

CS544

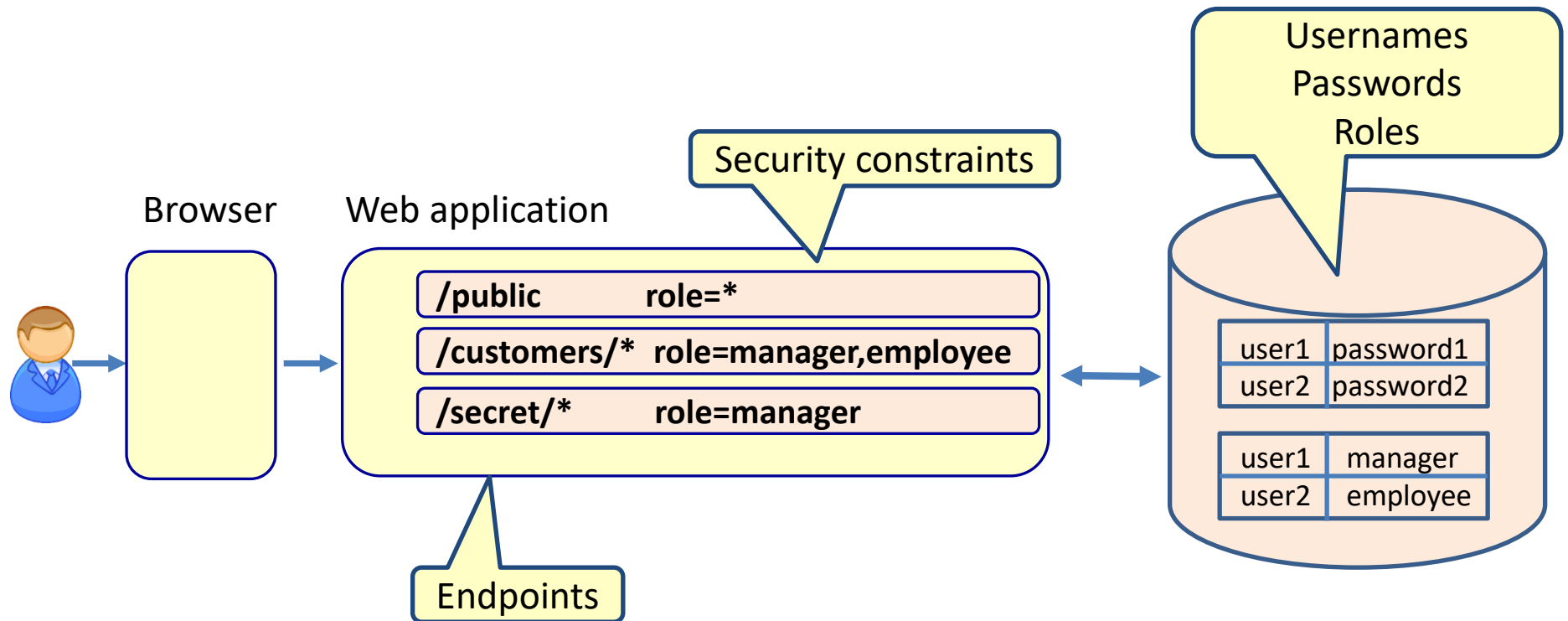
Lesson 14

SECURITY

Security

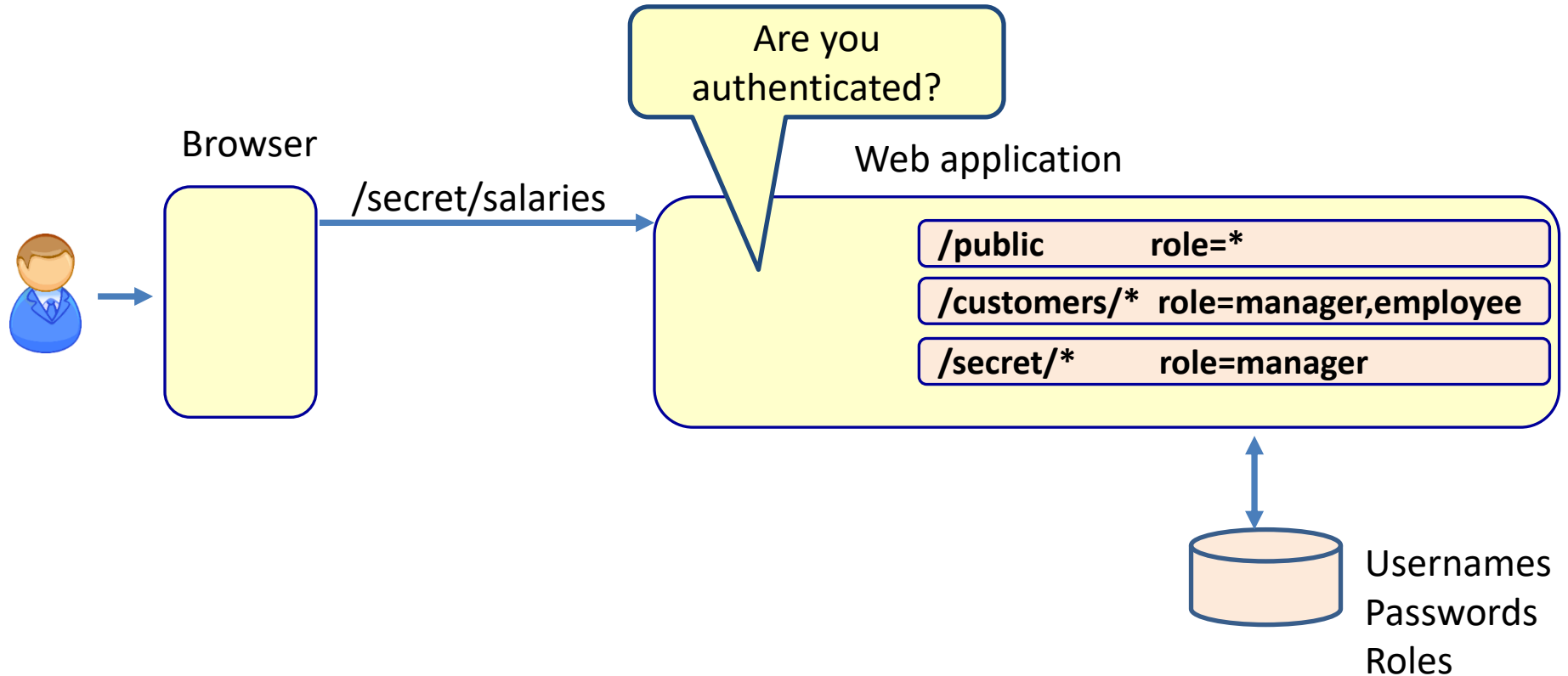
- Authentication
 - Are you who you say you are?
 - Username - password
- Authorization
 - Now that I know who you are, what are you allowed to do?

