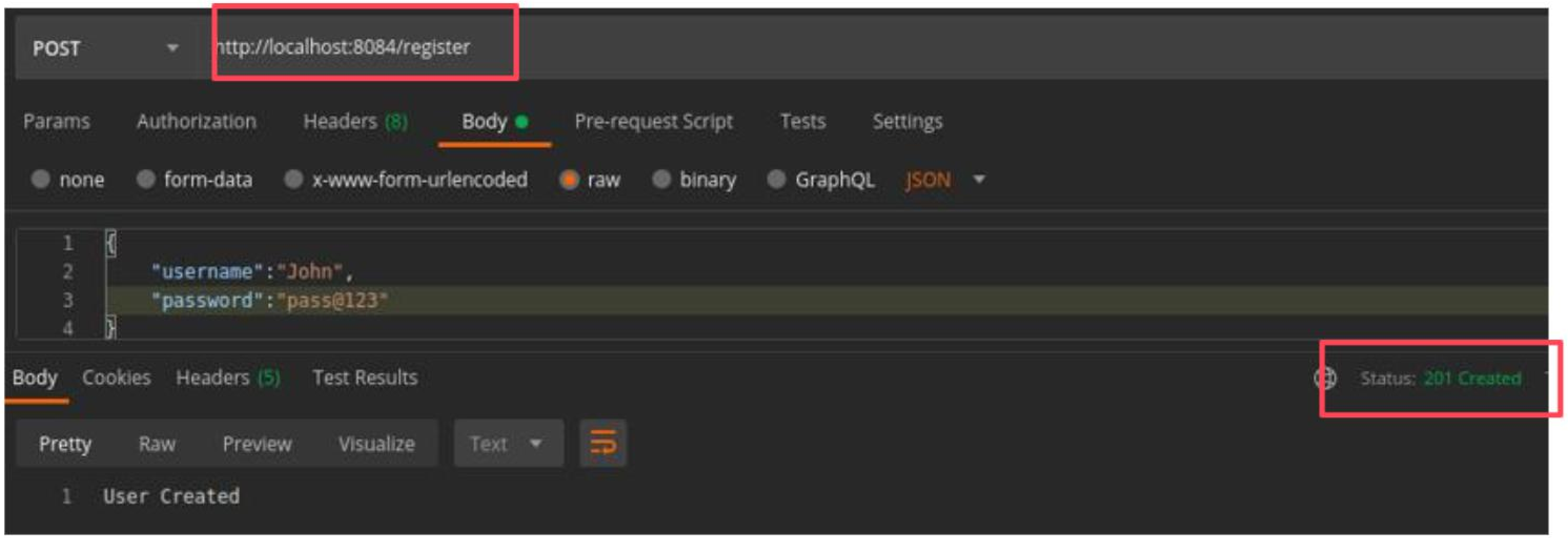
Which of the following are parts of a JSON Web Token?

1. Header
2. Payload
3. Signature
4. Footer

Menu

Slide Note

Step 1 – Postman Output (Register)



http://localhost:8084/register

```
{
    "username":"John",
    "password":"pass@123"
}
```

