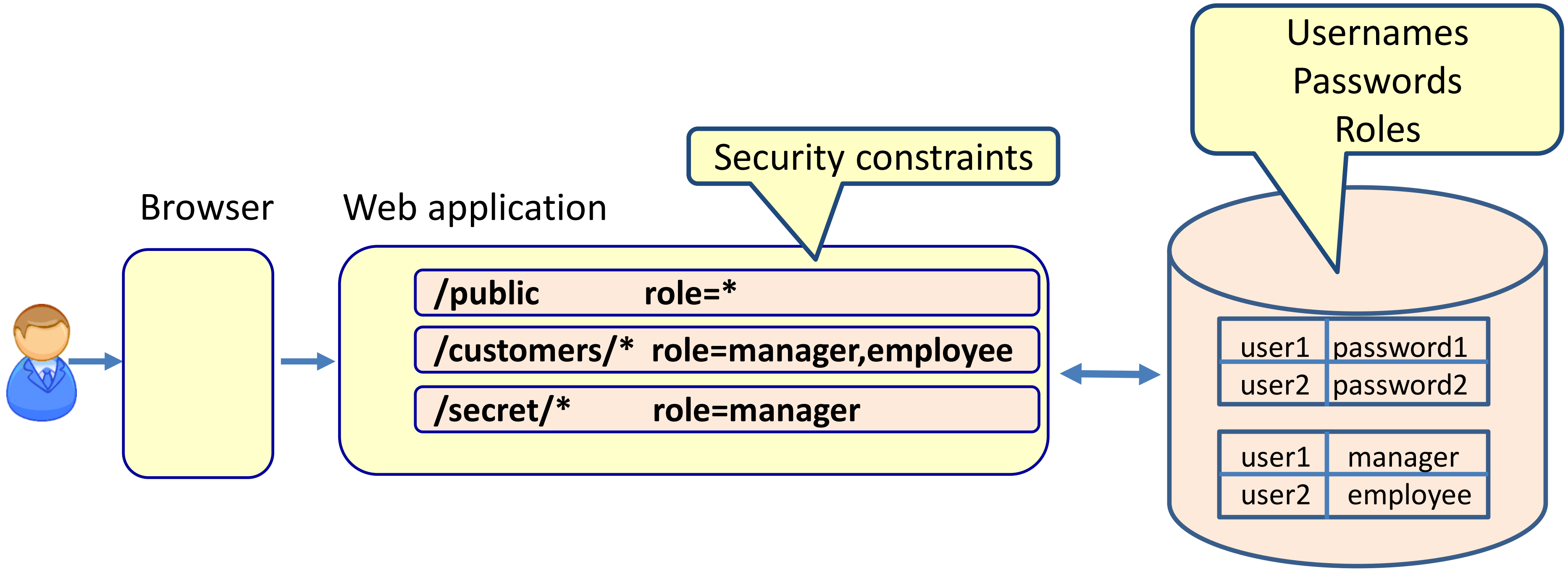
Lesson 8

SECURITY

Security

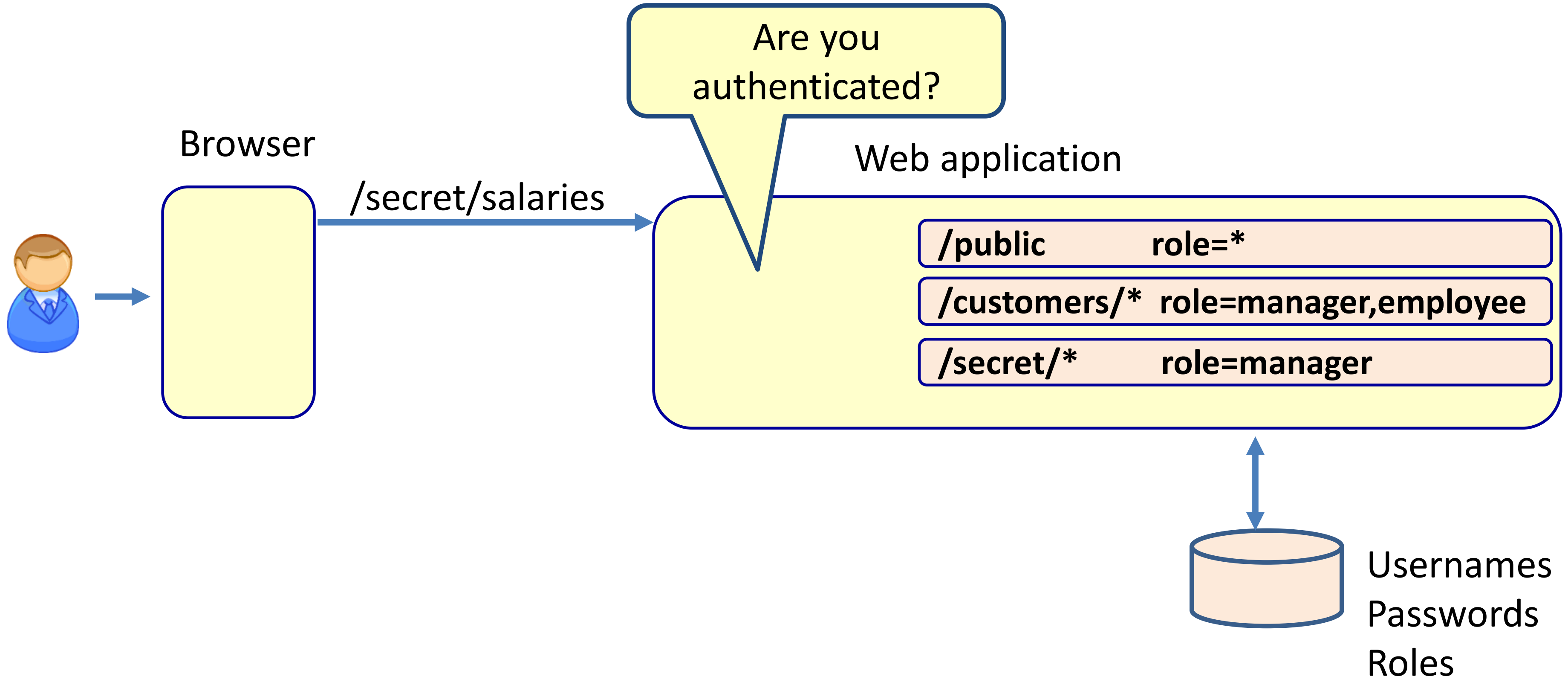
- Authentication
  - Are you who you say you are?
    - Username - password
- Authorization
  - Now that I know who you are, what are you allowed to do?

Role based security

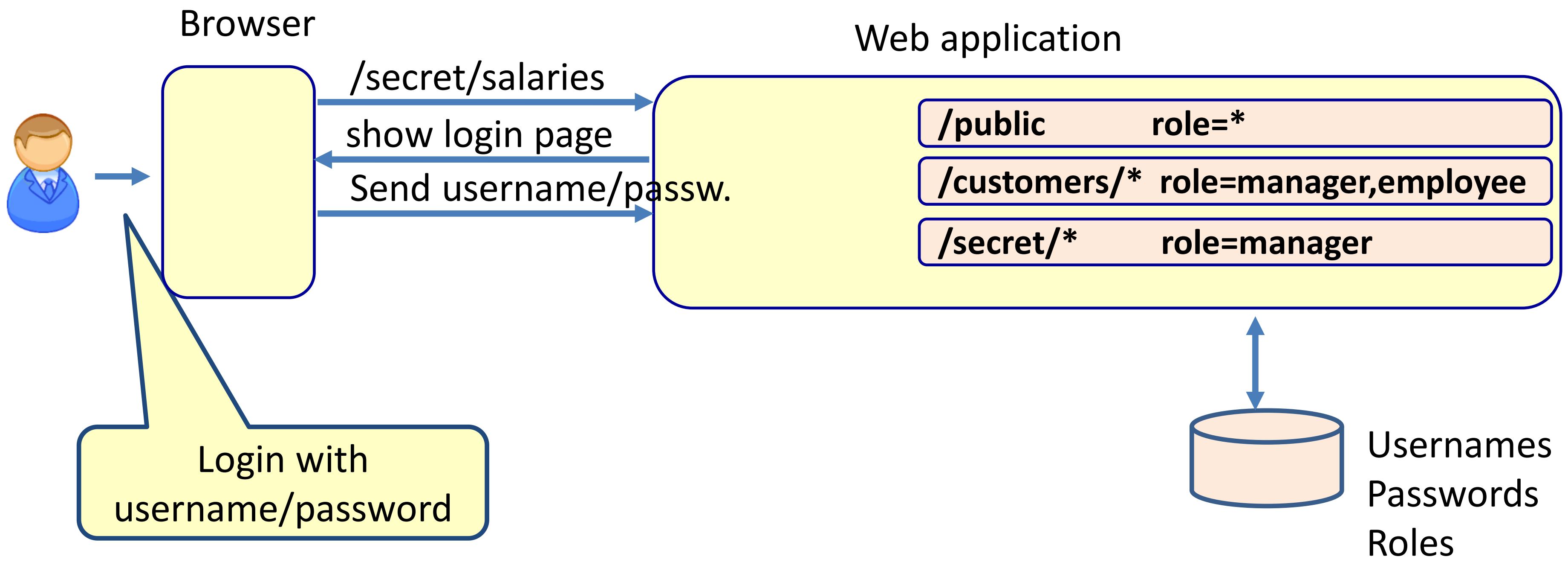


WEB APPLICATION SECURITY

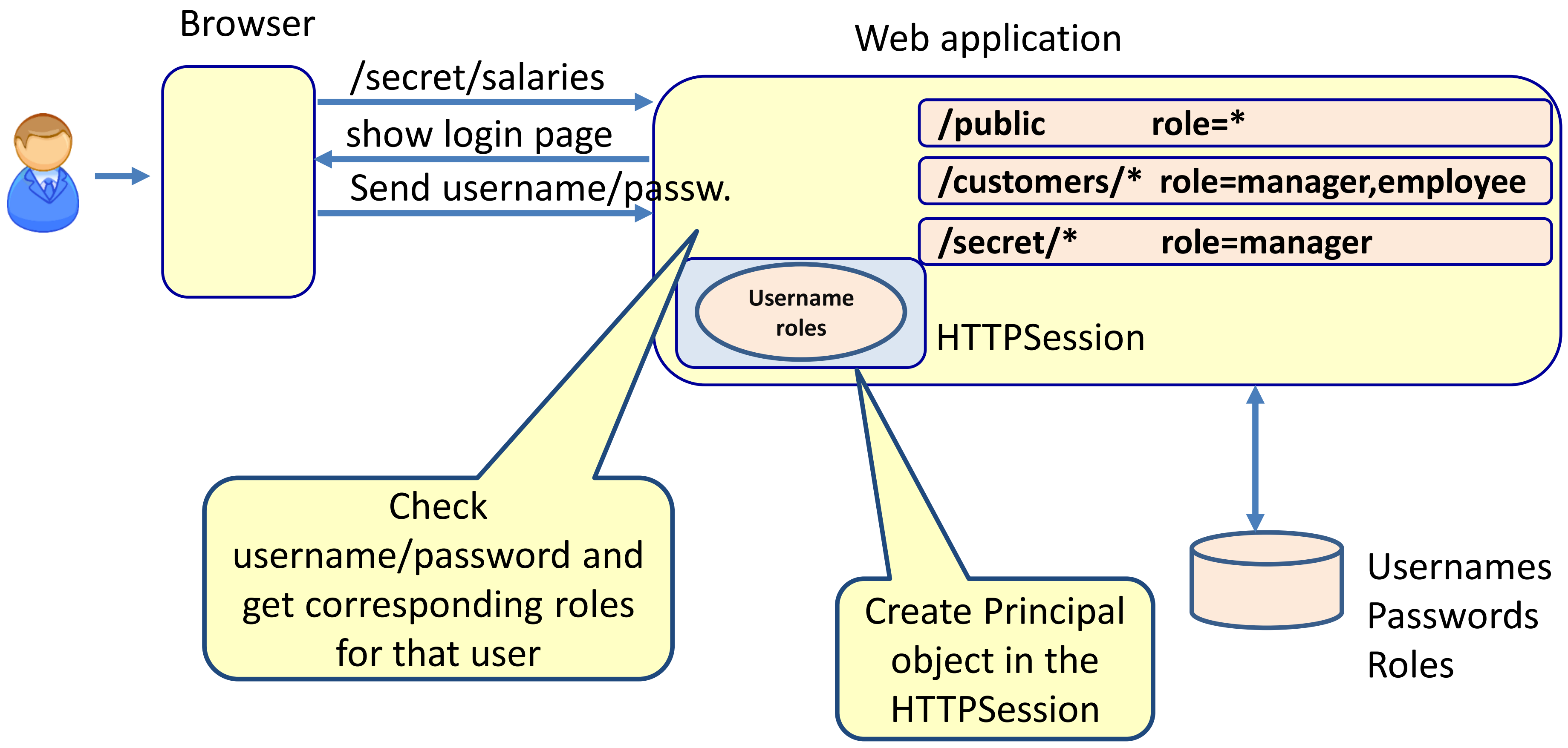
Web application security



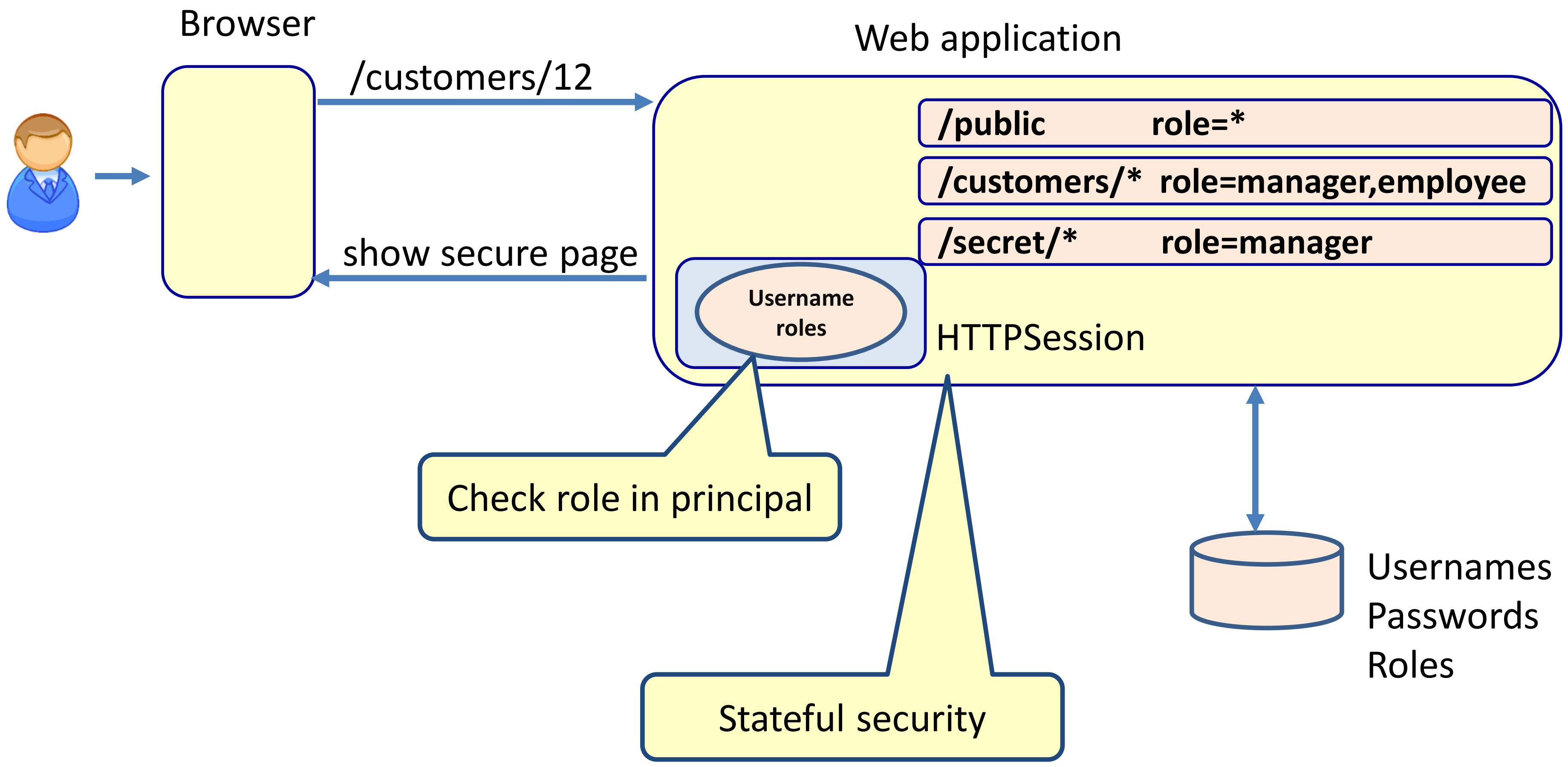
Web security



Web security



Web security



Spring boot security

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/**").hasAnyRole("USER")
                .and()
            .formLogin();
    }