Role based security

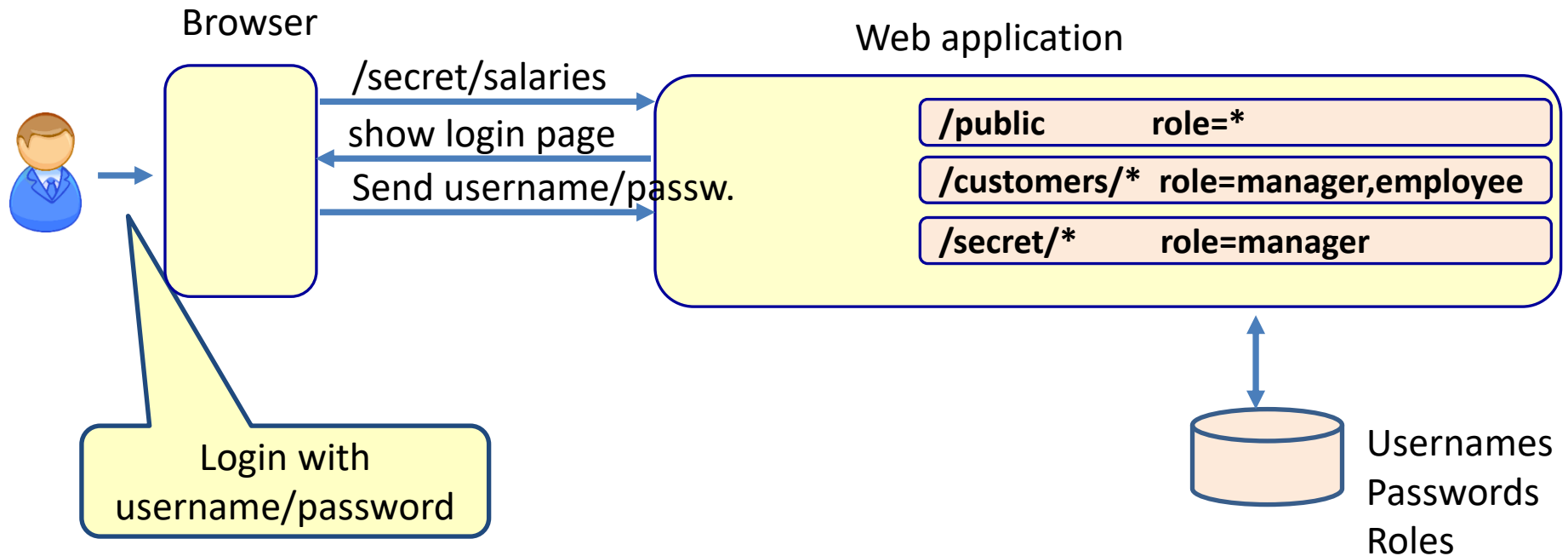


WEB APPLICATION SECURITY

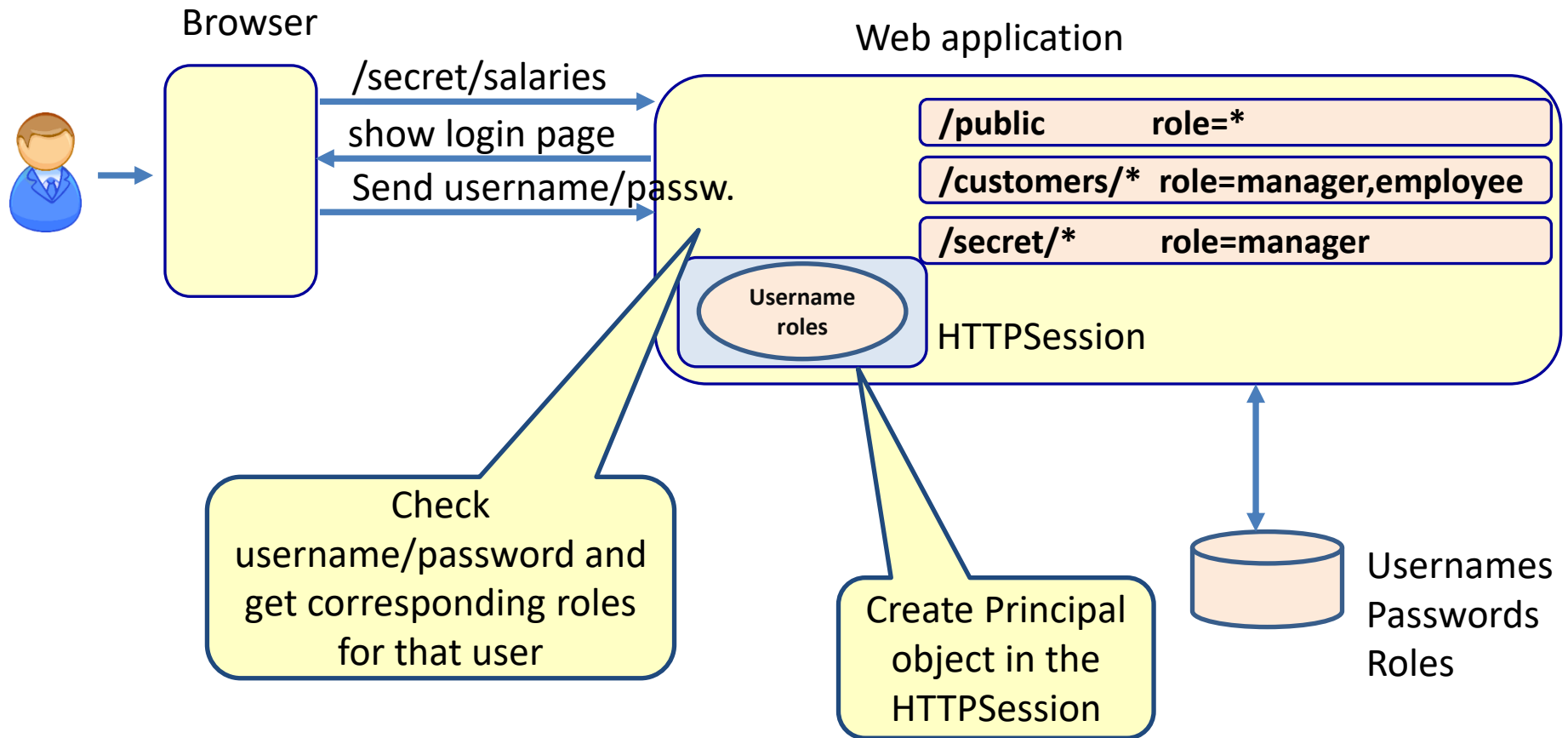
Web application security



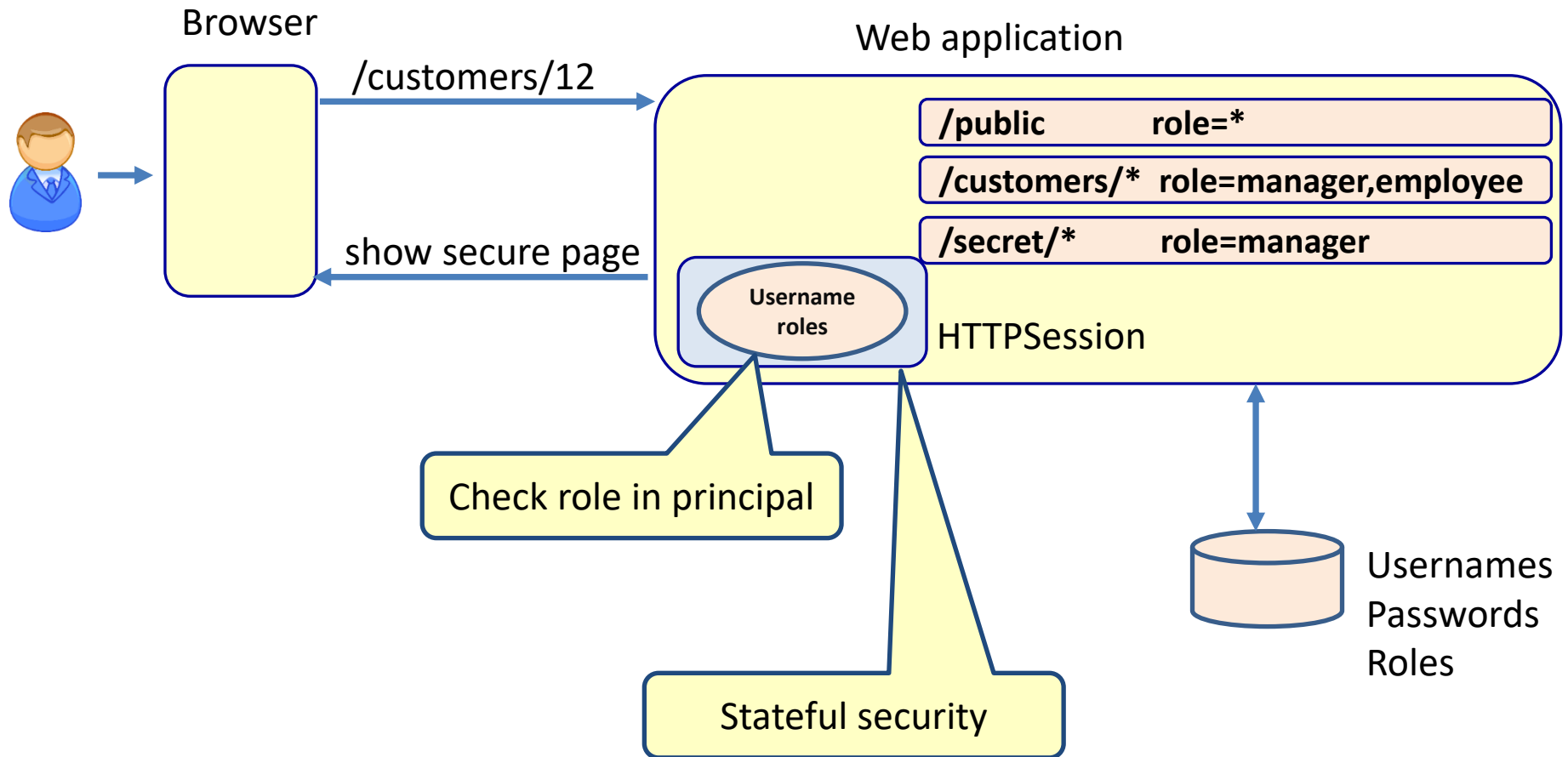
Web security



Web security

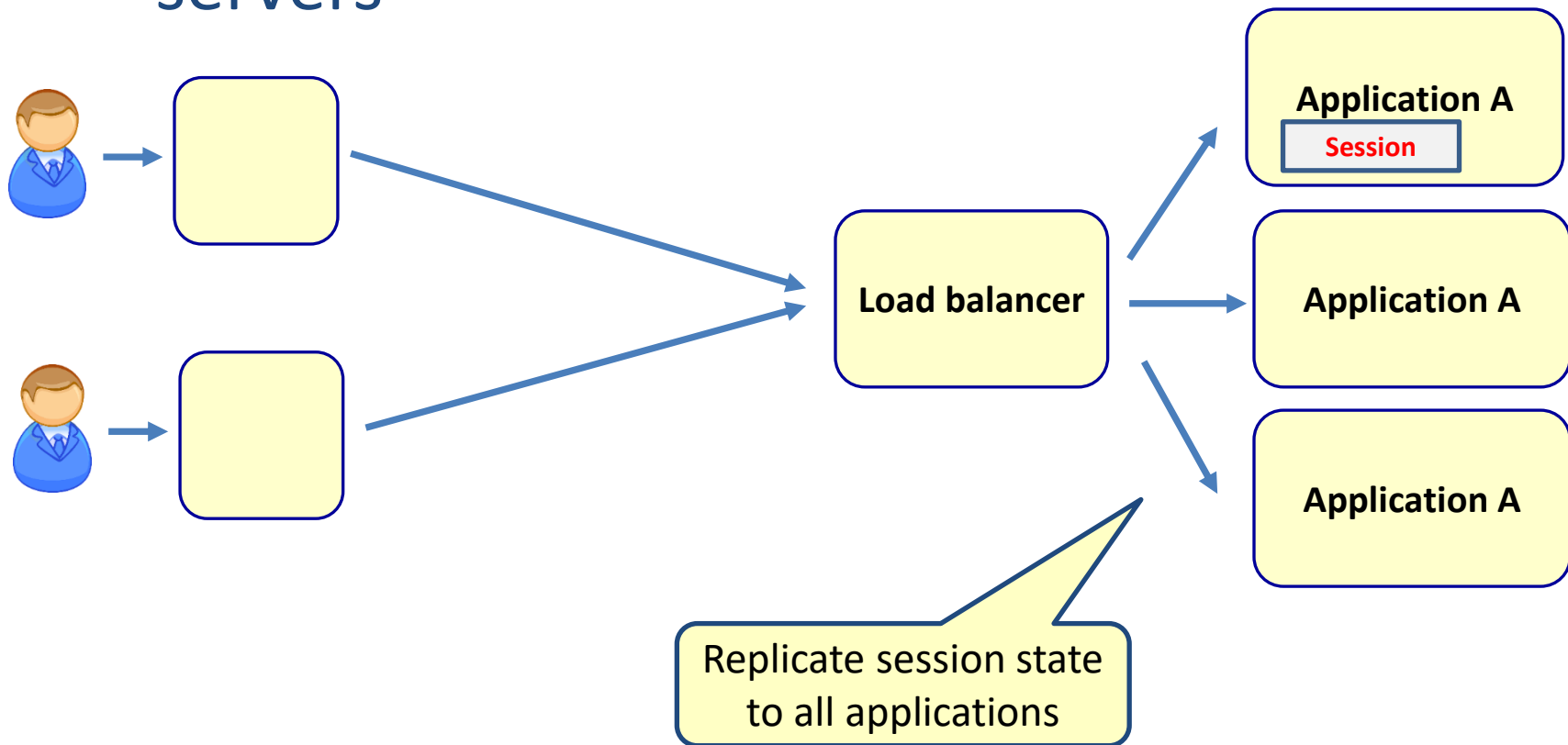


Web security



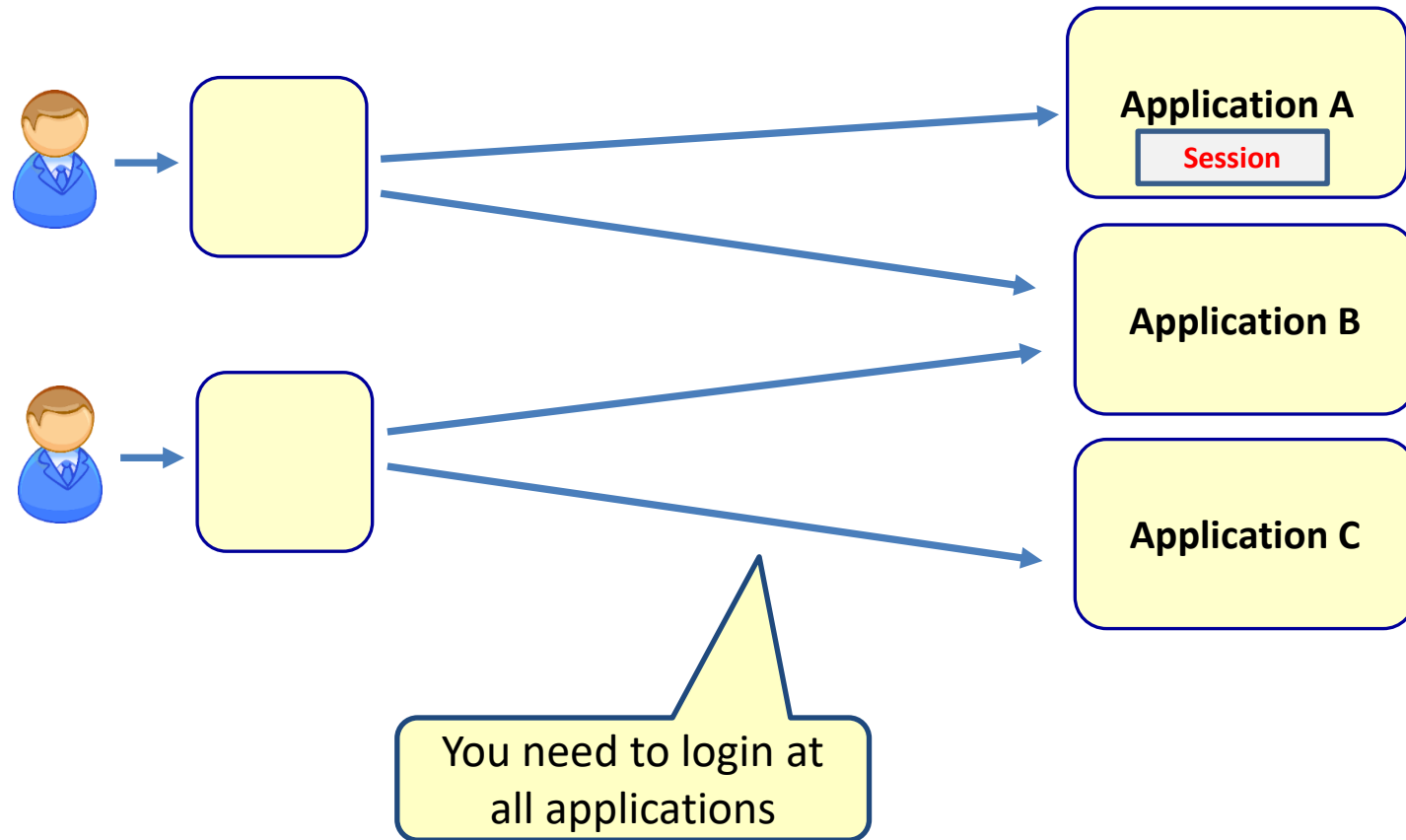
Problem with stateful security

- Hard to implement when we have multiple servers

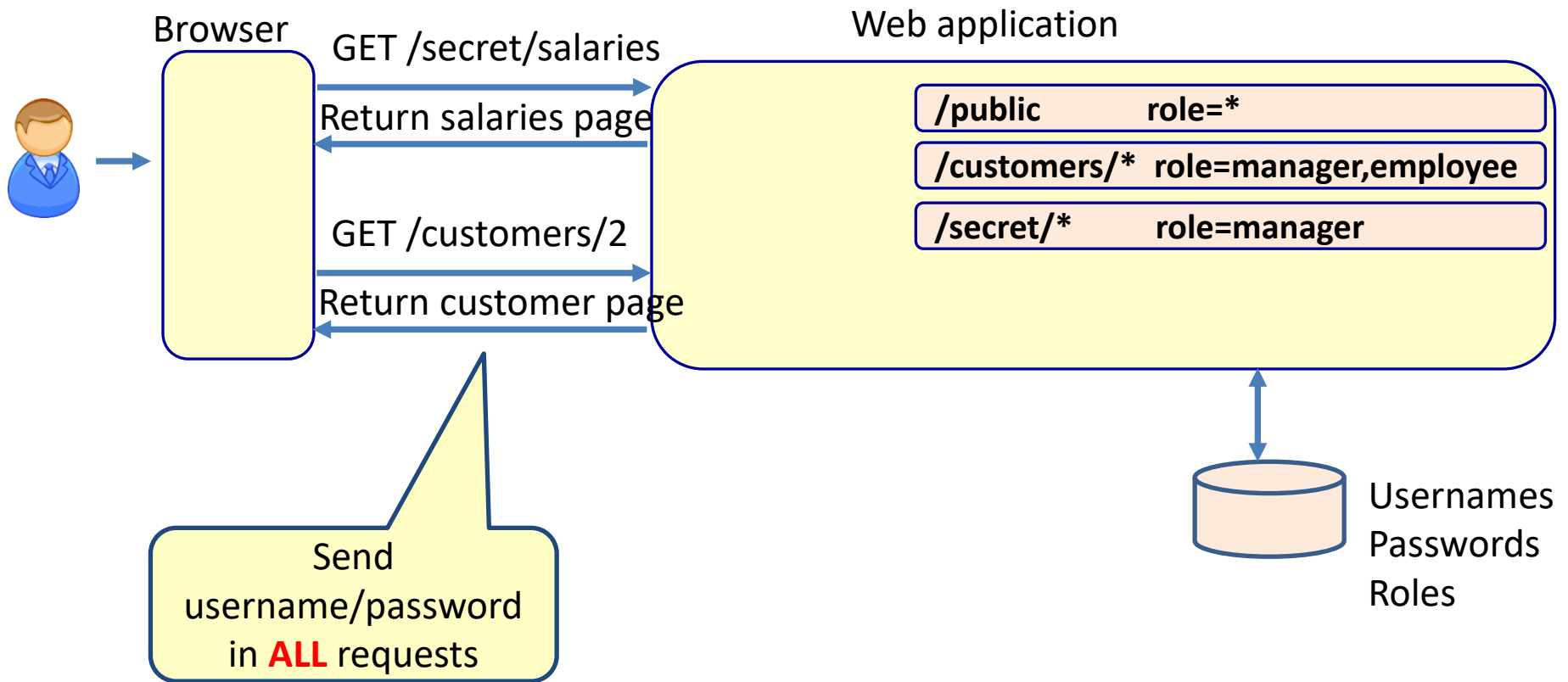


Problem with stateful security

- Calling multiple applications from one app



Better: Stateless security