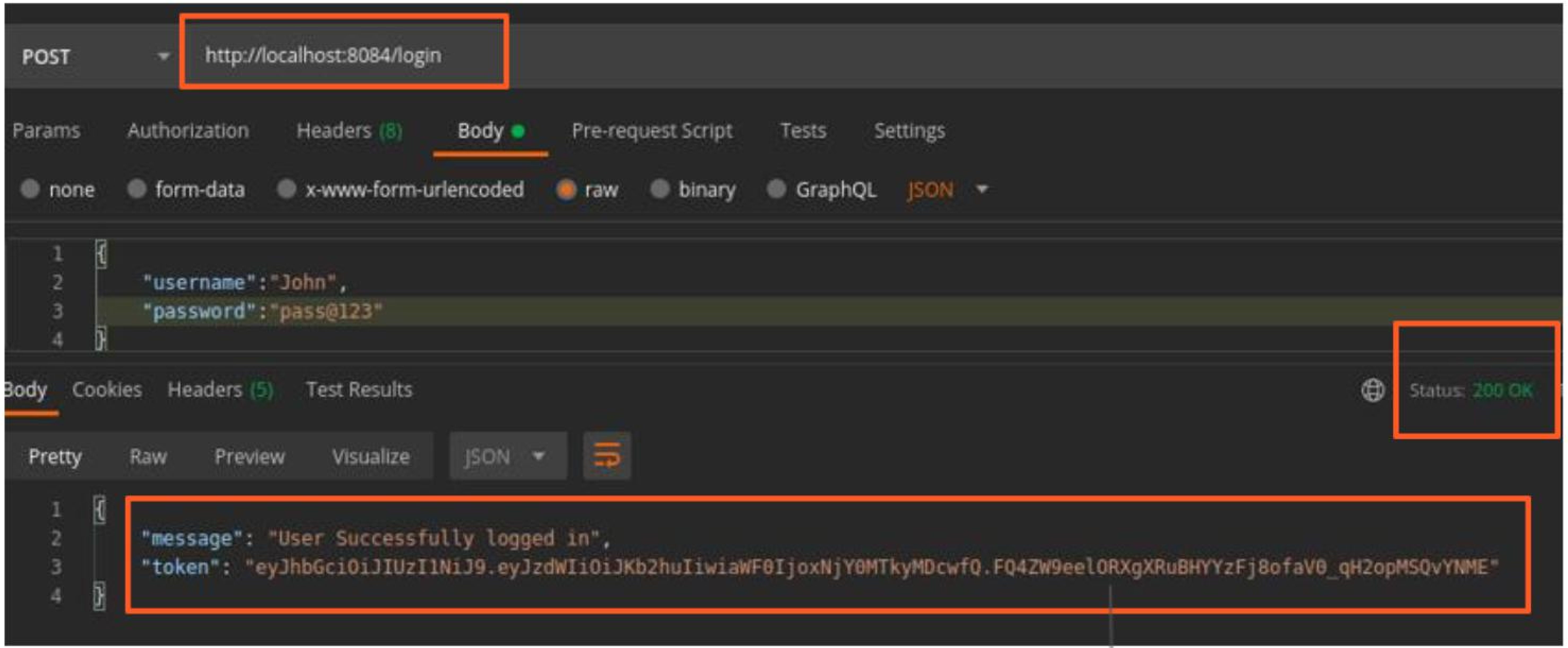
Status: 201 Created

```
User Created
```

Menu

00:32 00:00:38 32/ 38

Step 2 – Postman Output (Login)



http://localhost:8084/login

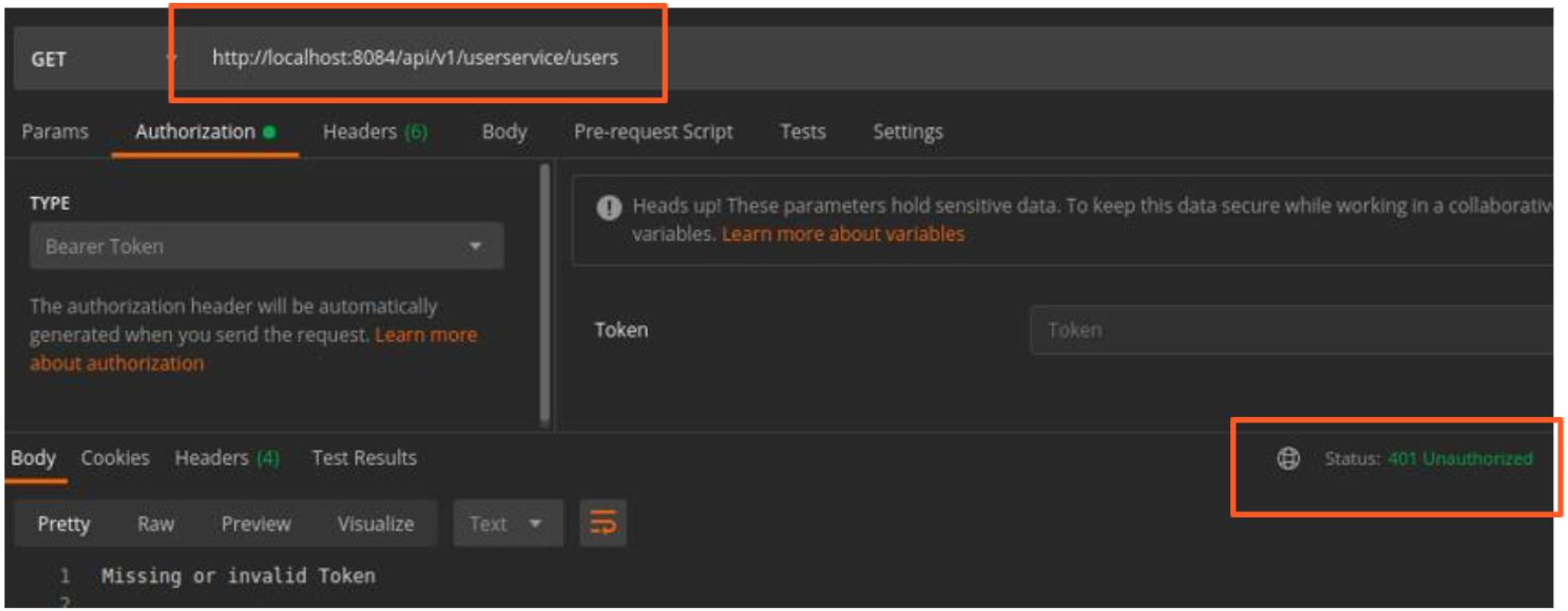
```
{
    "username":"John",
    "password":"pass@123"
}
```