    // create users
    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
                .withUser("user").password("{noop}pass").roles("USER");
    }
}
```

Give the role USER access to all pages

Use the default login form

Add an in-memory user

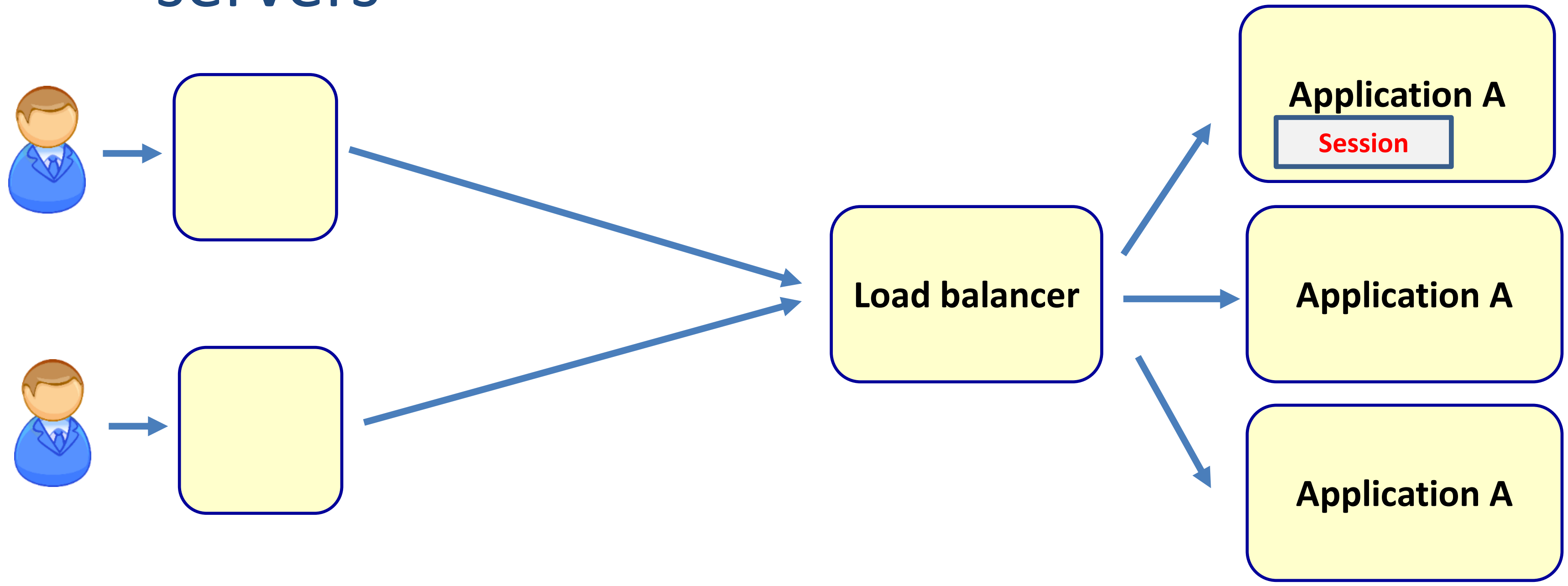
{noop} means do not use a password encoder

Getting security info from the database

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  DataSource dataSource;

  @Autowired
  public void configAuthentication(AuthenticationManagerBuilder auth) throws
      Exception {

    auth.jdbcAuthentication().dataSource(dataSource)
      .usersByUsernameQuery(
           "select username,password, enabled from users where username=?")
      .authoritiesByUsernameQuery(
           "select username, role from user_roles where username=?");
  }

...
```

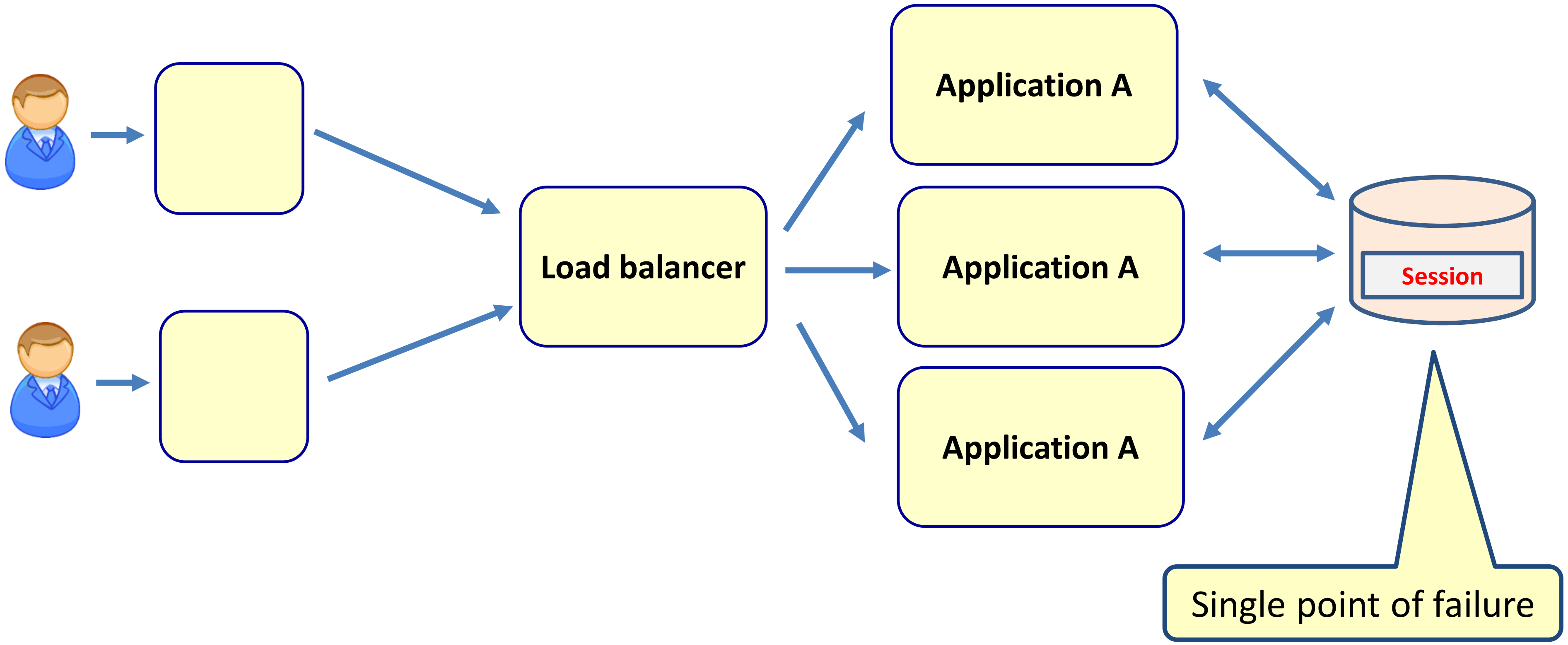
Problem with session-based security

- Hard to implement when we have multiple servers



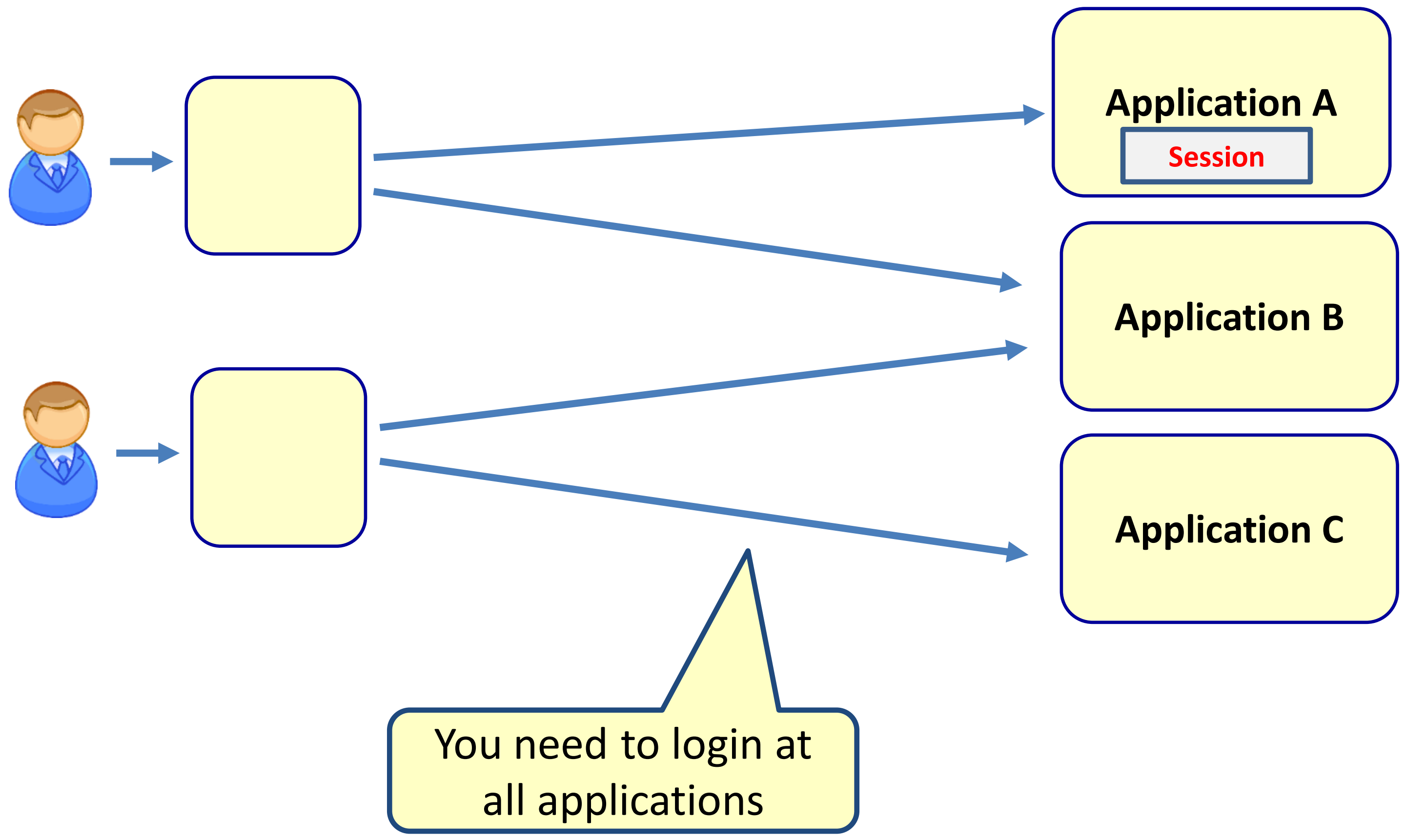
Problem with session-based security

- Use a central session store (Redis)