REST SECURITY

REST security

- Form based authentication
 - Send a HTML login page to the client
 - In REST we cannot send a HTML login page to the client
- Basic authentication
 - Send the username/password information from the client to the server

Basic authentication

- Add the following header to the HTTP request:
 - Key = '**Authorization**'
 - Value = '**Basic**' + base 64 encoding of a user ID and password separated by a colon

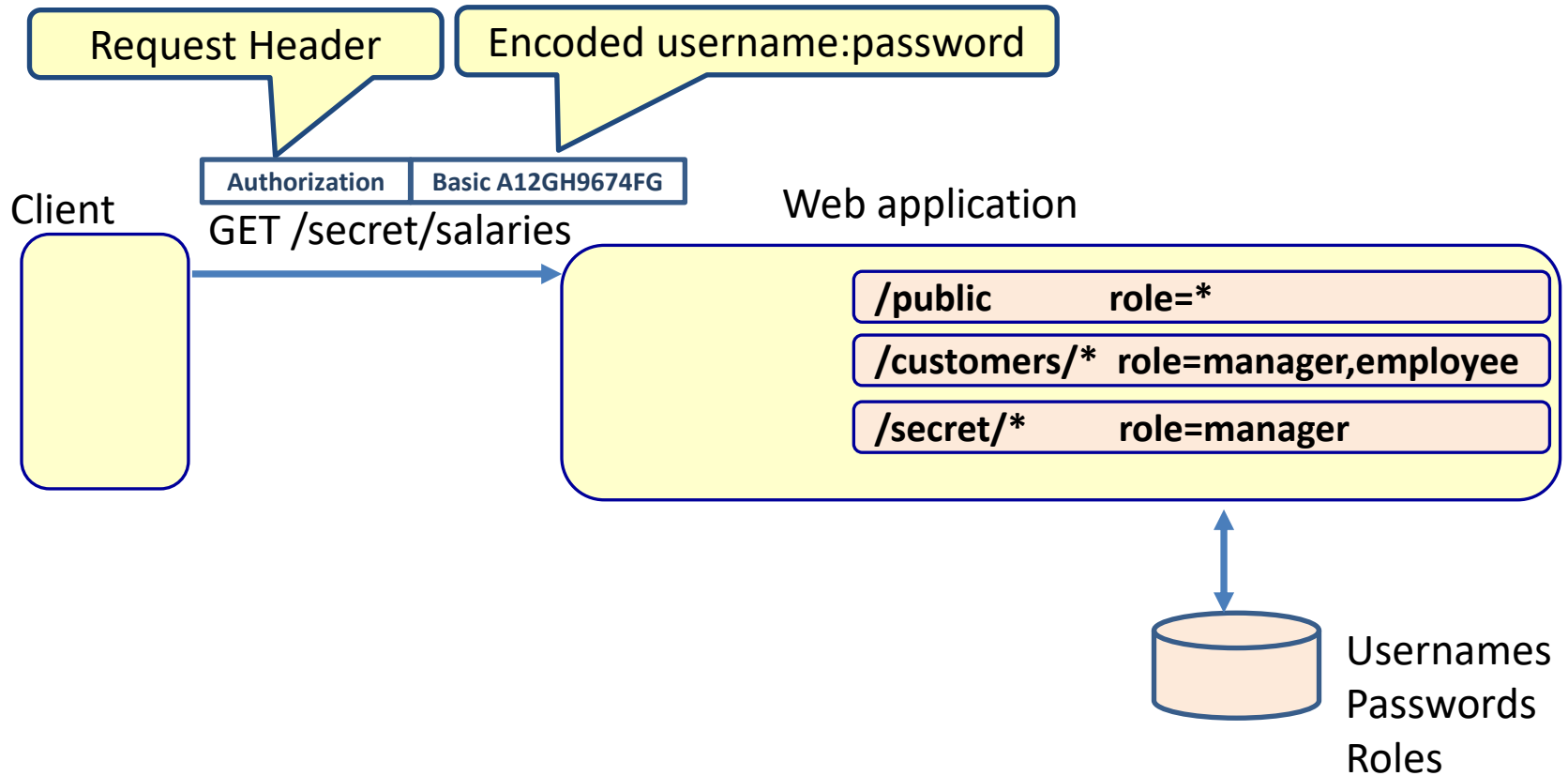
Headers ?	
Accept	application/json
Authorization	Basic YW5ha3Jhdmlk
Key	Value