Status: 200 OK

```
{
    "message": "User Successfully logged in",
    "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJKb2huIiwiaWF0IjoxNjY0MTkyMDcwfQ.FQ4ZW9eelORXgXRuBHYYzFj8ofaV0_qH2opMSQvYNME"
}
```

Token got generated

Step 3 – Postman Output (Without Token)



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

TYPE

Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about authorization

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborativ
variables. Learn more about variables

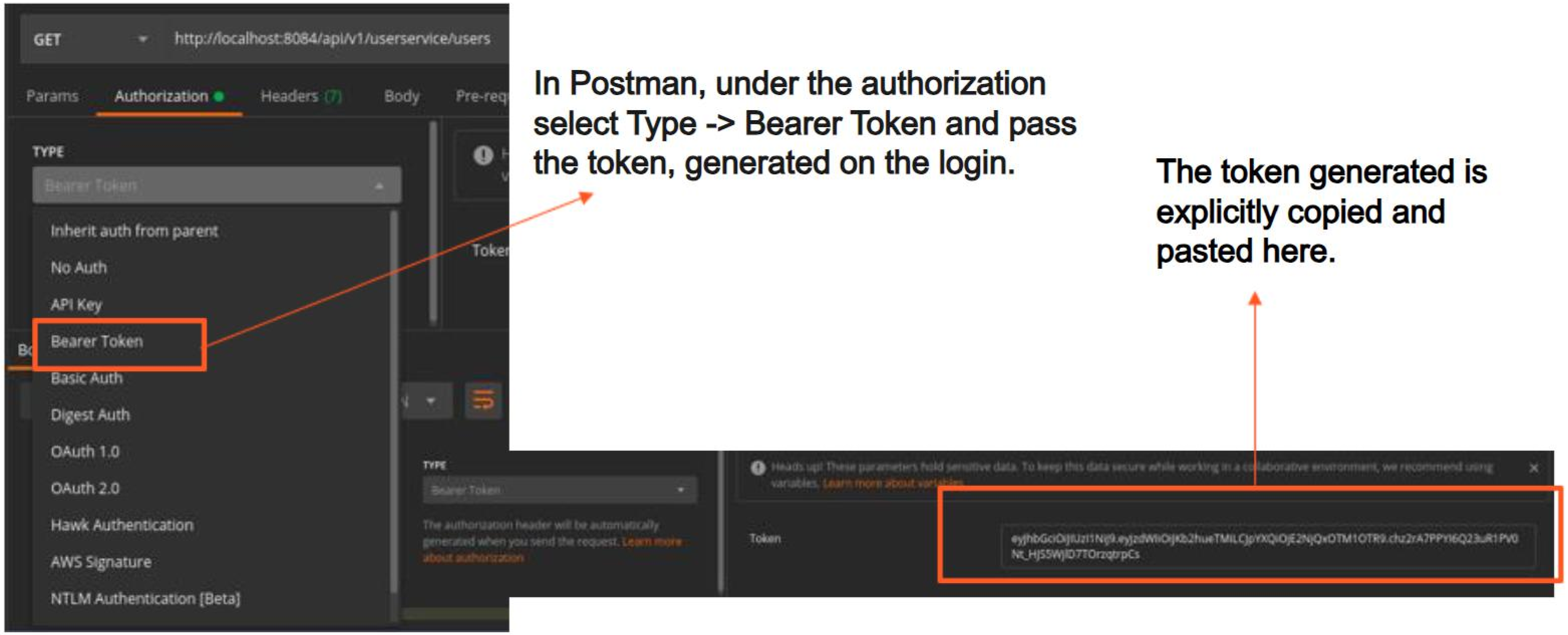
Token: Token

Status: 401 Unauthorized

```
Missing or invalid Token
```

Slide Note

Step 4 – Postman Output (Bearer Token Value)



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

The authorization header will be automatically generated when you send the request. Learn more about authorization