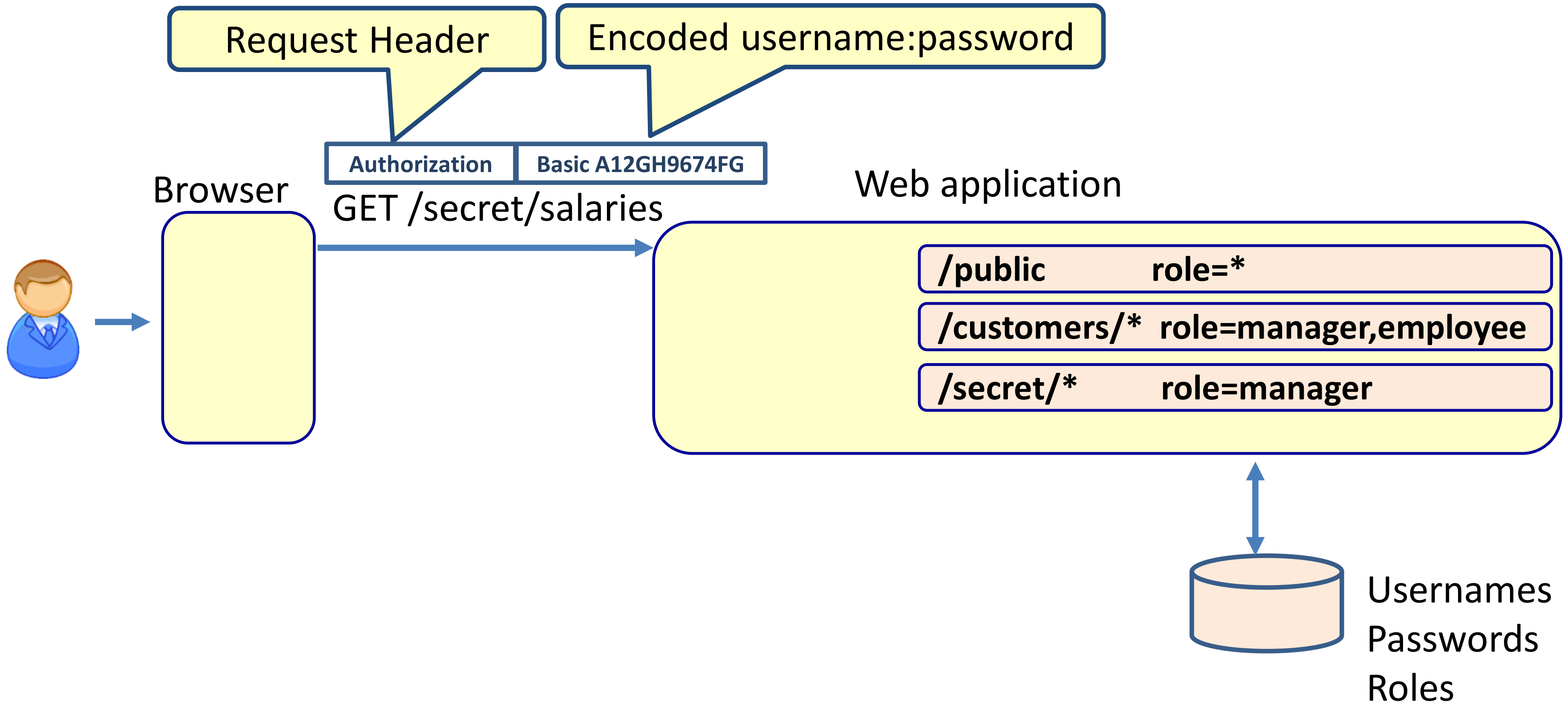
Problem with session-based security

- Calling multiple applications from one app

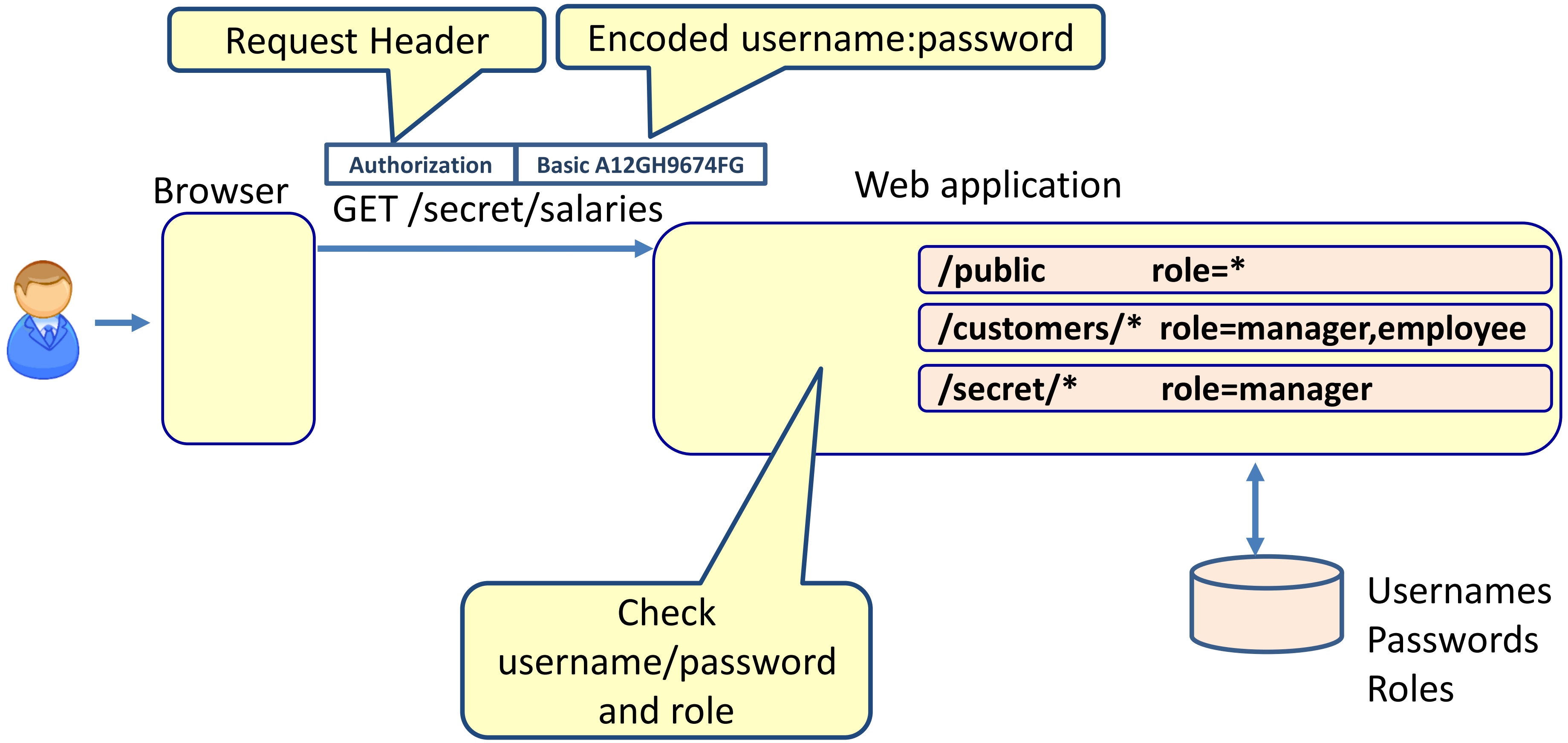


REST SECURITY

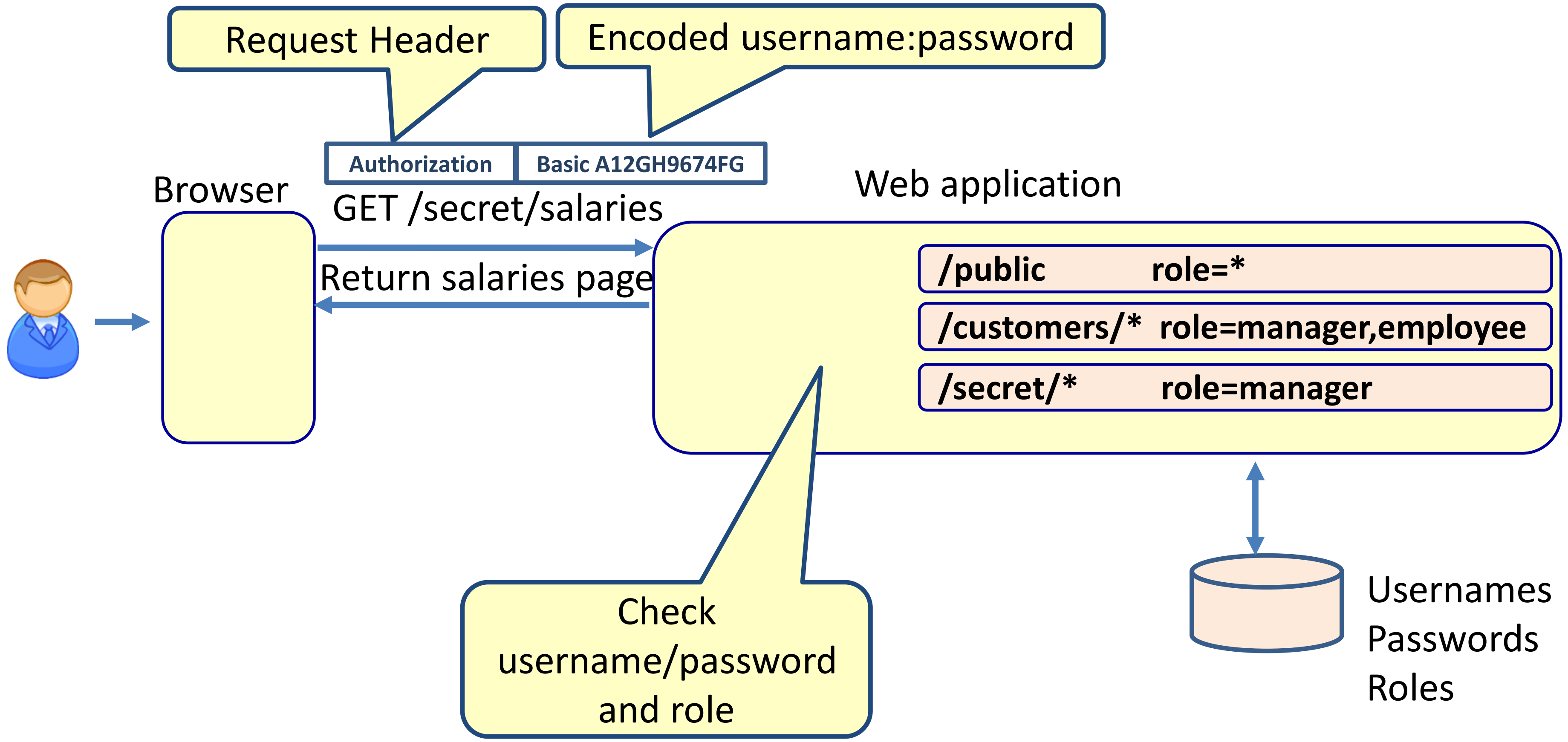
RESTsecurity



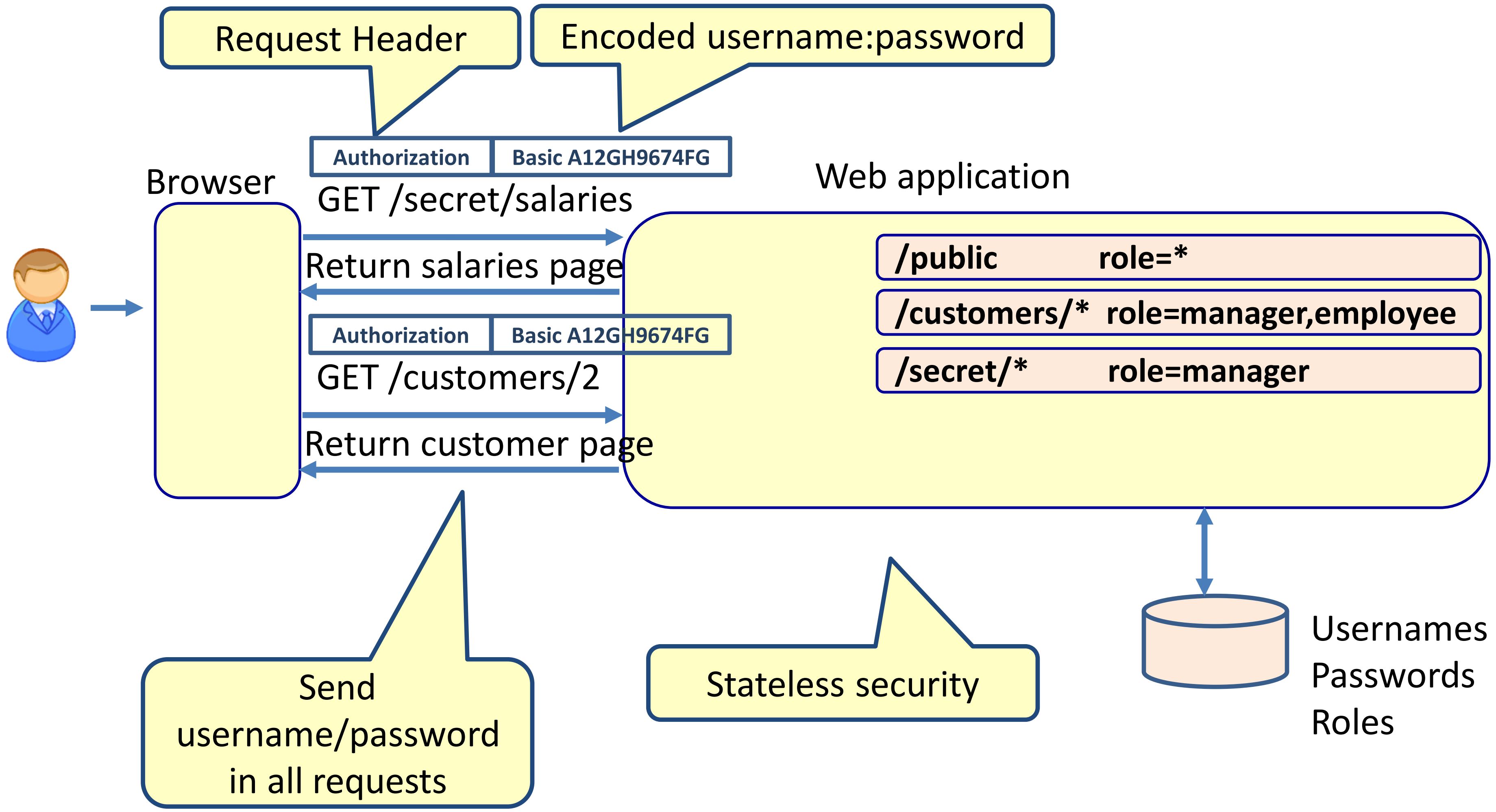
RESTsecurity



RESTsecurity



RESTsecurity



Rest security: Server

```
@RestController
public class MyController {

    @GetMapping("/productinfo")
    public String getProductInfo() {
        return "productinfo";
    }

    @GetMapping("/salaryinfo")
    public String getSalaryInfo() {
        return "salaryinfo";
    }
}
```

Spring security libraries

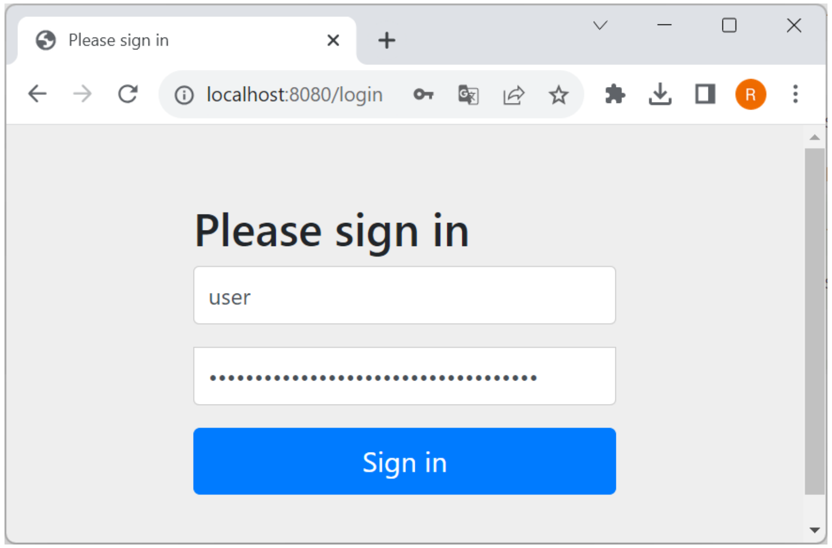
- When Spring Security is on the classpath, the auto-configuration secures all endpoints by default.

```xml
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

- The default user is user
- The default password is given when you start the application

```
Using default security password: fb8d763b-978a-471a-b7fb-f60139fdbc96
```

Spring boot security


Please sign in

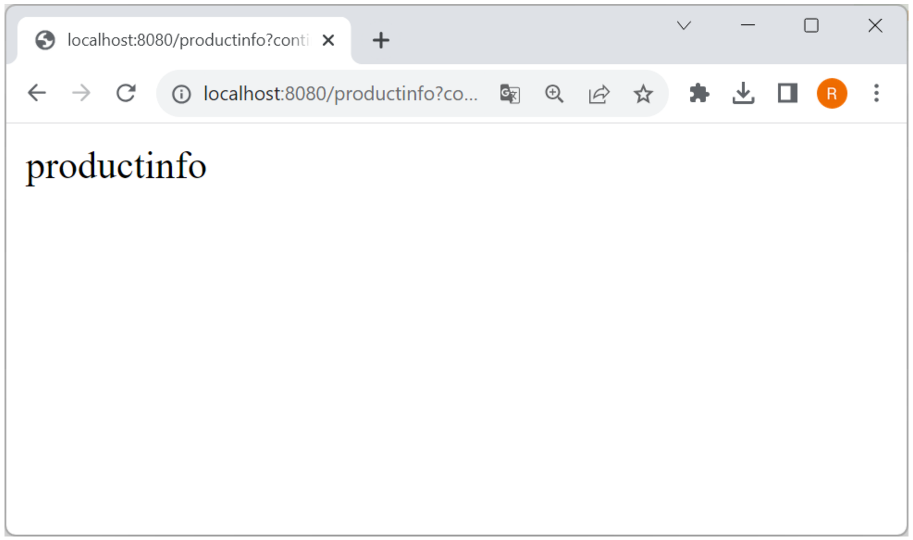
localhost:8080/login

Please sign in

user

••••••••••••••••••••••••••••••••••••••

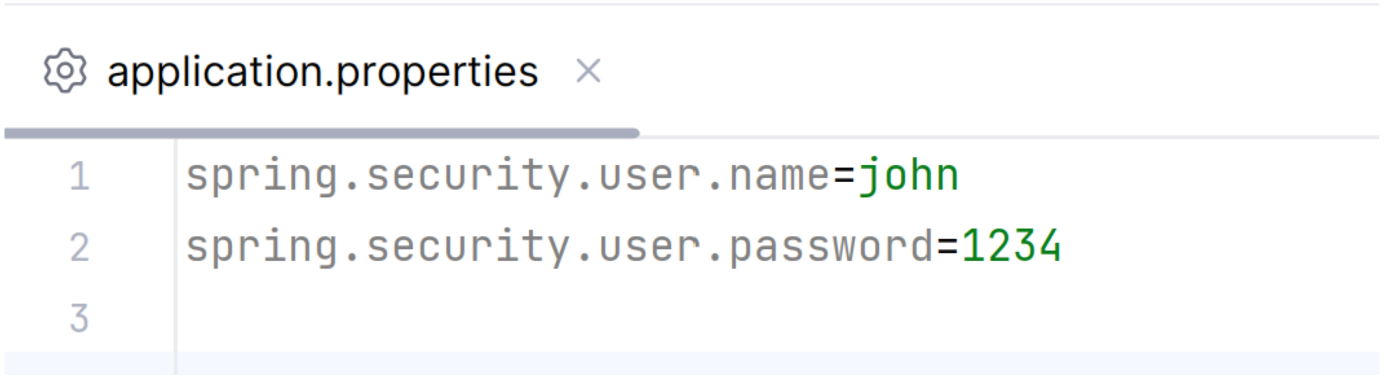
Sign in


localhost:8080/productinfo?conti

localhost:8080/productinfo?co...

productinfo

Set user in application.properties


application.properties

```
spring.security.user.name=john
spring.security.user.password=1234

```

USERNAME/PASSWORD BASED SPRING BOOT SECURITY WITH IN MEMORY USERS

Spring boot security

```
@Configuration
public class UserDetailServiceConfig {

  @Bean
  public UserDetailsService userDetailsService(BCryptPasswordEncoder bCryptPasswordEncoder) {
    InMemoryUserDetailsManager manager = new InMemoryUserDetailsManager();
    manager.createUser(User.withUsername("user")
          .password(bCryptPasswordEncoder.encode("user"))
          .roles("user")
          .build());
    manager.createUser(User.withUsername("admin")
          .password(bCryptPasswordEncoder.encode("admin"))
          .roles("admin")
          .build());
    return manager;
  }