Base64 encoding

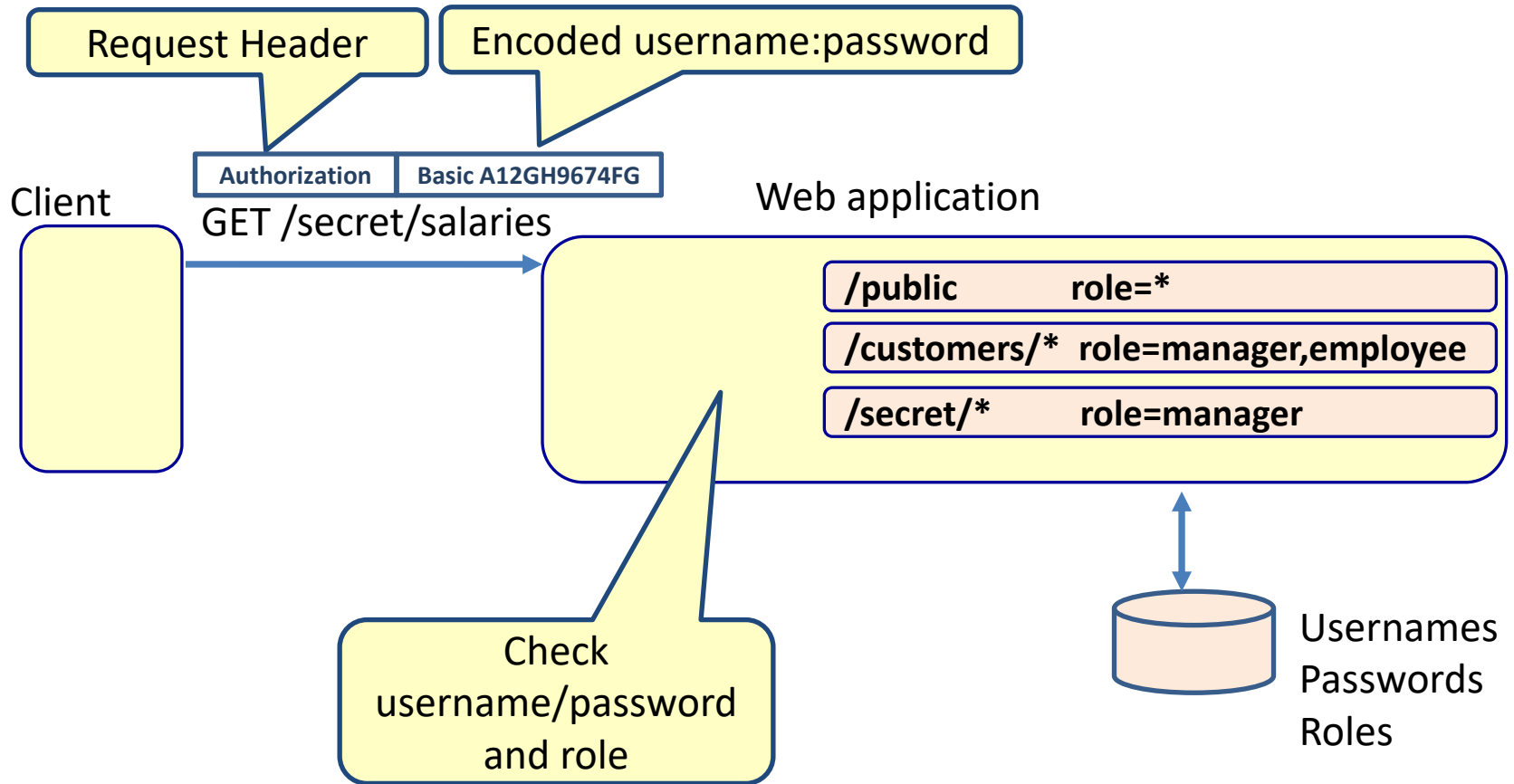
- Transform data into transport safe format
 - Characters that HTTP understands
- Not encryption (not secure)
- cryptii.com

The screenshot shows the cryptii.com Base64 encoding tool interface. It consists of three main panels. The first panel on the left is labeled 'Text' and contains the input text: 'The quick brown fox jumps over the lazy dog.'. The middle panel is labeled 'Base64' and shows the encoding options. It has a dropdown menu set to 'Base64 (RFC 3548, RFC 4648)' and a note indicating 'Encoded 60 chars'. The third panel on the right is labeled 'Text' and displays the resulting Base64 encoded string: 'VGhlIHFlYWNRIGJyb3duIGZveCBqdWlwcYBvdmVyIHRoZSBsYXp5IGRvZy4='.