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.eyJzdWIiOiJKb2hueTMiLCJpYXQiOjE2NjQxOTM1OTR9.chz2rA7PPYI6Q23uR1PV0
         Nt_HJS5WjlD7TOrzqtrpCs
```

In Postman, under the authorization select Type -> Bearer Token and pass the token, generated on the login.

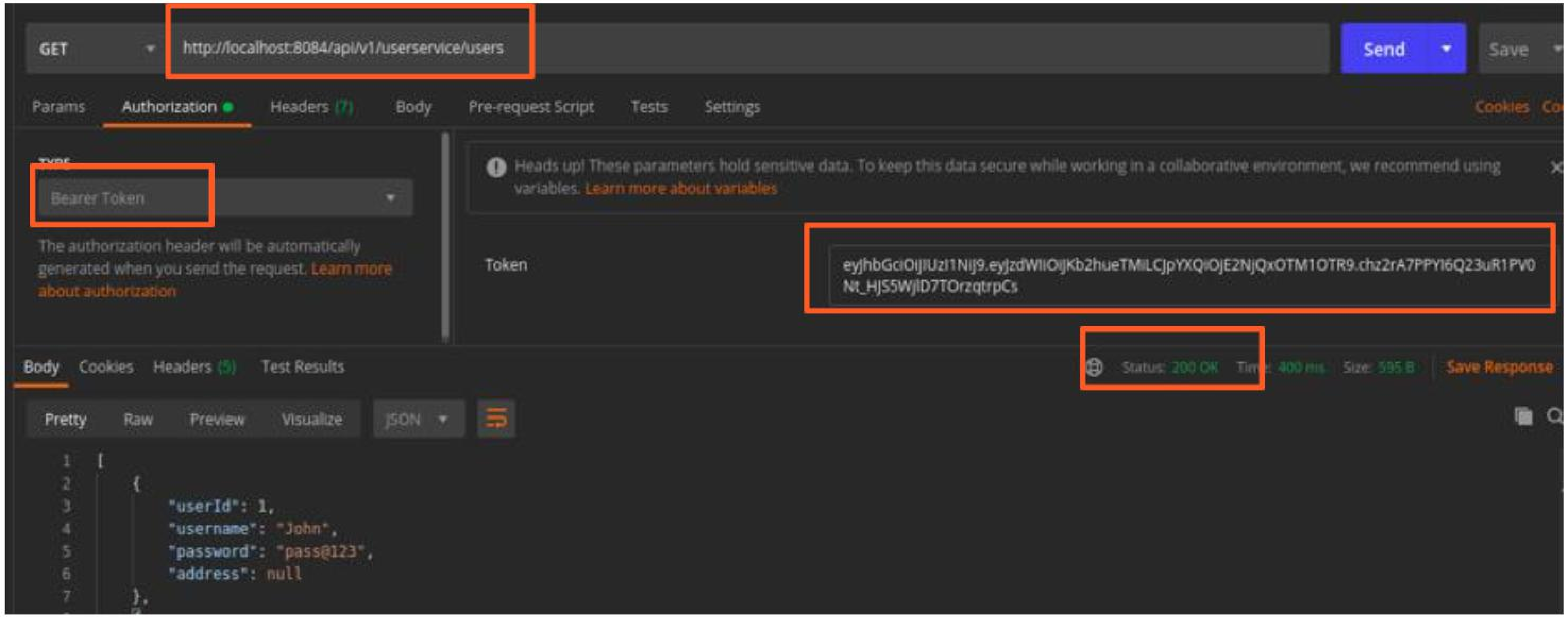
The token generated is explicitly copied and pasted here.

Menu

00:35 / 00:38

35/ 38

Step 5 – Postman Output (With Token)



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

TYPE: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about authorization

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.eyJzdWIiOiJKb2hueTMiLCJpYXQiOjE2NjQxOTM1OTR9.chz2rA7PPYI6Q23uR1PV0Nt_HJS5WjID7TOrzqtrpCs

Status: 200 OK Time: 400 ms Size: 595 B Save Response

```
[
    {
        "userId": 1,
        "username": "John",
        "password": "pass@123",
        "address": null
    },
```

Slide Note

1. First register the API that needs to be tested – the user information will get save d.

   Save 2 – 3 users information

2. Test the login API – the service should generate a JWT token.

3. Test getAllUsers() without passing the token – an error should be thrown from the controller as "Missing or invalid Authorization header."

4. Again, test getAllUsers() with sending the JWT token generated in step 2 in the header – as shown in the slide. Now, all of the users that are saved will be given back.

Authenticate the Application Using JWT Token

Write a Spring Boot application having features such as register user, login user, and getAllDetails of the user. Implement the JWT token.

Check the solution here.

DEMO

Menu