  @Bean
  public BCryptPasswordEncoder bCryptPasswordEncoder() {
    return new BCryptPasswordEncoder();
  }
}
```

Authentication

Add in-memory users

Spring boot security

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
      http.authorizeHttpRequests( authConfig -> {
              authConfig.requestMatchers(HttpMethod.GET, "/productinfo").permitAll();
              authConfig.requestMatchers(HttpMethod.GET, "/salaryinfo").hasRole("admin");
          }).httpBasic(Customizer.withDefaults());

      return http.build();
   }
}
```

Authorization

Method based authorization

```
@RestController
public class MyController {
@GetMapping("/info")
   public ResponseEntity<?> getInfo() {
      return new ResponseEntity<String> ("info", HttpStatus.OK);
   }
   @GetMapping("/user")
   @PreAuthorize("hasRole('user')")
   public ResponseEntity<?> getUserInfo() {
      return new ResponseEntity<String> ("user info", HttpStatus.OK);
   }
   @GetMapping("/admin")
   @PreAuthorize("hasRole('admin')")
   public ResponseEntity<?> getAdminInfo() {
      return new ResponseEntity<String> ("admin info", HttpStatus.OK);
   }
}
```

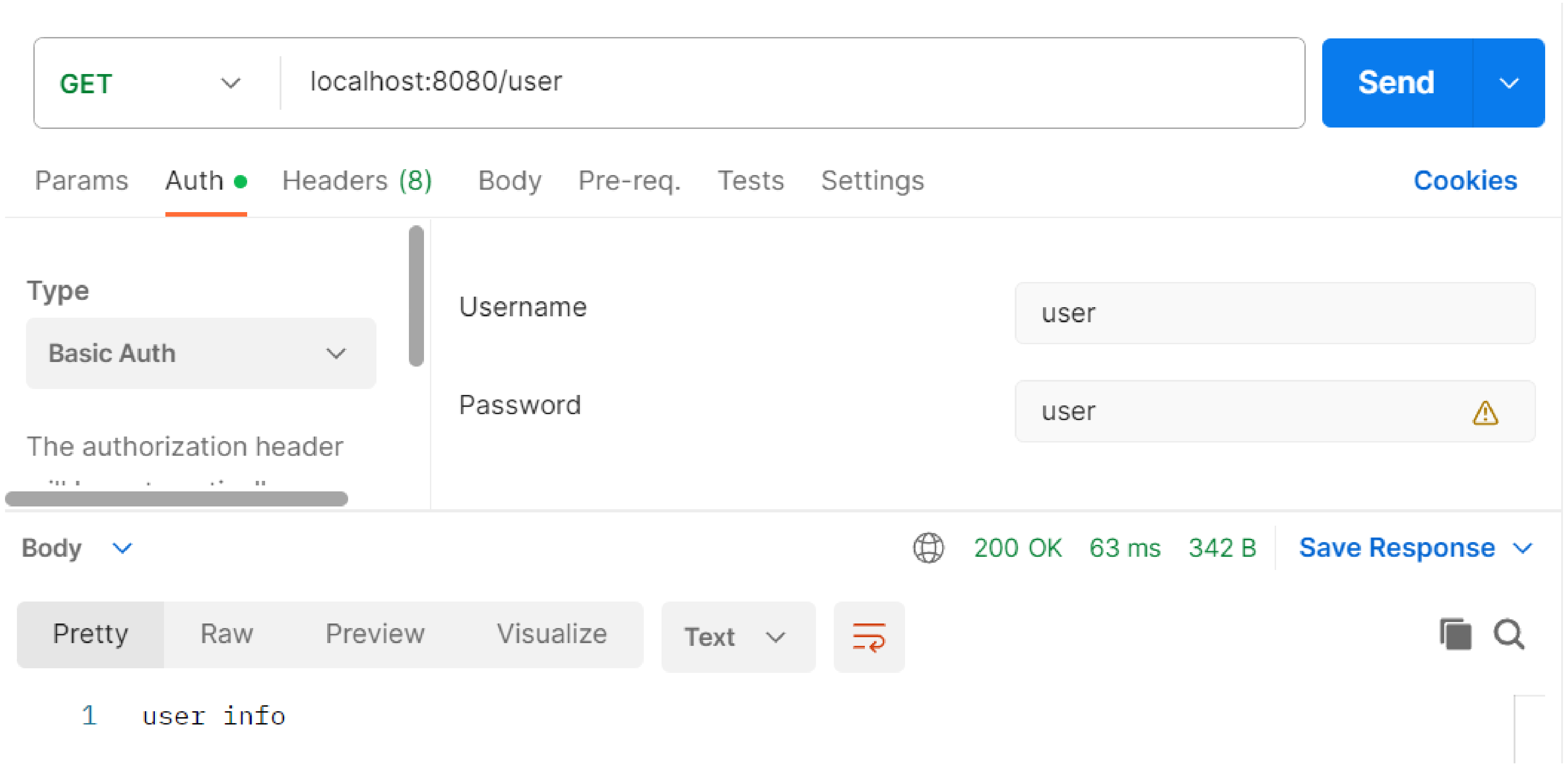
Authorization

Authorization

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfigMethodSecurity {
   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception
{
      http.httpBasic(Customizer.withDefaults());

      return http.build();
   }
}
```

Method security

Postman


GET localhost:8080/user

Type: Basic Auth

The authorization header

Username: user

Password: user

200 OK 63 ms 342 B

```
user info
```

USERNAME/PASSWORD BASED SPRING BOOT SECURITY WITH DATABASE BASED AUTHENTICATION

database based authentication

```
@Document
public class User {
   @Id
   private String username;
   private String password;

   private List<Role> roles = new ArrayList<>();
```

```
public class Role {

   private String role;
```

```
public interface UserRepository extends MongoRepository<User, Integer> {
   User findByUsername(String username);
}
```

database based authentication

```java
public class SecurityUser implements UserDetails {

    private final User user;

    public SecurityUser(User user) {
        this.user = user;
    }

    @Override
    public String getUsername() {return user.getUsername();}
    @Override
    public String getPassword() {return user.getPassword();}

    @Override
    public List<GrantedAuthority> getAuthorities(){
        List<GrantedAuthority> authorities = new ArrayList<>();
        user.getRoles().forEach(role -> authorities.add(new SimpleGrantedAuthority(role.getRole())));
        return authorities;
    }

    @Override
    public boolean isAccountNonExpired() { return true; }
    @Override
    public boolean isAccountNonLocked() {return true;}
    @Override
    public boolean isCredentialsNonExpired() {return true;}
    @Override
    public boolean isEnabled() {return true;}
}
```

database based authentication

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
  @Bean
  public UserDetailsService userDetailsService() {return new UserDetailsServiceImpl();}

  @Bean
  public BCryptPasswordEncoder passwordEncoder() {return new BCryptPasswordEncoder();}

  @Bean
  public AuthenticationProvider authenticationProvider() {
    DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
    authProvider.setUserDetailsService(userDetailsService());
    authProvider.setPasswordEncoder(passwordEncoder());
    return authProvider;
  }

  @Bean
  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        .authorizeHttpRequests( authConfig -> {authConfig
        .requestMatchers(HttpMethod.GET, "/info").permitAll()
        .requestMatchers(HttpMethod.GET, "/user").hasAuthority("user")
        .requestMatchers(HttpMethod.GET, "/admin").hasAuthority("admin");
    }).httpBasic(Customizer.withDefaults());

    return http.build();
  }
}
```

BCryptPasswordEncoder

database based authentication

```
@Configuration
public class UserDetailsServiceImpl implements UserDetailsService {

   @Autowired
   private UserRepository userRepo;

   @Override
   public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
      User user = userRepo.findByUsername(username);
      return new SecurityUser(user);
   }
}
```

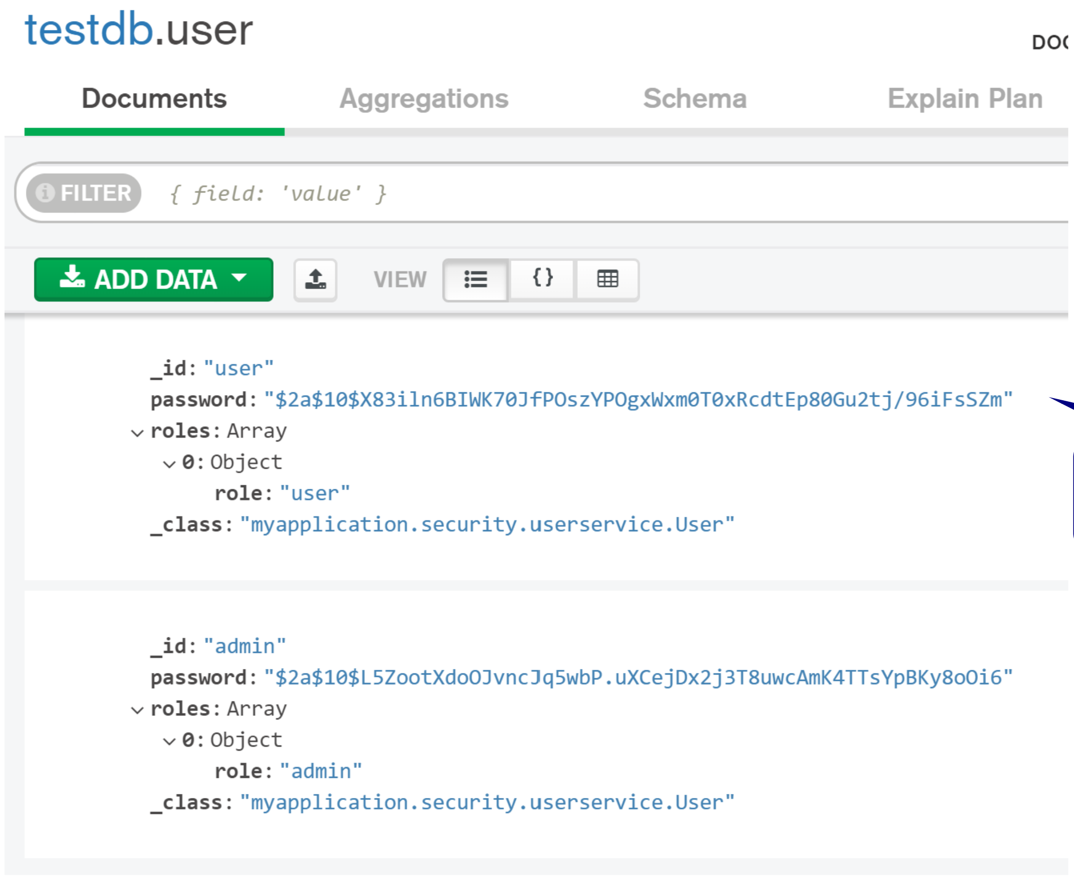
database based authentication

```
@SpringBootApplication
public class Application implements CommandLineRunner {

  @Autowired
  private UserRepository userRepo;
  @Autowired
  private BCryptPasswordEncoder passwordEncoder ;

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

  @Override
  public void run(String... args) throws Exception {
    userRepo.save(new User("user", passwordEncoder.encode("user"), "user"));
    userRepo.save(new User("admin", passwordEncoder.encode("admin"), "admin"));
  }
}
```

database based authentication

```
@RestController
public class MyController {
@GetMapping("/info")
   public ResponseEntity<?> getInfo() {
      return new ResponseEntity<String> ("info", HttpStatus.OK);
   }
   @GetMapping("/user")
   public ResponseEntity<?> getUserInfo() {
      return new ResponseEntity<String> ("user info", HttpStatus.OK);
   }
   @GetMapping("/admin")
   public ResponseEntity<?> getAdminInfo() {
      return new ResponseEntity<String> ("admin info", HttpStatus.OK);
   }
}
```

Users in the database

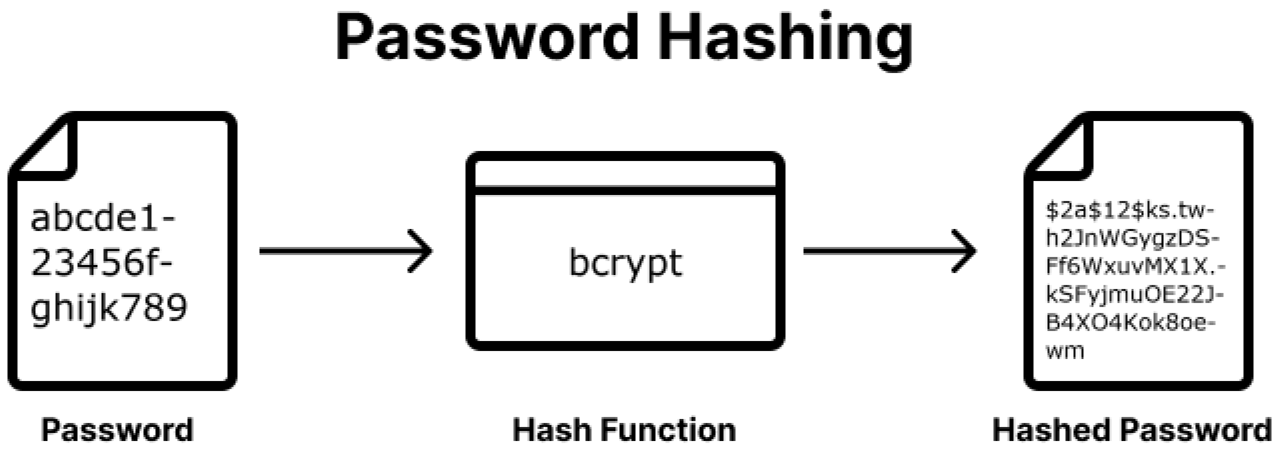


```
  _id: "user"
  password: "$2a$10$X83iln6BIWK70JfPOszYPOgxWxm0T0xRcdtEp80Gu2tj/96iFsSZm"
v roles: Array
  v 0: Object
      role: "user"
  _class: "myapplication.security.userservice.User"
```

```
  _id: "admin"
  password: "$2a$10$L5ZootXdoOJvncJq5wbP.uXCejDx2j3T8uwcAmK4TTsYpBKy8oOi6"
v roles: Array
  v 0: Object
      role: "admin"
  _class: "myapplication.security.userservice.User"
```