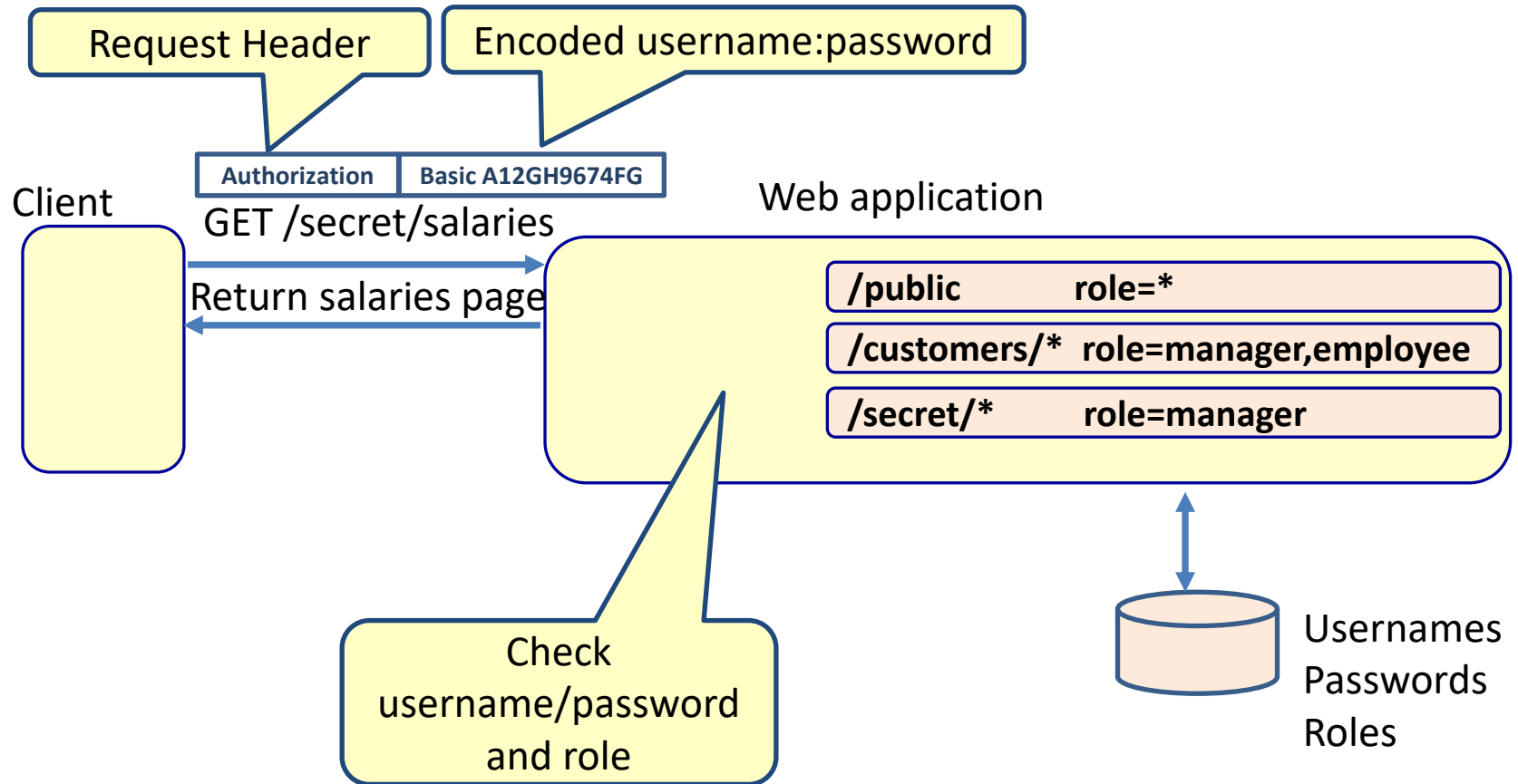
REST security



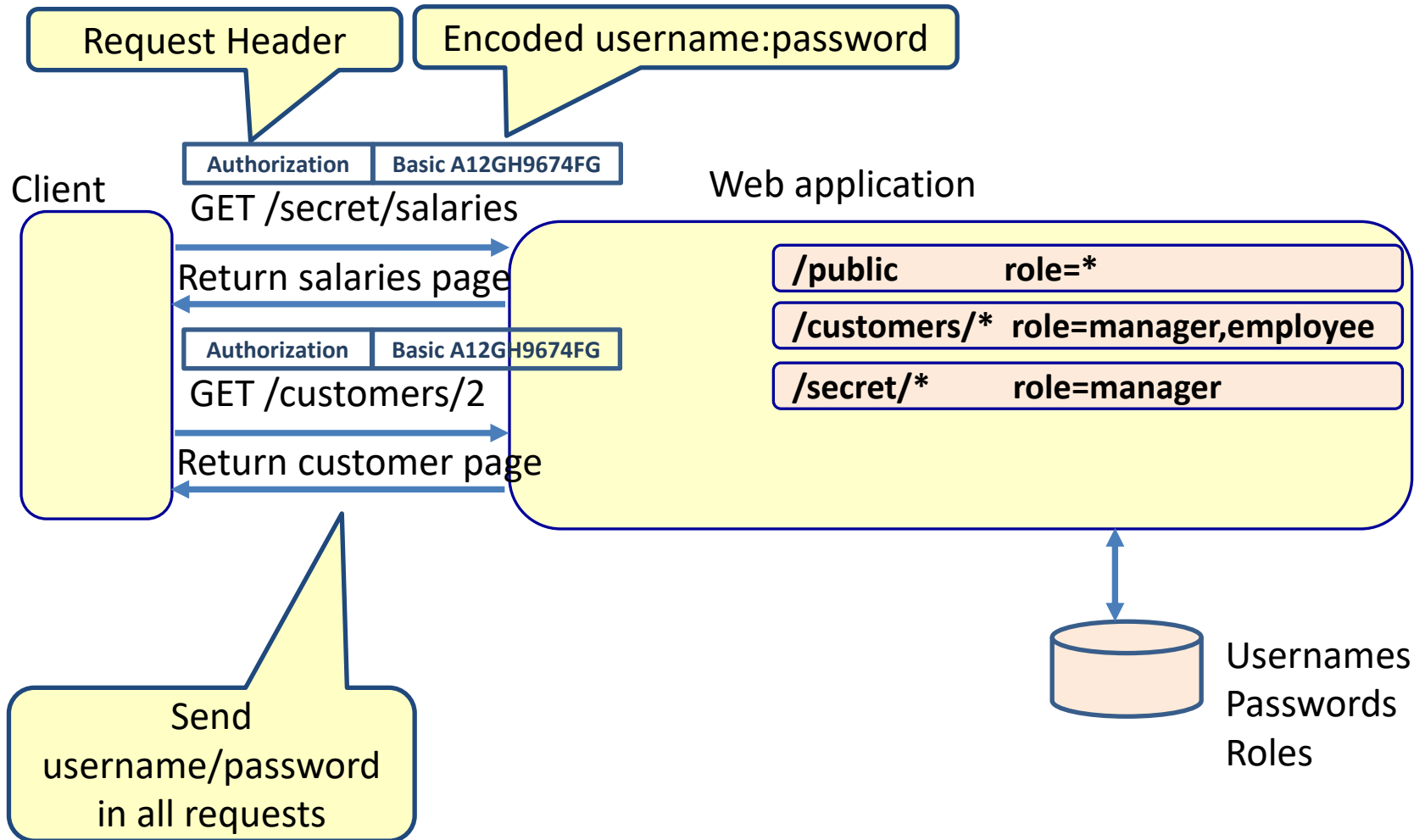
REST security



REST security



REST security



Rest security: Server

```
@RestController
public class MyController {

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

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

Spring security libraries

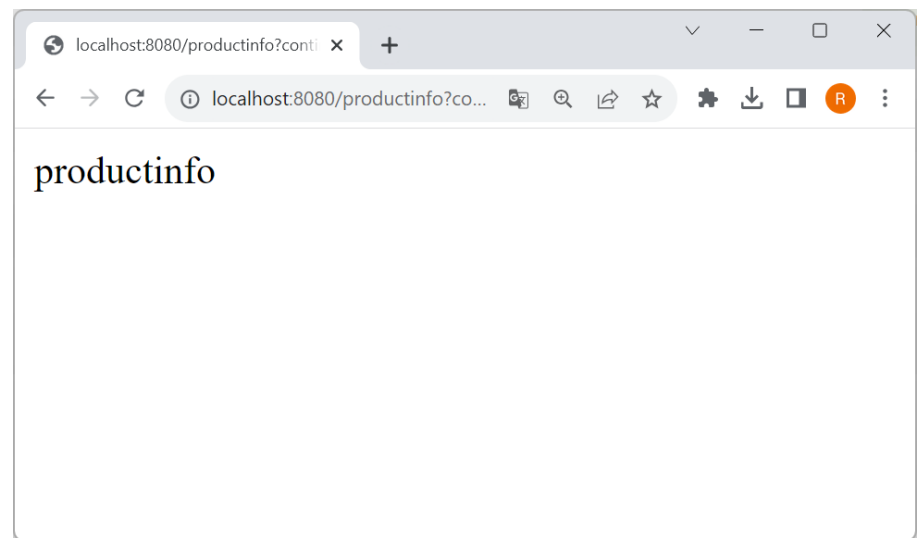
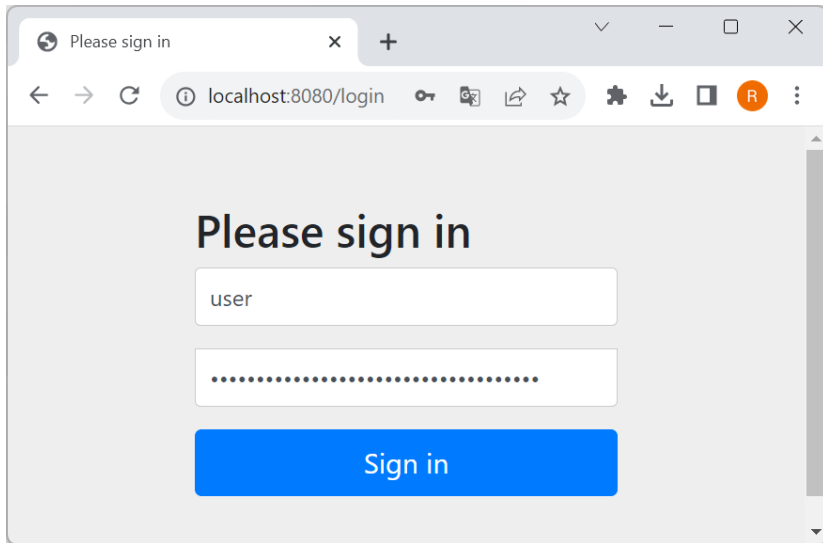
- When Spring Security is on the classpath, the auto-configuration secures all endpoints by default.

```
<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-f60139fdb96
```

Spring boot security



Set user in application.properties

⚙ application.properties ✕

```
1 spring.security.user.name=john
2 spring.security.user.password=1234
3
```

USERNAME/PASSWORD BASED SPRING BOOT SECURITY WITH IN MEMORY USERS

Spring boot security

@Configuration

```
public class UserDetailsServiceConfig {
```

@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();  
}
```

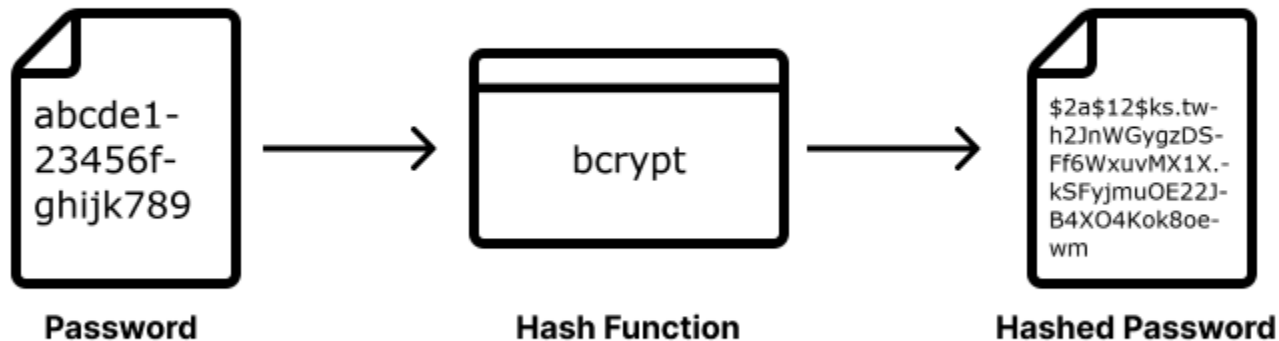
Authentication

Add in-memory users

BCryptPasswordEncoder

Hashing the password

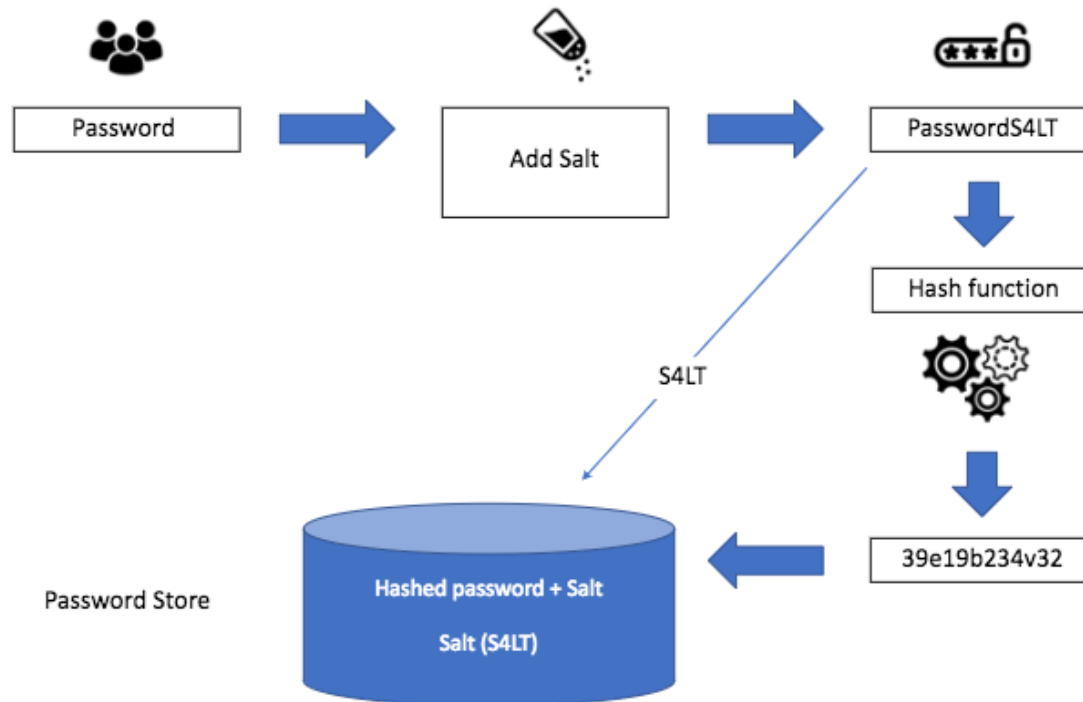
Password Hashing



You cannot retrieve the password from the hashed string

```
hash ("hello") = 3d3929g23994939e83b2ac5b9e29e1b1c19384
hash ("hb1lo") = 8dfac912a93f8169afe7dd238f33644939e83b
hash ("blitz") = 83b2afe7dd38f3364493938f33644939d3fg4f
```

Bcrypt encoding



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

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

Send

Params

Auth

Headers (8)

Body

Pre-req.

Tests

Settings

Cookies

Type

Basic Auth

The authorization header

Username

user

Password

user

Body

200 OK 63 ms 342 B

Save Response

Pretty

Raw

Preview

Visualize

Text

1 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

```
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;
}

BCryptPasswordEncoder

@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();
}

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

testdb.user

DOC

Documents

Aggregations

Schema

Explain Plan

FILTER { field: 'value' }

ADD DATA



VIEW



```
_id: "user"
password: "$2a$10$X83i1n6BIWK70JfPOszYPOgxWxm0T0xRcdtEp80Gu2tj/96iFsSZm"
roles: Array
  0: Object
    role: "user"
_class: "myapplication.security.userservice.User"
```

Bcryptdecoded
password

```
_id: "admin"
password: "$2a$10$L5ZootXdo0JvncJq5wbP.uXCejDx2j3T8uwcAmK4TTsYpBKy8o0i6"
roles: Array
  0: Object
    role: "admin"
_class: "myapplication.security.userservice.User"
```

Getting info data

The screenshot shows a REST client interface with the following components:

- Request Bar:** Method `GET` and URL `localhost:8080/info`. A `Send` button is on the right.
- Tabs:** `Params`, `Auth` (selected), `Headers (8)`, `Body`, `Pre-req.`, `Tests`, and `Settings`.
- Auth Panel:** Under the `Auth` tab, the `Type` is set to `No Auth`. A message states: "This request does not use any authorization. Learn more about [authorization](#)".
- Response Bar:** Shows `200 OK`, `116 ms`, and `337 B`. It includes a `Save as example` button.
- Response Body:** The `Body` tab is active, showing the response in `Text` format: `1 info`.

Getting user data

GET

localhost:8080/user

Send

ParamsAuthHeaders (9)BodyPre-req. TestsSettings

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

user

Password

user

Body

200 OK335 ms342 B

Save as example

PrettyRawPreviewVisualizeText

1 user info

Get the admin data

GET

localhost:8080/admin

Send

ParamsAuthHeaders (9)BodyPre-req. TestsSettings

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 OK90 ms344 B

Save as example

PrettyRawPreviewVisualizeText

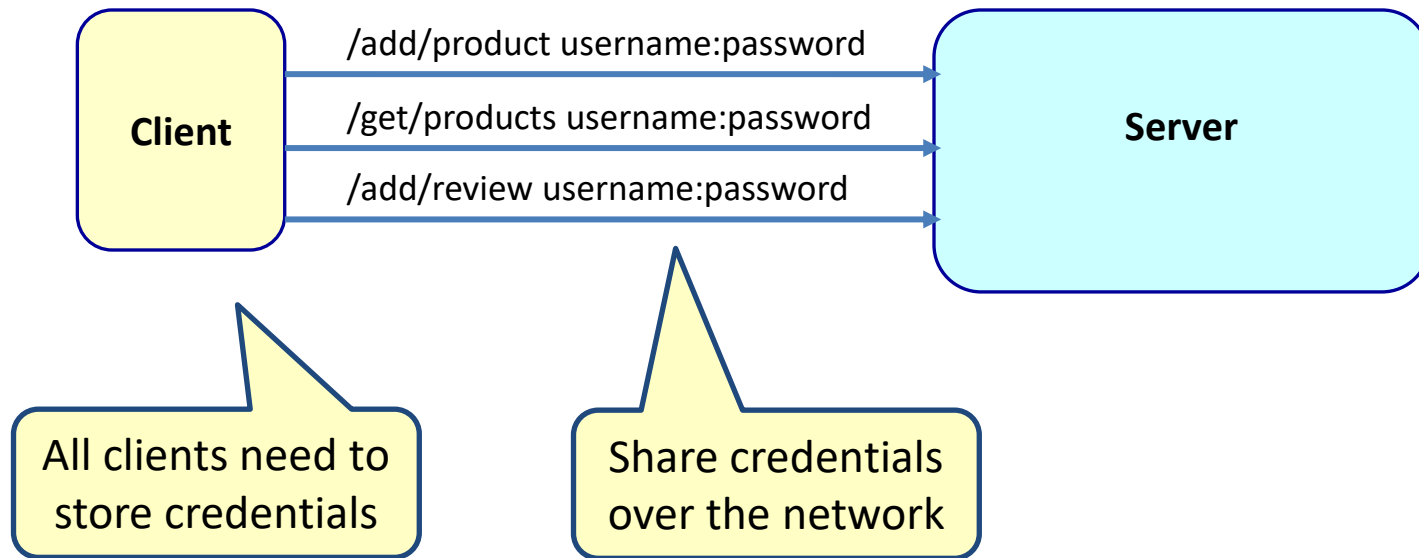
1 admin info

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.*

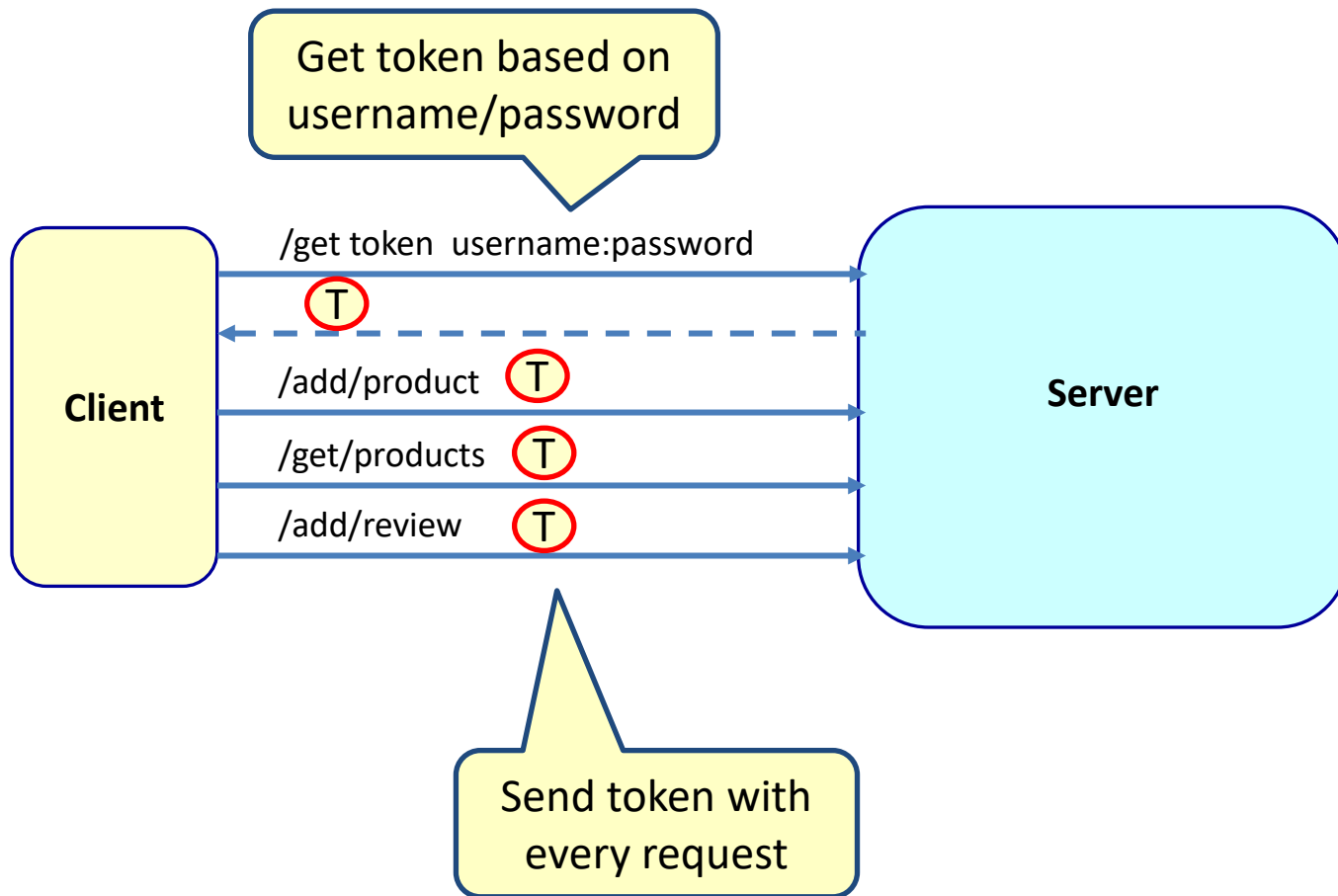
Problems with HTTP basic authentication

- We have to send credentials for every request



Better solution: JWT

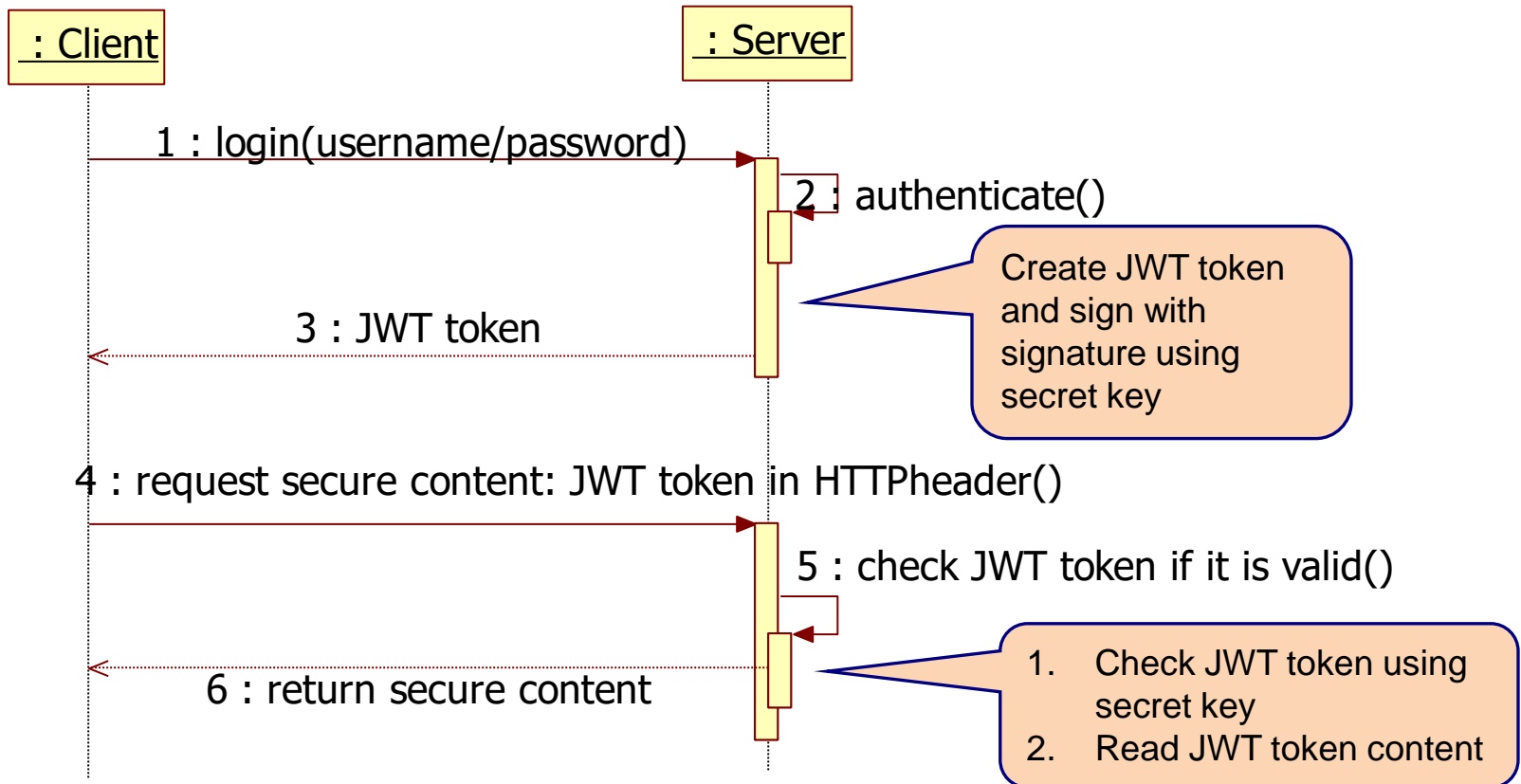
- Send a token for every request



JWT

JWT

■ JSON Web Token



Not secure, not encrypted content!

Algorithm HS256

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MzkwMjQ.eyJmcmVudCI6IjE2MzkwMjQifQ
```