Bcryptdecoded password

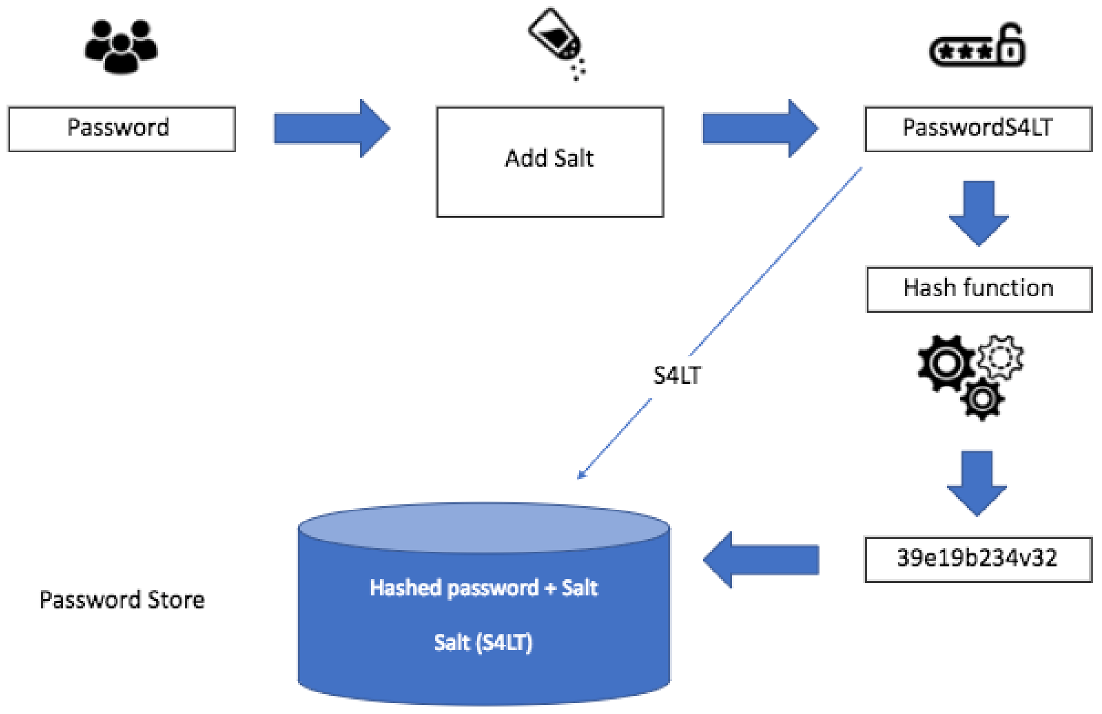
Hashing the password



You cannot retrieve the password from the hashed string

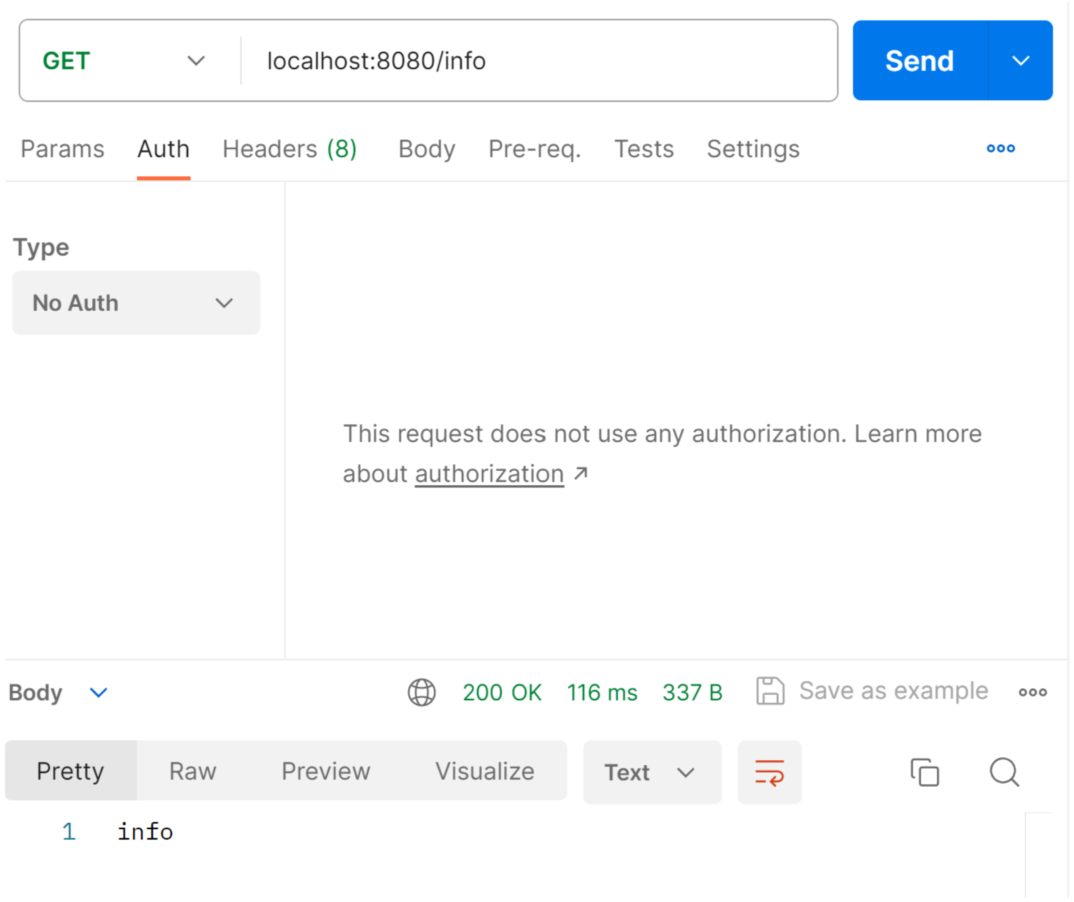
```
hash ("hello") = 3d3929g23994939e83b2ac5b9e29e1b1c19384
hash ("hbllo") = 8dfac912a93f8169afe7dd238f33644939e83b
hash ("blitz") = 83b2afe7dd38f3364493938f33644939d3fg4f
```

Bcrypt encoding



```
hash ("hello")               = a90219323994939e83b2ac5b9e29e1b1c19384
hash ("hello" + "Qxe39dfkdX") = 8dfac912a93f8as98d8sd09sd9s3644939e83b
hash ("hello" + "S399d3x94d") = c9d9d9s7dd38f3364493938f33644939d3fg4f
```

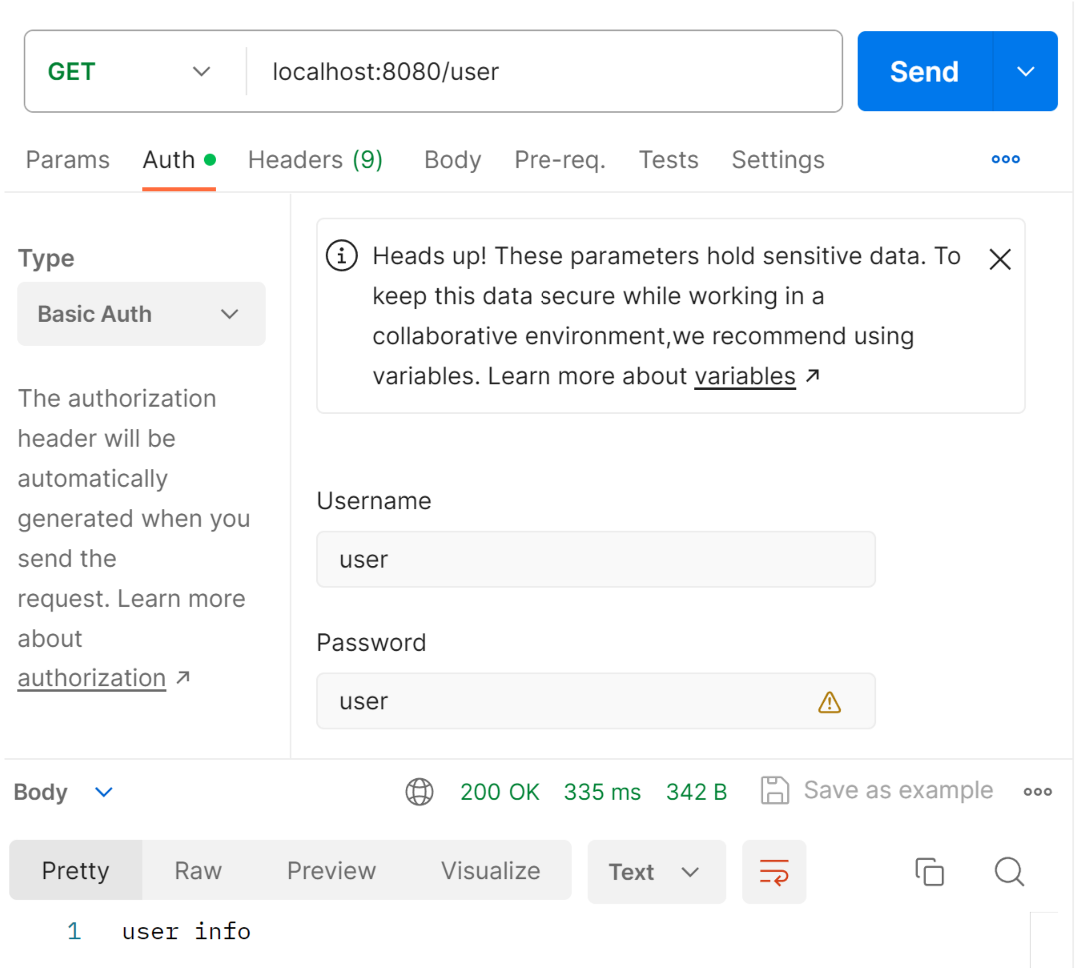
Getting info data



This request does not use any authorization. Learn more about authorization ↗

```
info
```

Getting user data



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

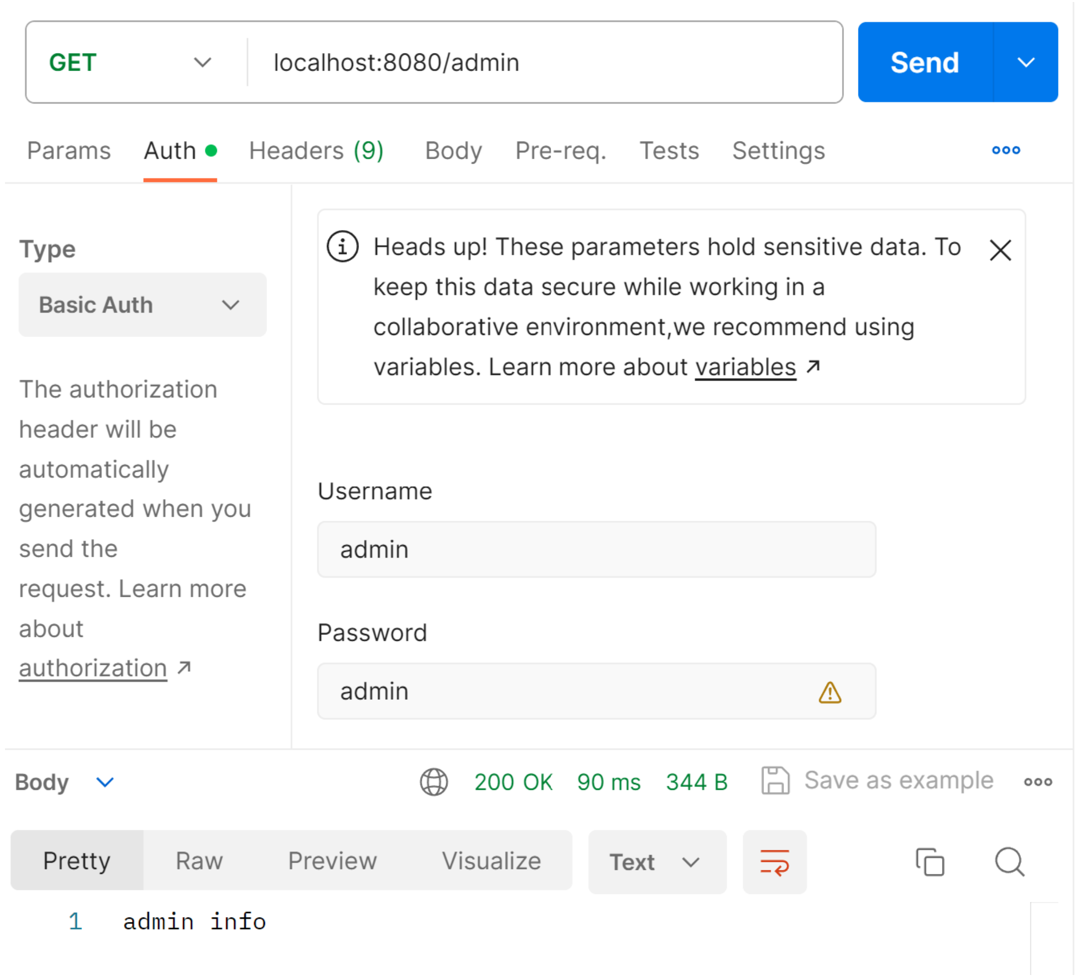
Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment,we recommend using variables. Learn more about variables ↗

Username: user

Password: user

```
user info
```

Get the admin data



GET localhost:8080/admin

Send

Params | Auth ● | Headers (9) | Body | Pre-req. | Tests | Settings

Type

Basic Auth

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 ↗

Username

admin

Password

admin

Body 200 OK 90 ms 344 B Save as example

Pretty | Raw | Preview | Visualize | Text

```
admin info
```

Main point

- To make an application secure we have to configure both authentication and authorization using role-based security. *When one operates from the level of the Unified Field, one has access to all intelligence of creation.*

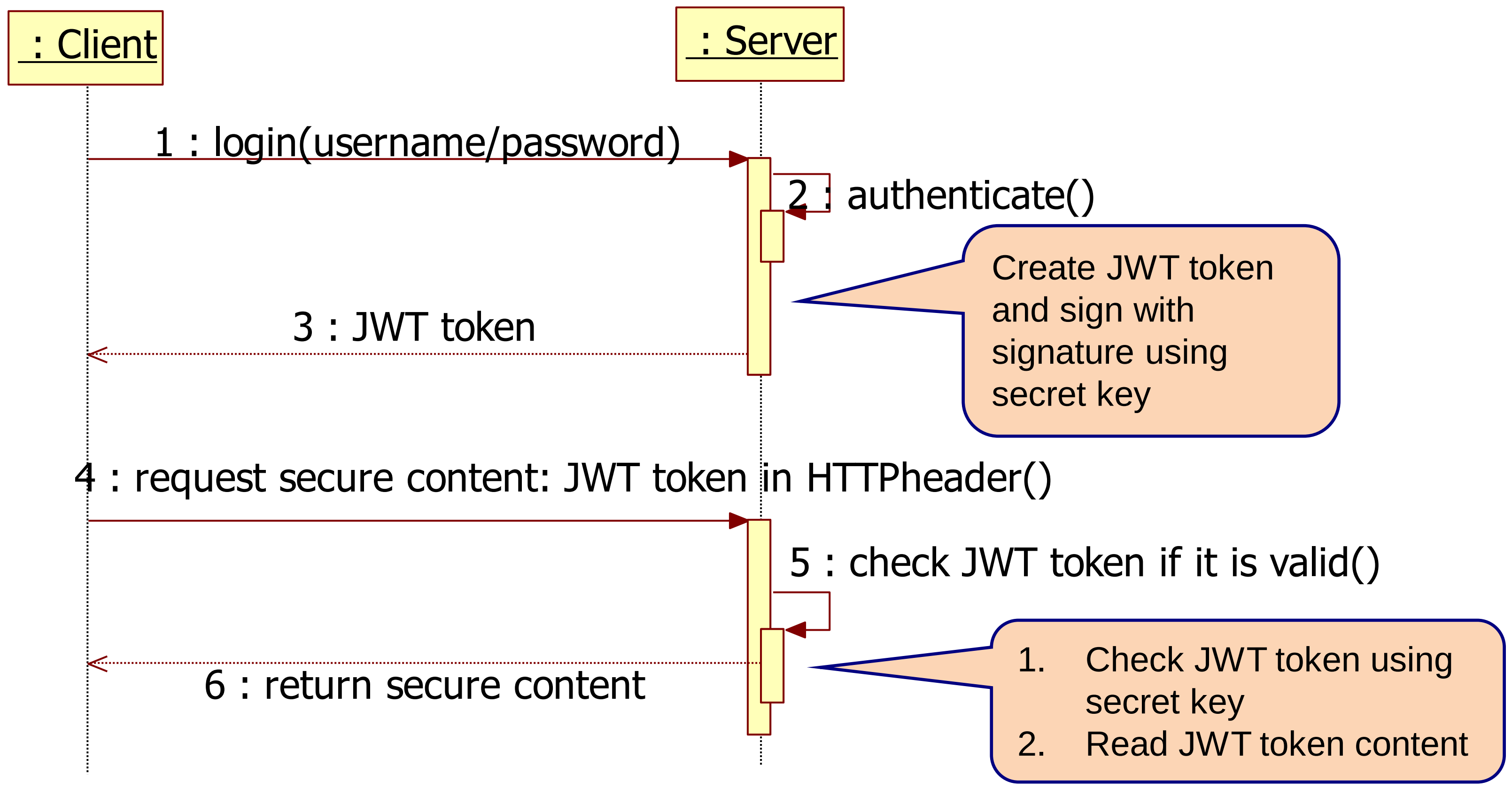
Problem with sending encoded username/password in header