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)

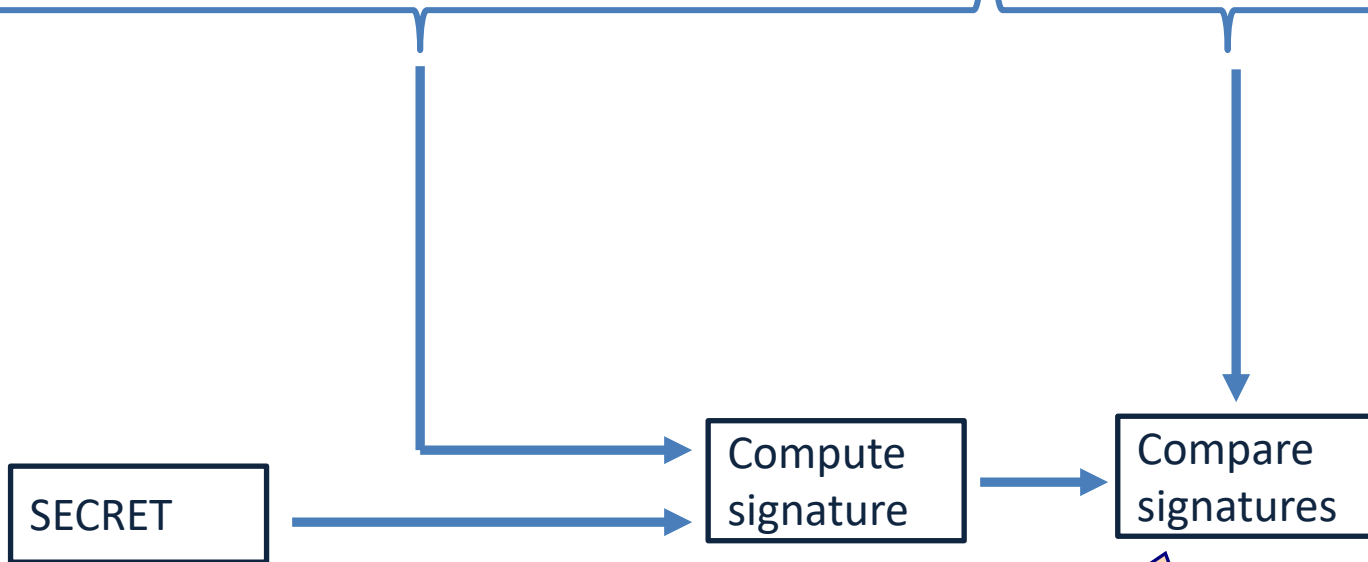
Issued at

VERIFY SIGNATURE

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

Check JWT signature

hhhhhhhhhhhhhhhhhh.ppppppppppppppppppppp.ssssssssssssssssssss



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

hhhhhhhhhhhhhhhhhhhh.pppppppppppppppppppppp.sssssssssssssssssss

Base 64
encode

Base 64
encode

```
{  
  ....  
  ....  
}
```

header

```
{  
  ....  
  ....  
}
```

payload

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

GET localhost:8080/api/all Send

Params

Query Params

	Key	Value	D...	...	Bulk Edit
	Key	Value	Description		

Body 200 OK 25 ms 355 B Save as example

Pretty Text

```
1 everyone can see this
```

GET localhost:8080/api/users Send

Params

Query Params

	Key	Value	D...	...	Bulk Edit
	Key	Value	Description		

Body 403 Forbidden 41 ms 439 B Save as example

Pretty JSON

```
1 {
2   "timestamp": "2023-11-07T02:08:27.566+00:00",
3   "status": 403,
4   "error": "Forbidden",
5   "path": "/api/users"
6 }
```

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

The screenshot shows a REST client interface with a POST request to `localhost:8080/auth/signup`. The request body is a JSON object with the following fields: `firstName` (bob), `password` (user), `email` (john@gmail.com), and `lastName` (johnson). The response is a 200 OK status with a response time of 367 ms and a body size of 489 B. The response body is a JSON object containing a `token` field with a long alphanumeric string.

Request:

```
POST localhost:8080/auth/signup
```

Body (JSON):

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

Response:

```
200 OK 367 ms 489 B
```

Body (JSON):

```
{
  "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJqb2huQGdtYWlsLmNvbSIsIm1hdCI6MTY5OTMyMzY1MCwiZXhwIjoxNjk5MzI3MjUwfQ.G-iz_VDiJkwkcyxvsUH4PHBz-5YqY6nCD4HdvoZMWfA"
}
```

database

testdb.user

⌵

Documents

Aggregations

Schema

Explain Plan

 FILTER { field: 'value' }

 ADD DATA ▾



VIEW



```
_id: "admin@admin.com"
firstName: "admin"
lastName: "admin"
password: "$2a$10$0p6pyW01hHWH2LcMS1lx..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$10$Ze31ktb9PZZn9CNTuD4ZVONrzwK2HGzschOFoHky0g.LQR2.xUQQC"
role: "ROLE_USER"
updatedAt: 2023-11-07T02:26:46.944+00:00
_class: "jwtexample.models.User"
```

Signin user

POST localhost:8080/auth/signin Send

Params Auth Headers (10) **Body** Pre-req. Tests Settings

raw JSON Beautify

```
1 {  
2   "email": "john@gmail.com",  
3   "password": "user"  
4 }
```

Body 200 OK 310 ms 489 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiJ9.  
    eyJzdWIiOiJqb2huQGdtYWlsLmNvbSIsIm1hdCI6MTY5OTMyNDM1OC  
    wiZXhwIjoxNjk5MzI3OTU4fQ.  
    lP3mhrYqfQ_wFuppqcm1N2piLEZl9SeCB4Zyik9-FK0"  
3 }
```

Token for user

Get users info

GET localhost:8080/api/users Send

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

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](#)

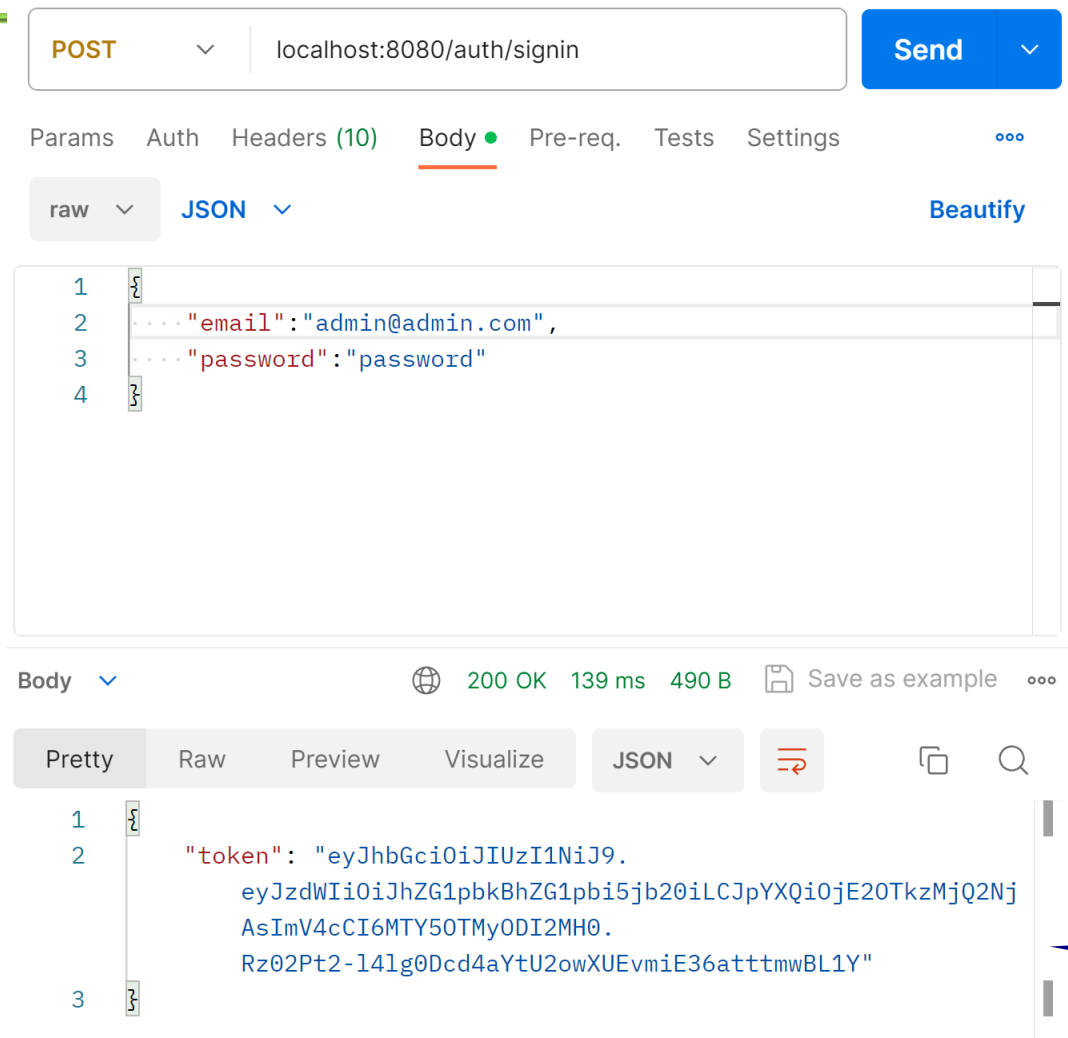
Token
eyJhbGciOiJIUzI1NiJ9.eyJzdWUiOiJqb2huQC...

Body 200 OK 81 ms 357 B Save as example

Pretty Raw Preview Visualize Text

```
1 ONLY users can see this
```


Signin admin



POST localhost:8080/auth/signin

Send

Params Auth Headers (10) Body Pre-req. Tests Settings

raw JSON Beautify

```
1 {
2   ... "email": "admin@admin.com",
3   ... "password": "password"
4 }
```

Body 200 OK 139 ms 490 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbkBiZG1pb20iLCJpYXQiOiJlY20kZmJQ2NjAsImV4cCI6MTY5ODMyODI2MH0.Rz02Pt2-14lg0Dcd4aYtU2owXUEvmiE36atttmwBL1Y"
3 }
```

Token for admin

Get admins info

The screenshot shows a REST client interface with the following components:

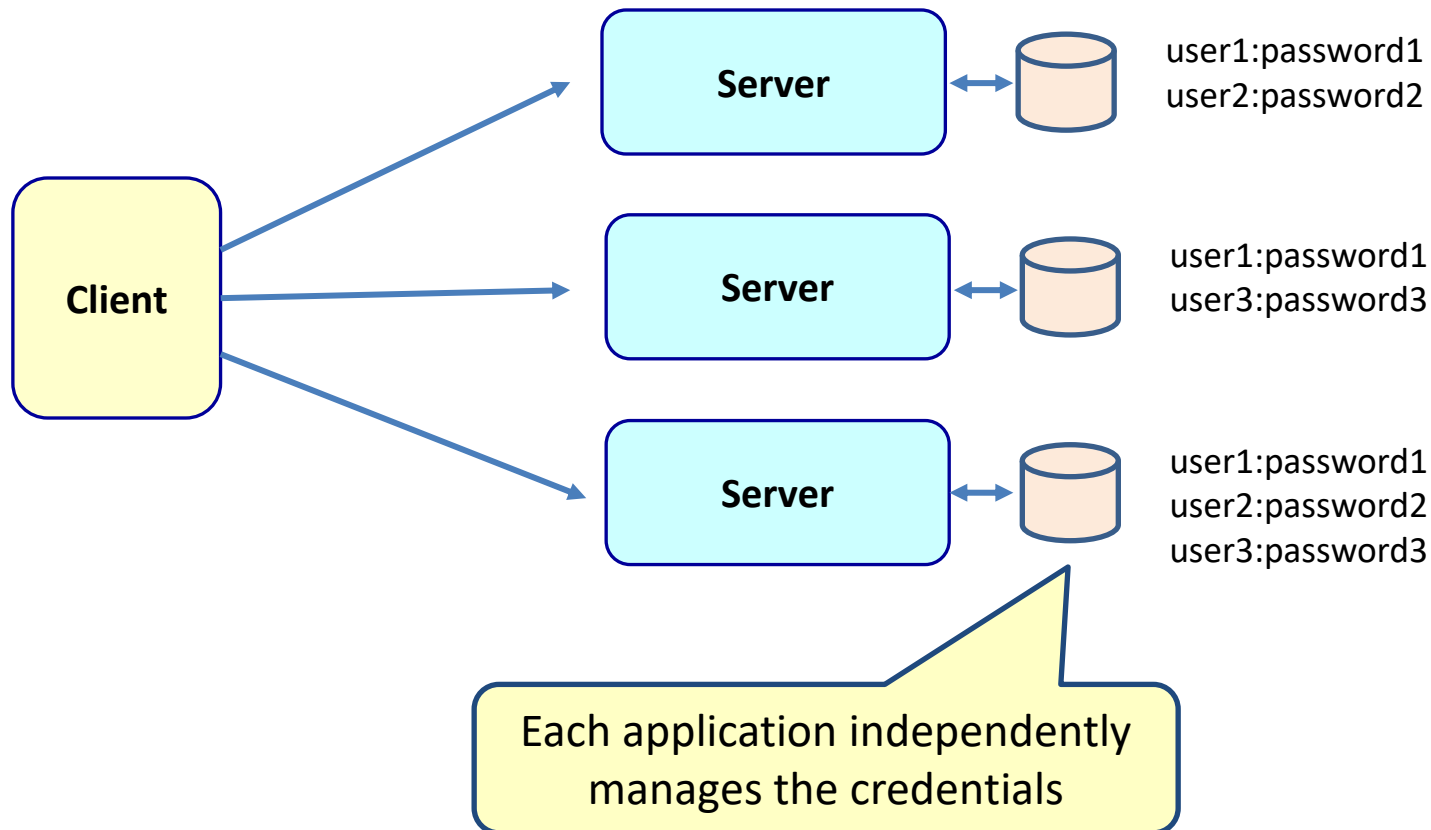
- Request Bar:** Method `GET`, URL `localhost:8080/api/admins`, and a `Send` button.
- Tabs:** `Params`, `Auth` (selected), `Headers (9)`, `Body`, `Pre-req.`, `Tests`, and `Settings`.
- Auth Section:**
 - Type:** `Bearer Token`.
 - Description:** "The authorization header will be automatically generated when you send the request. Learn more about".
 - Token:** `eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbk...`
- Response Bar:** Status `200 OK`, Time `13 ms`, Size `358 B`, and a `Save as example` button.
- Response Body:** `1 ONLY admins can see this`

JWT

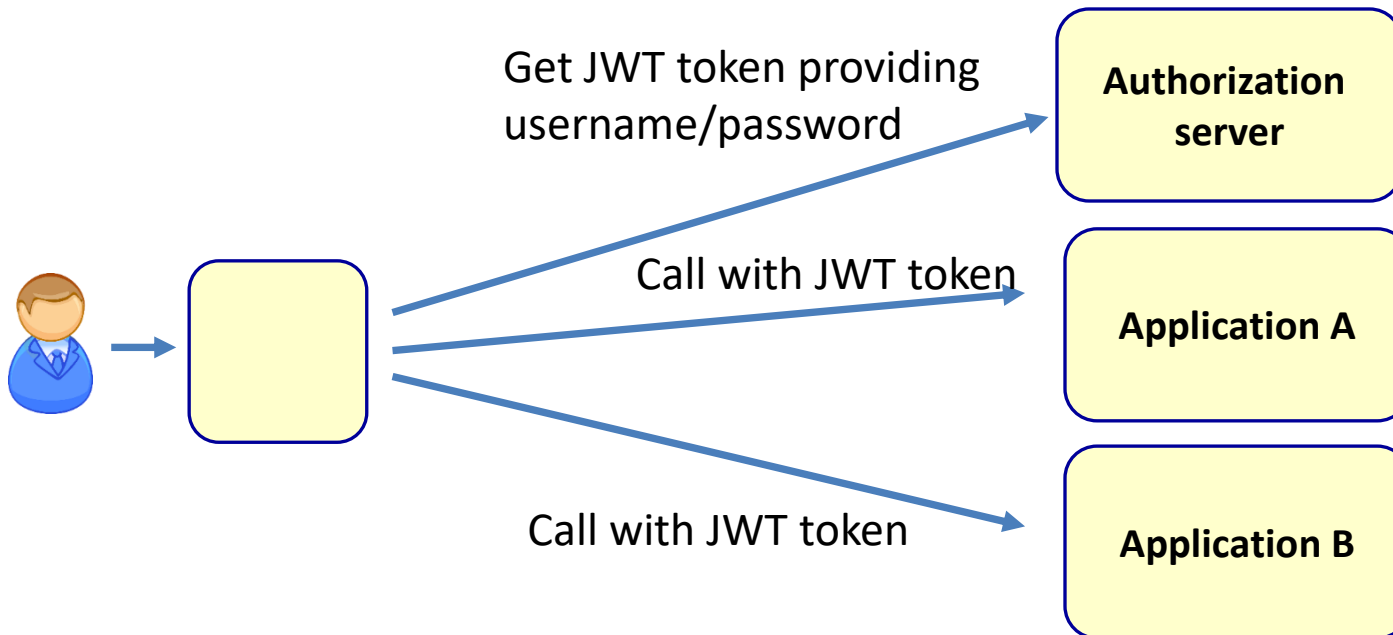
- Never place secure content in a JWT token
 - JWT token is only signed, not encrypted
- What if someone steals the JWT token
 - Use token expiration
 - Server maintains list of blacklisted JWT's

Problems with HTTP basic or JWT authentication

- Every system needs to manage these credentials



Solution: OAuth2 + JWT



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 a 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