- NOT secure!
- Solution: JWT

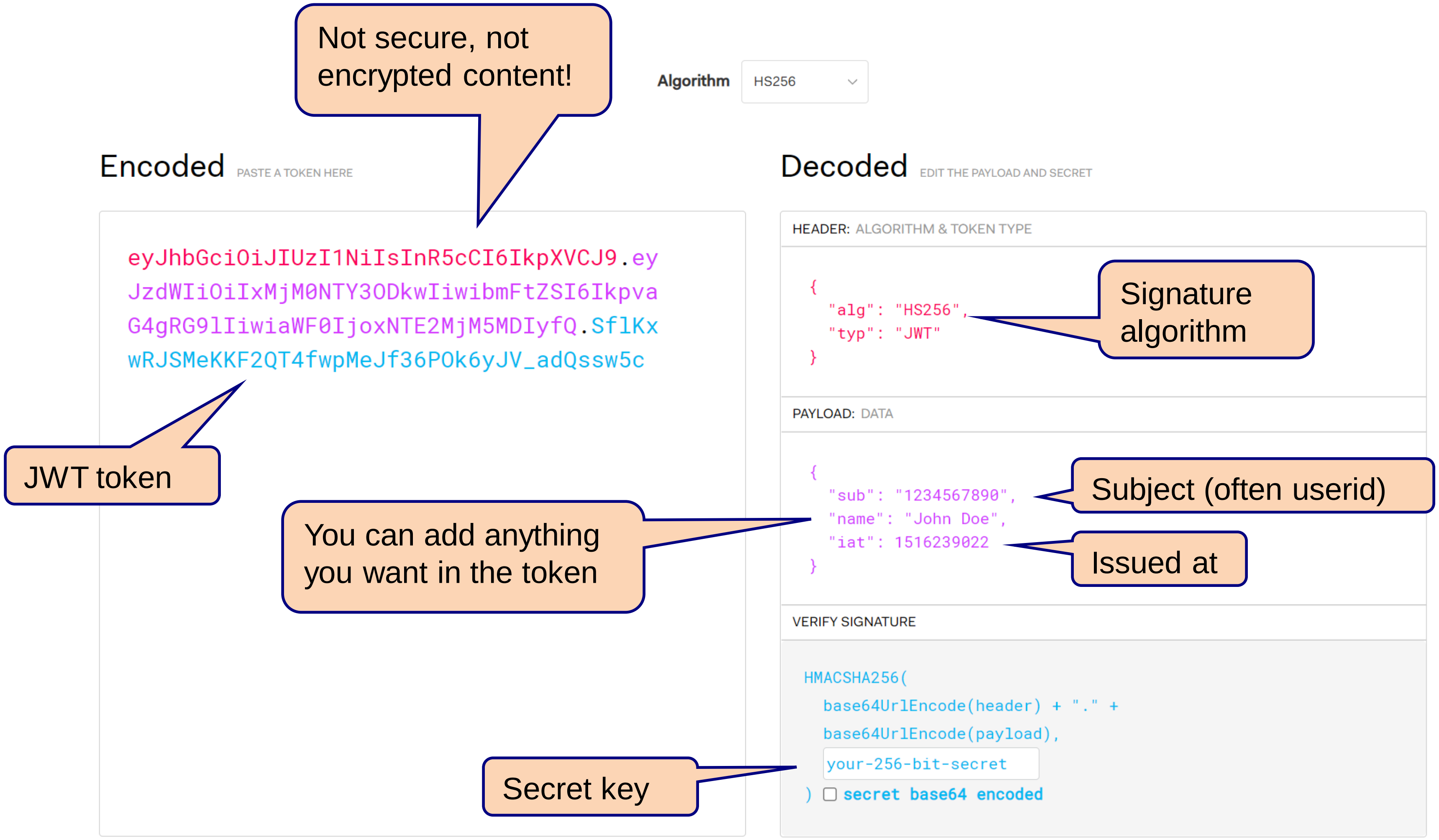
JWT

JWT

- JSON Web Token


: Client
: Server
1 : login(username/password)
2 : authenticate()
Create JWT token and sign with signature using secret key
3 : JWT token
4 : request secure content: JWT token in HTTPheader()
5 : check JWT token if it is valid()
1. Check JWT token using secret key
2. Read JWT token content
6 : return secure content

jwt.io



Not secure, not encrypted content!

Algorithm HS256

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

JWT token

You can add anything you want in the token

Secret key

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

Signature algorithm

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

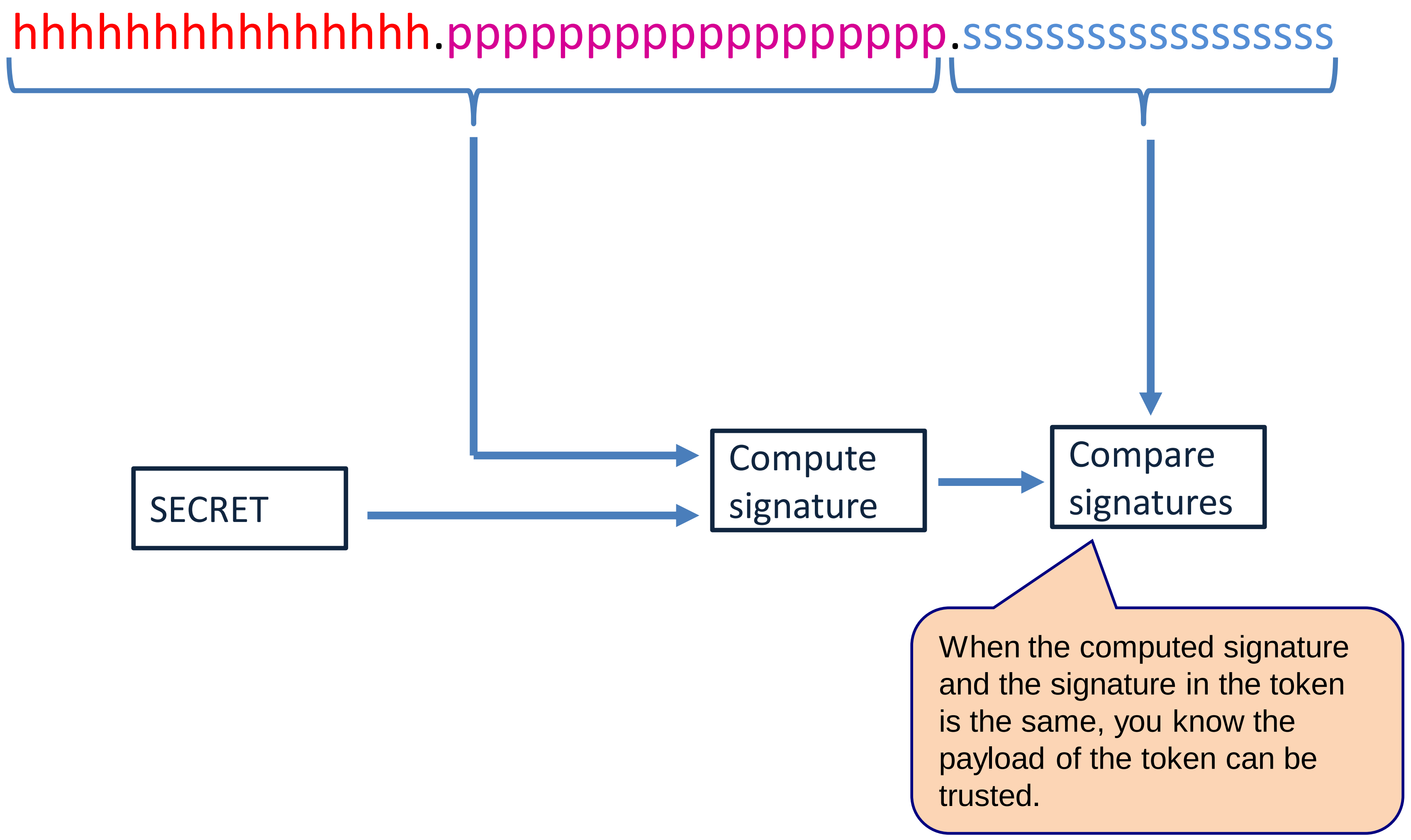
Subject (often userid)

Issued at

VERIFY SIGNATURE

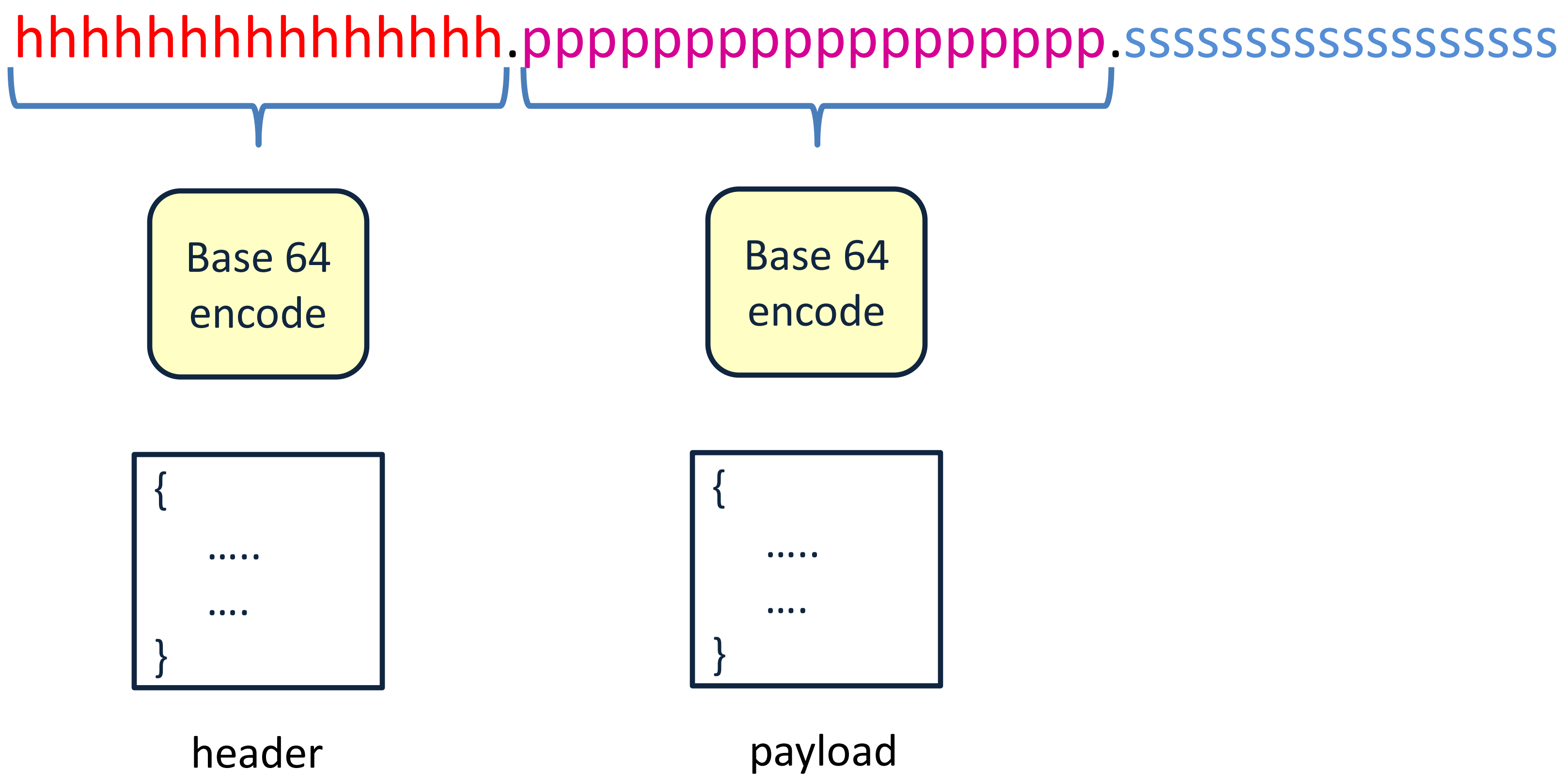
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Check JWT signature



When the computed signature and the signature in the token is the same, you know the payload of the token can be trusted.

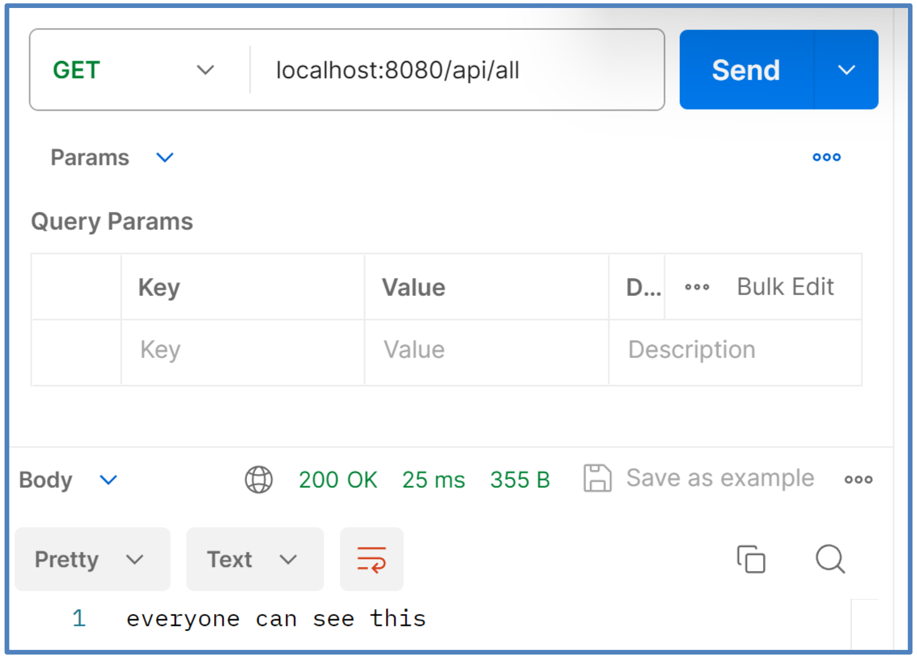
Get JWT token content



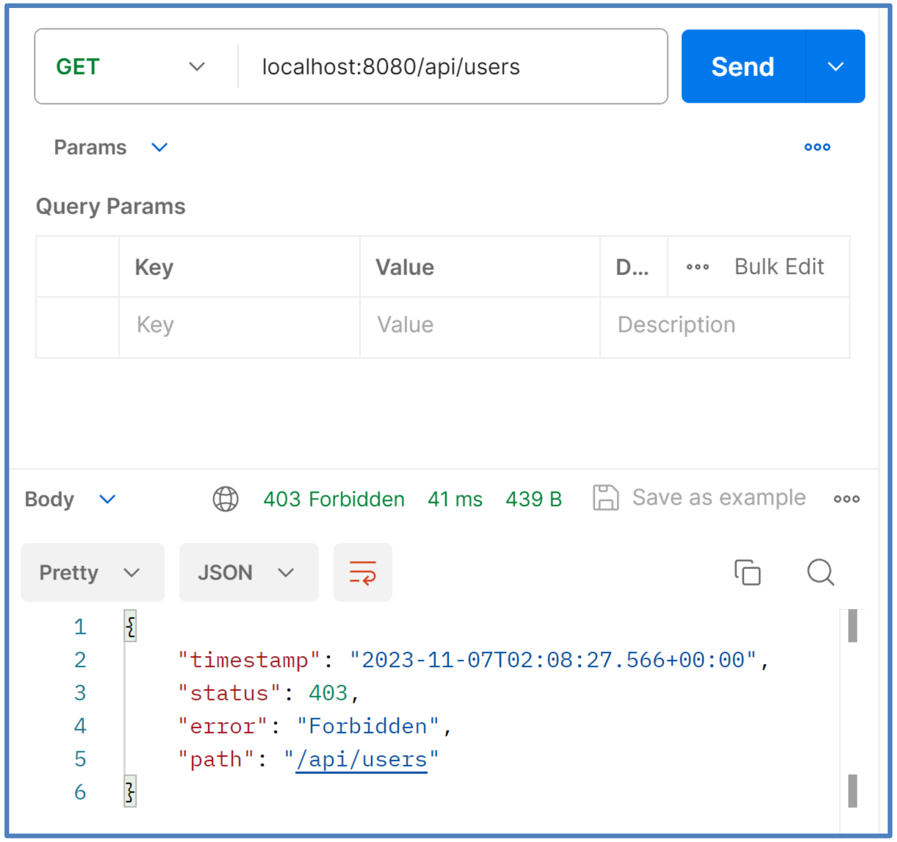
JWT example

```
@RestController
@RequestMapping("/api/v1/test")
public class MyController {

  @GetMapping("/all")
  public String allEndPoint() {
     return "everyone can see this";
  }

  @GetMapping("/users")
  @PreAuthorize("hasRole('USER')")
  public String usersEndPoint() {
    return "ONLY users can see this";
  }

  @GetMapping("/admins")
  @PreAuthorize("hasRole('ADMIN')")
  public String adminsEndPoint() {
    return "ONLY admins can see this";
  }
}
```

JWT example



```
everyone can see this
```

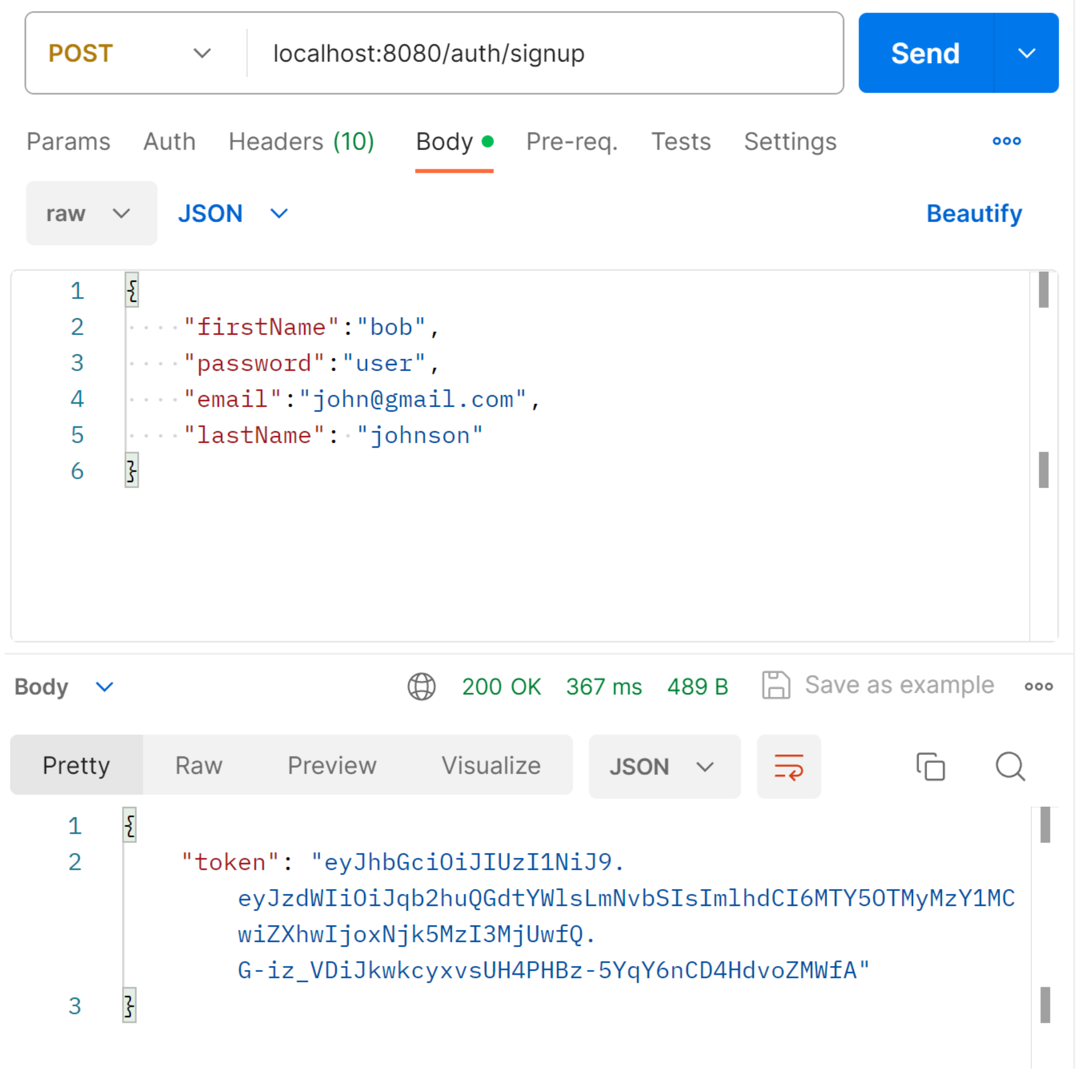


```
{
    "timestamp": "2023-11-07T02:08:27.566+00:00",
    "status": 403,
    "error": "Forbidden",
    "path": "/api/users"
}
```

JWT example

```
@RestController
@RequestMapping("/auth")
public class AuthenticationController {

   @Autowired
   private  AuthenticationService authenticationService;

   @PostMapping("/signup")
   public JwtAuthenticationResponse signup(@RequestBody SignUpRequest request) {
      return authenticationService.signup(request);
   }

   @PostMapping("/signin")
   public JwtAuthenticationResponse signin(@RequestBody SignInRequest request) {
      return authenticationService.signin(request);
   }
}
```

Signup user



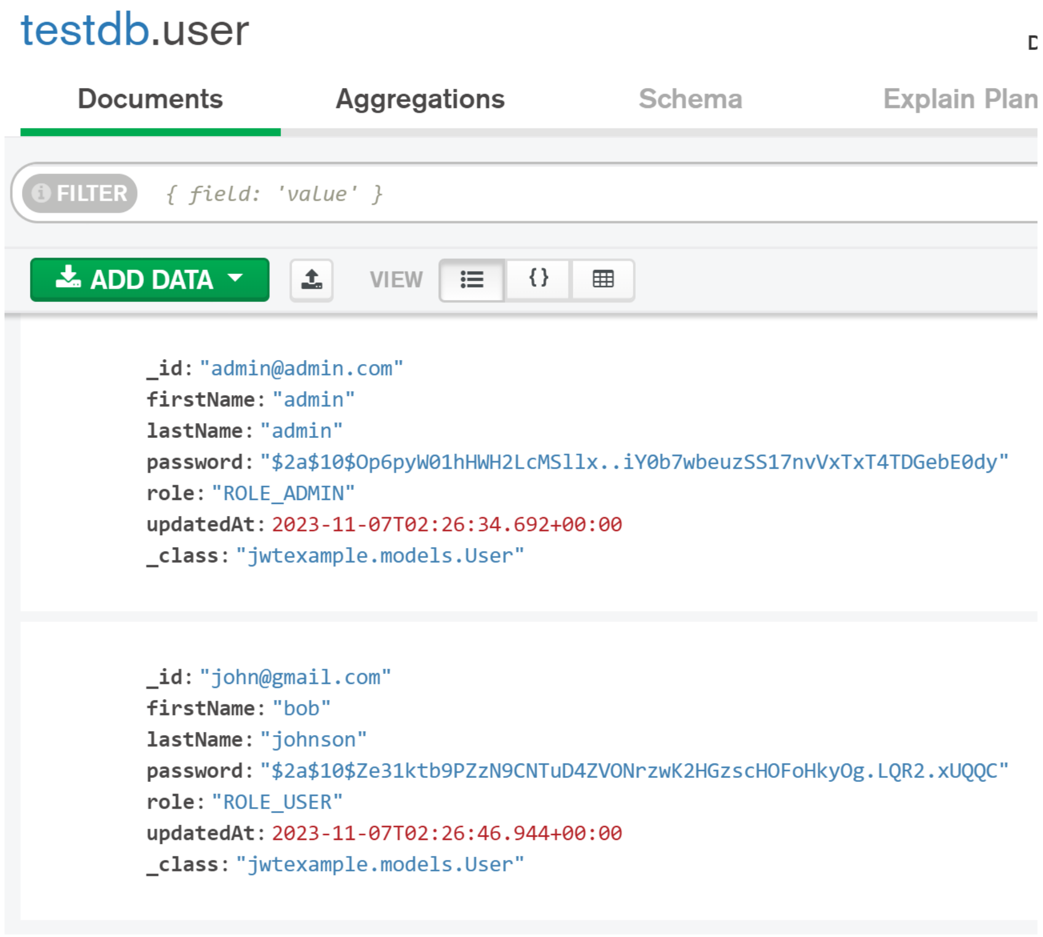
localhost:8080/auth/signup

```
{
    "firstName":"bob",
    "password":"user",
    "email":"john@gmail.com",
    "lastName": "johnson"
}
```

200 OK 367 ms 489 B

```
{
    "token": "eyJhbGciOiJIUzI1NiJ9.
        eyJzdWIiOiJqb2huQGdtYWlsLmNvbSIsImlhdCI6MTY5OTMyMzY1MC
        wiZXhwIjoxNjk5MzI3MjUwfQ.
        G-iz_VDiJkwkcyxvsUH4PHBz-5YqY6nCD4HdvoZMWfA"
}
```

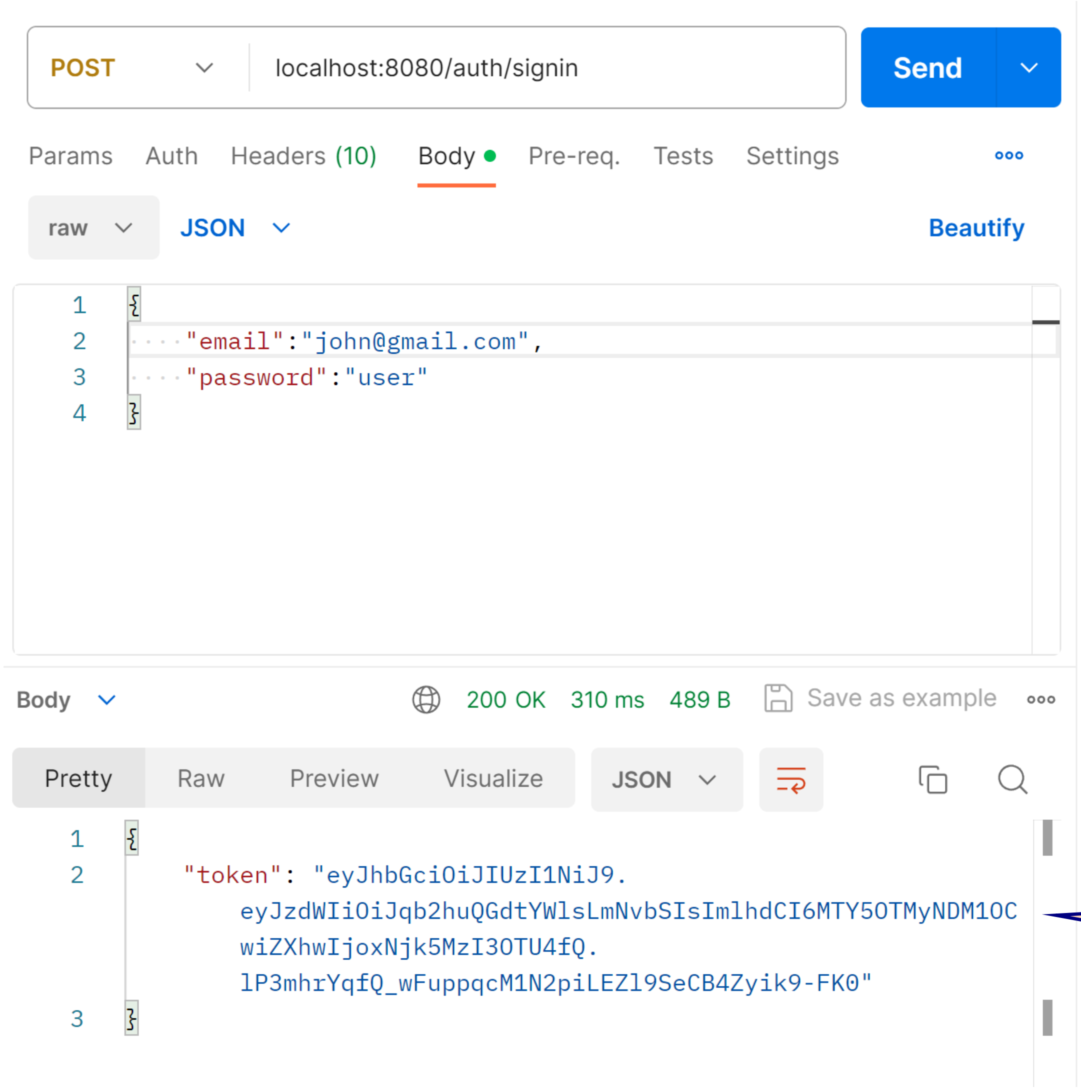
database



```
_id: "admin@admin.com"
firstName: "admin"
lastName: "admin"
password: "$2a$10$Op6pyW01hHWH2LcMSllx..iY0b7wbeuzSS17nvVxTxT4TDGebE0dy"
role: "ROLE_ADMIN"
updatedAt: 2023-11-07T02:26:34.692+00:00
_class: "jwtexample.models.User"
```

```
_id: "john@gmail.com"
firstName: "bob"
lastName: "johnson"
password: "$2a$10$Ze31ktb9PZzN9CNTuD4ZVONrzwK2HGzscHOFoHkyOg.LQR2.xUQQC"
role: "ROLE_USER"
updatedAt: 2023-11-07T02:26:46.944+00:00
_class: "jwtexample.models.User"
```

Signin user



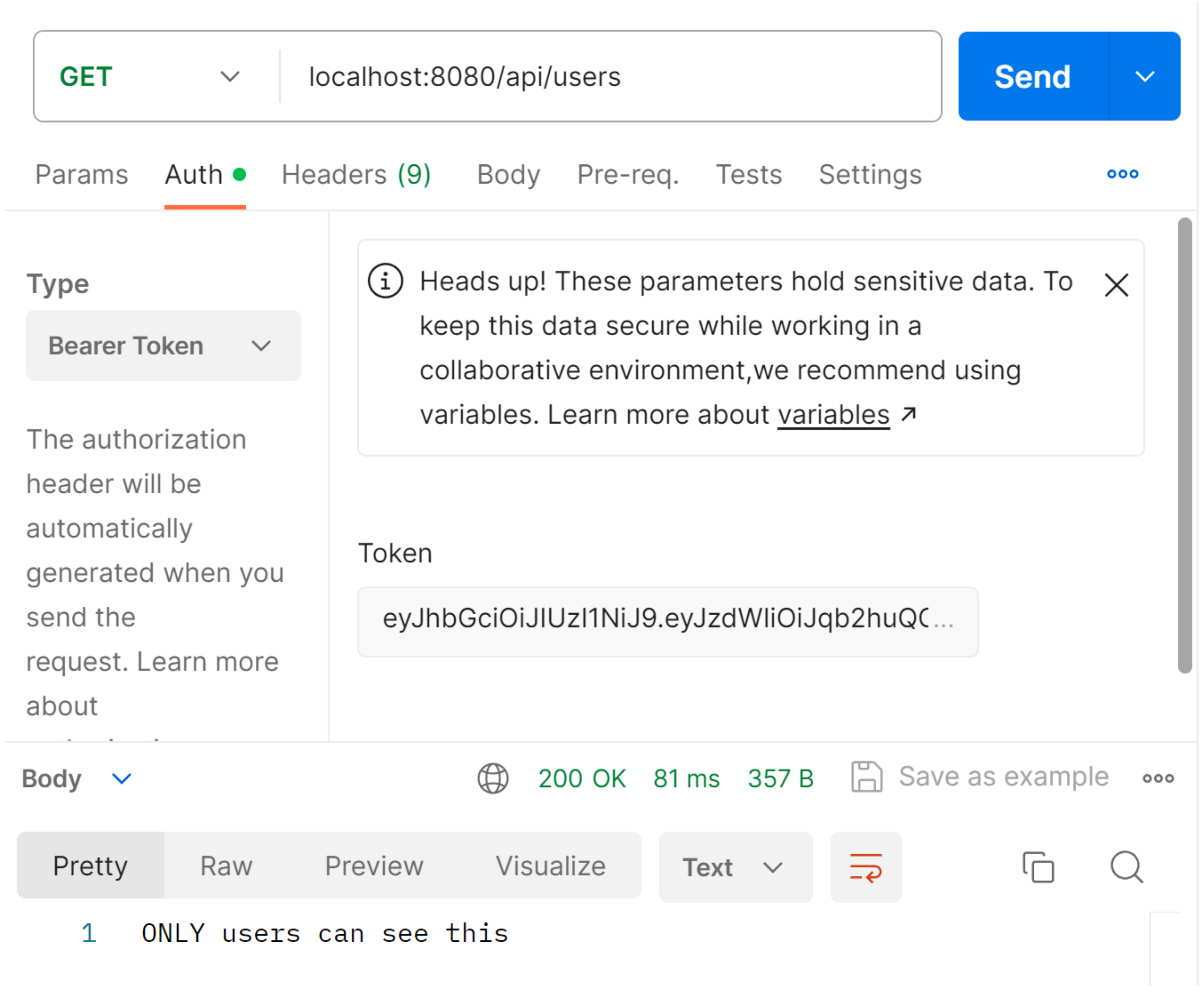
localhost:8080/auth/signin

```
{
    "email":"john@gmail.com",
    "password":"user"
}
```

```
{
    "token": "eyJhbGciOiJIUzI1NiJ9.
        eyJzdWIiOiJqb2huQGdtYWlsLmNvbSIsImlhdCI6MTY5OTMyNDM1OC
        wiZXhwIjoxNjk5MzI3OTU4fQ.
        lP3mhrYqfQ_wFuppqcM1N2piLEZl9SeCB4Zyik9-FK0"
}
```

Token for user

Get users info



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

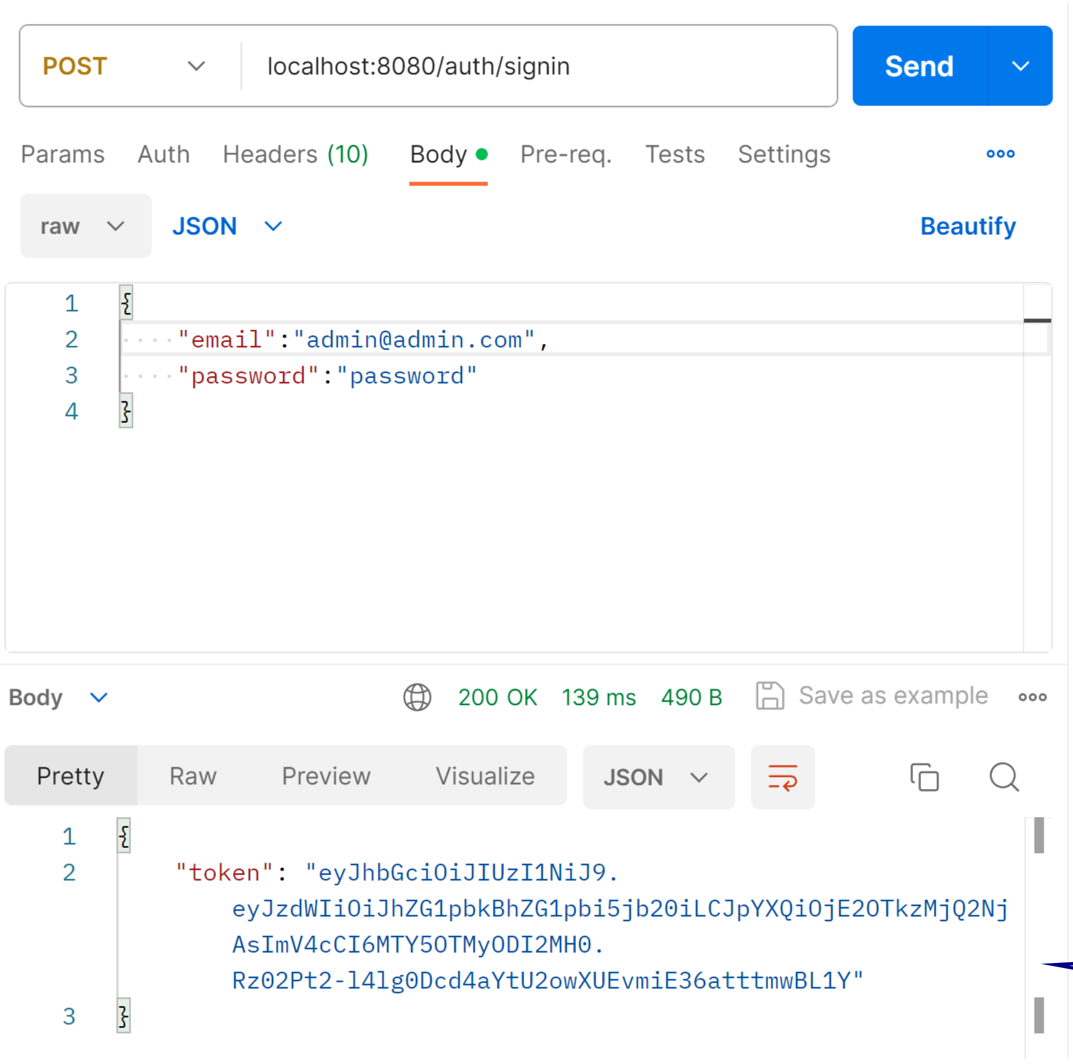
Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment,we recommend using variables. Learn more about variables ↗

Token

eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJqb2huQC...

```
ONLY users can see this
```

Signin admin



localhost:8080/auth/signin

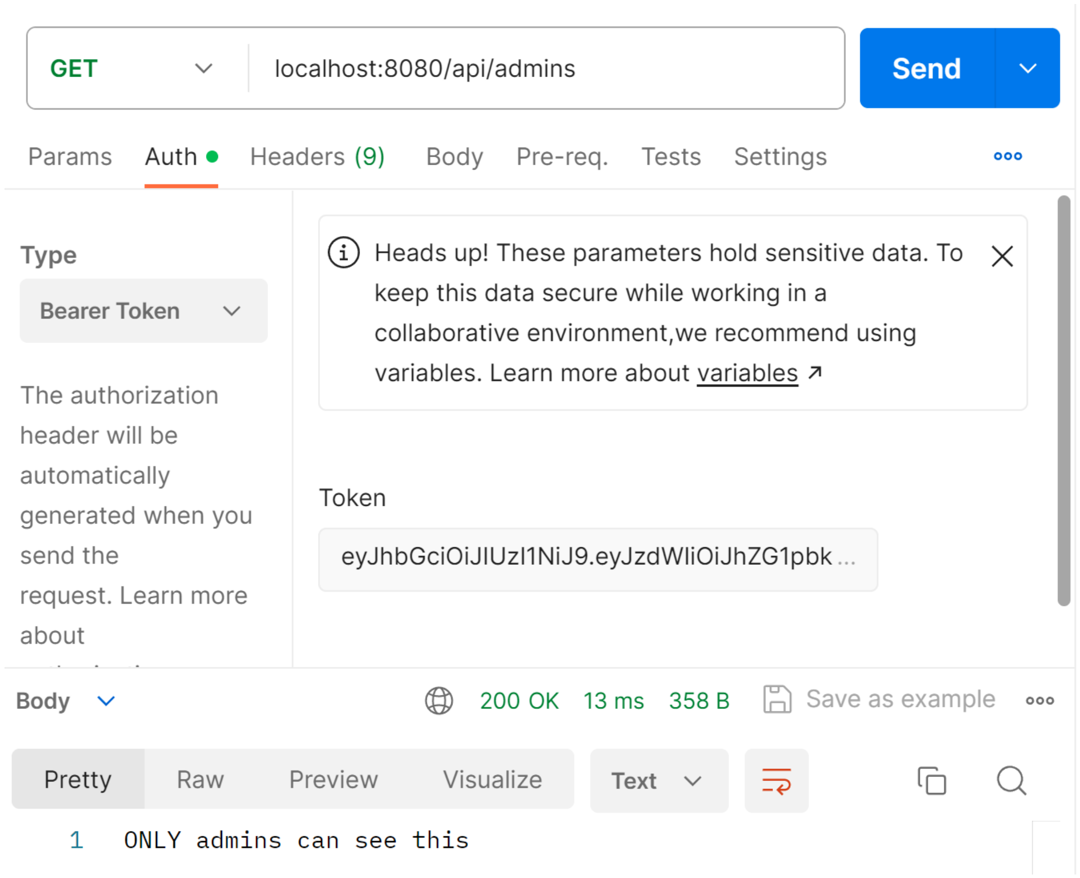
```
{
    "email":"admin@admin.com",
    "password":"password"
}
```

200 OK 139 ms 490 B

```
{
    "token": "eyJhbGciOiJIUzI1NiJ9.
        eyJzdWIiOiJhZG1pbkBhZG1pbi5jb20iLCJpYXQiOjE2OTkzMjQ2Nj
        AsImV4cCI6MTY5OTMyODI2MH0.
        Rz02Pt2-l4lg0Dcd4aYtU2owXUEvmiE36atttmwBL1Y"
}
```

Token for admin

Get admins info



GET localhost:8080/api/admins

Type: Bearer Token

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

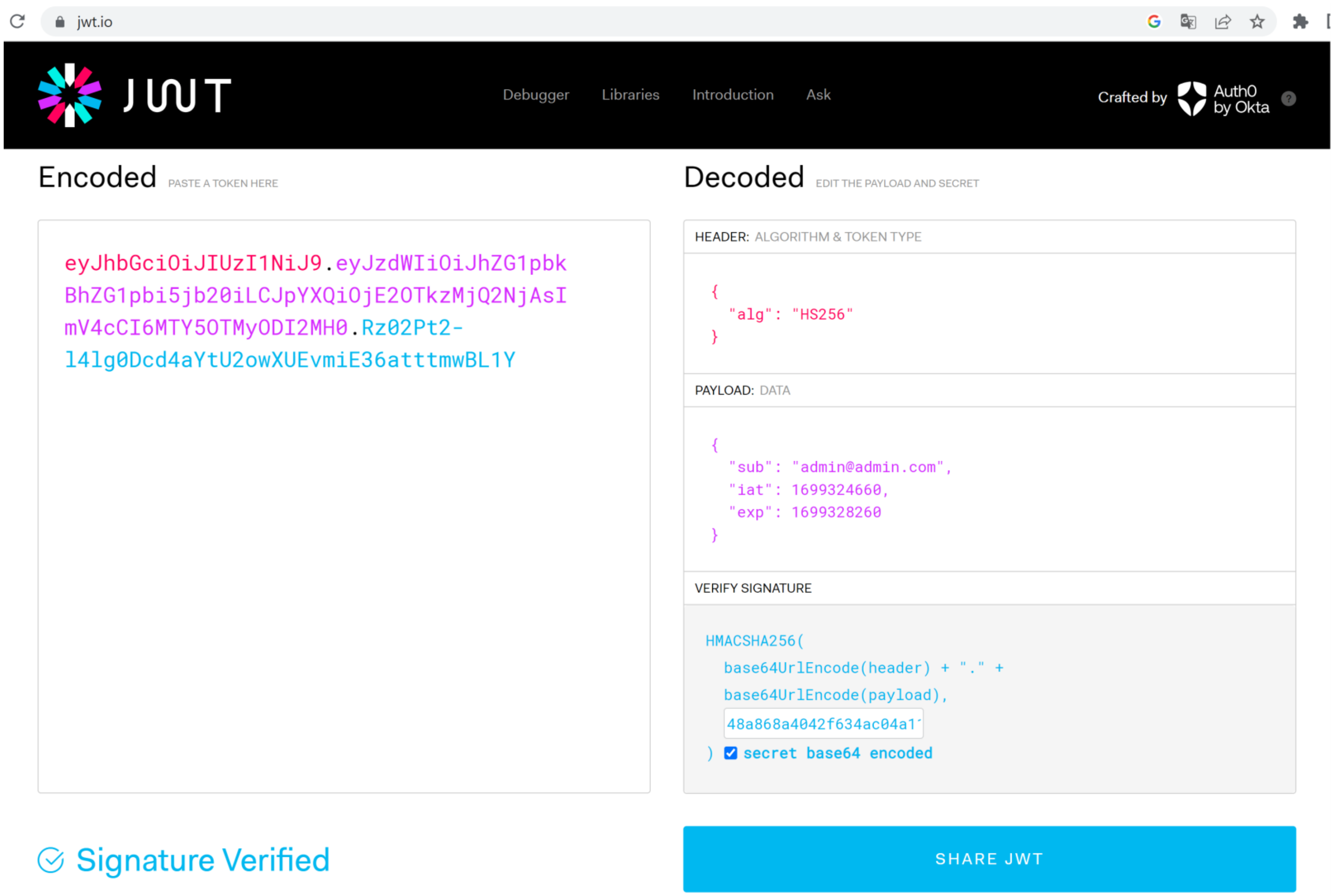
Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment,we recommend using variables. Learn more about variables ↗

Token: eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbk ...

Body — 200 OK 13 ms 358 B

```
ONLY admins can see this
```

JWT token



jwt.io

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbk
BhZG1pbi5jb20iLCJpYXQiOjE2OTkzMjQ2NjAsI
mV4cCI6MTY5OTMyODI2MH0.Rz02Pt2-
l4lg0Dcd4aYtU2owXUEvmiE36atttmwBL1Y
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256"
}
```

PAYLOAD: DATA

```
{
  "sub": "admin@admin.com",
  "iat": 1699324660,
  "exp": 1699328260
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  48a868a4042f634ac04a1
) ☑ secret base64 encoded
```

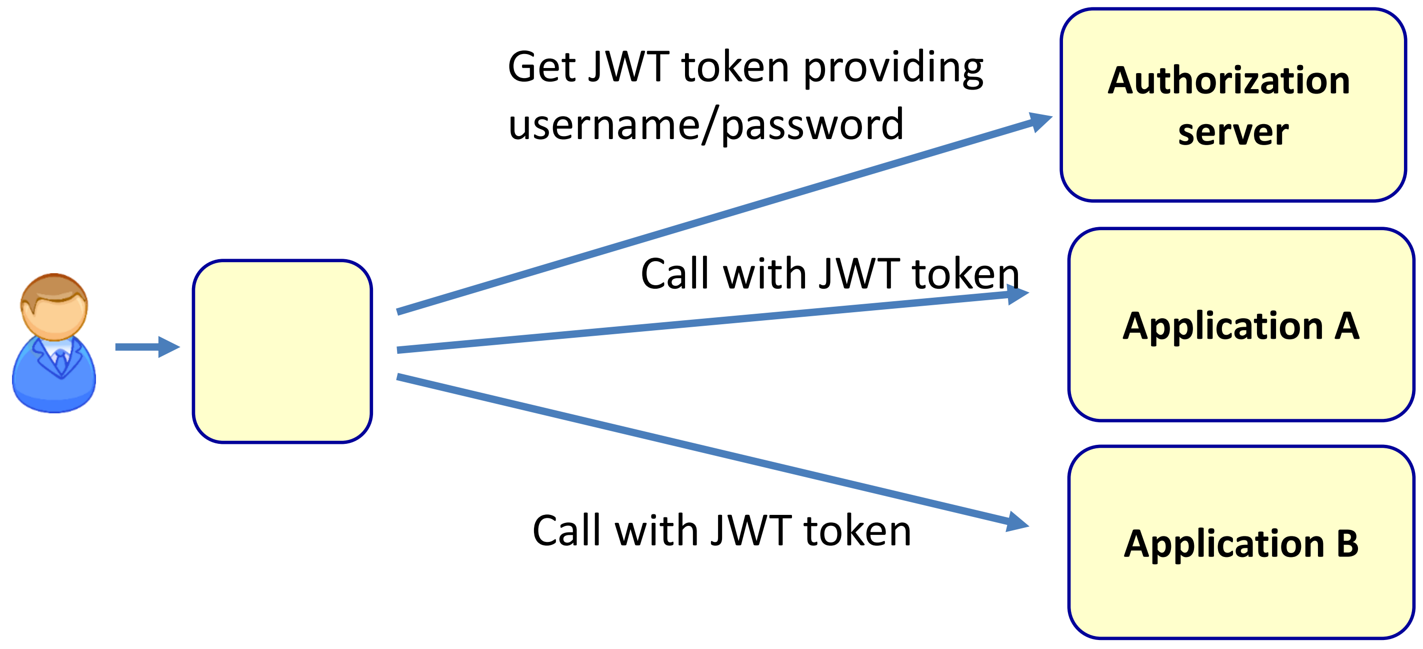
Signature Verified

SHARE JWT

JWT

- Never place secure content in a JWT token
  - JWT token is only signed, not encrypted
- What if someone steals the JWT token
  - Use token expiration
  - Server maintains list of blacklisted JWT’s

OAuth2


Get JWT token providing username/password
Authorization server
Call with JWT token
Application A
Call with JWT token
Application B

Main point

- REST endpoints can be made secure with JWT tokens . *The TM technique is the key to transcend and access pure consciousness.*

Connecting the parts of knowledge with the wholeness of knowledge

1. Spring security makes implementing security simple by defining authentication and authorization in one simple configuration file.
2. To make REST endpoints secure you need token-based security

3. **Transcendental consciousness** is the field of all intelligence that is accessible to everyone.
4. **Wholeness moving within itself:** In Unity Consciousness one realizes that all outer creation are just expressions of one’s own Self