WELCOME BACK
LESSON 6

Spring/Spring Boot Crash Course

For TD Bank

MEET YOUR CRASH COURSE TEAM

TANGY F.
CEO

HAL M.
DEVELOPER

WILLIAM D.
DEVELOPER

LESSON 6

AGENDA

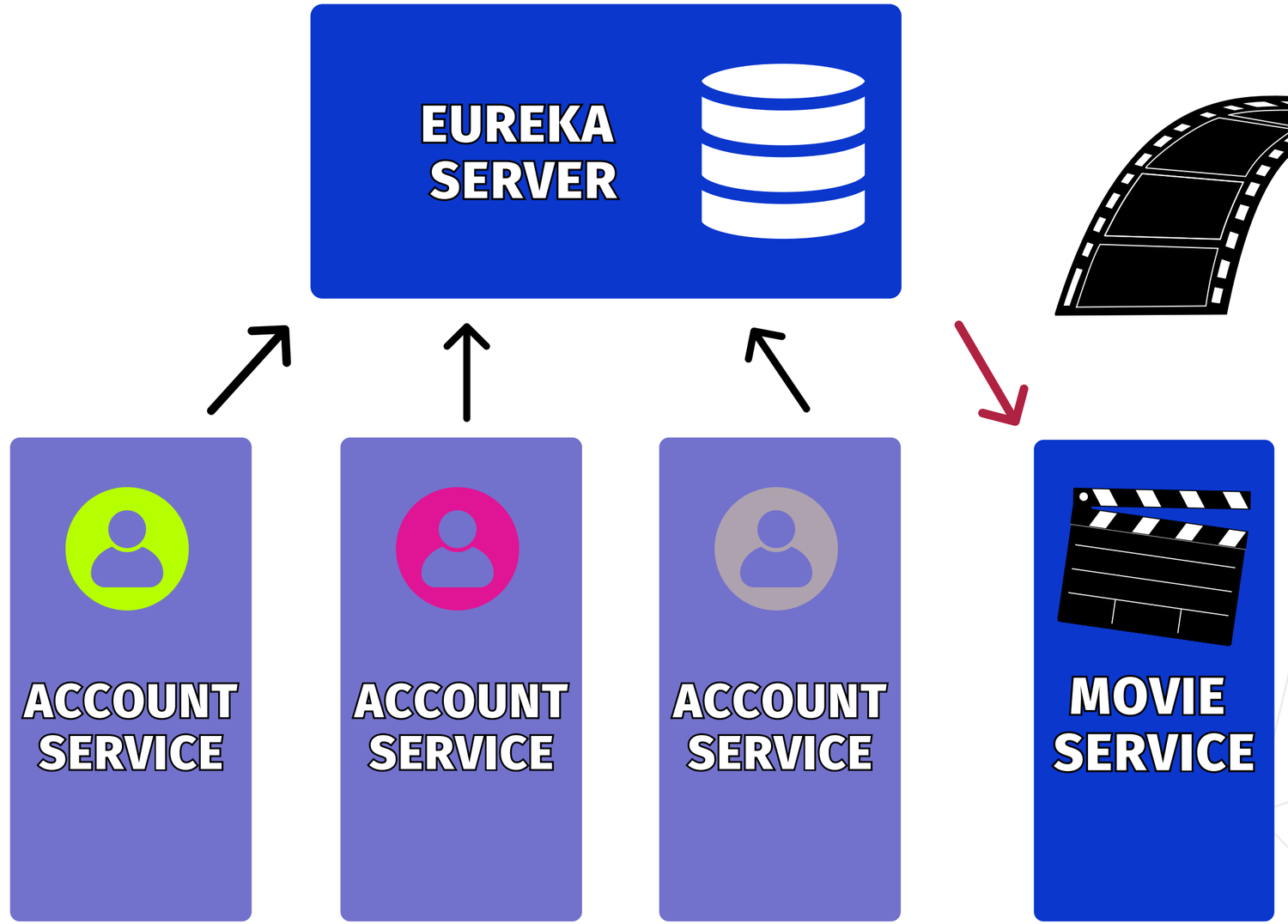
1. Recap Lesson of 5:
>>Rapid review of lesson five

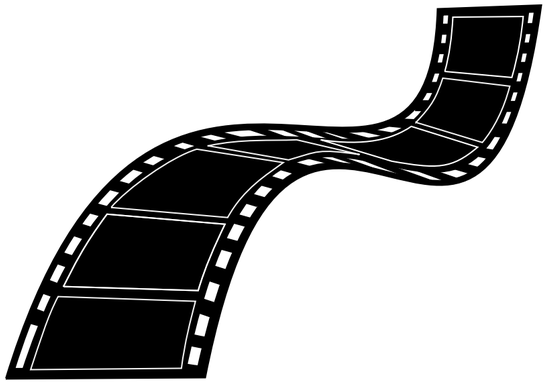
2. Current Lesson- Authorization using JWT
{Jason Web Token}

3. Q&A

WE ARE BUILDING

BITE SIZE
MOVIE REVIEW APP





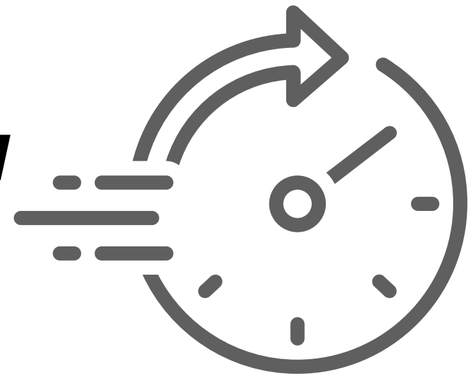
Creative Devs. Software

Rapid Review of

LESSON 4&5

WELCOME TO LESSON 6

Rapid Review Trivia

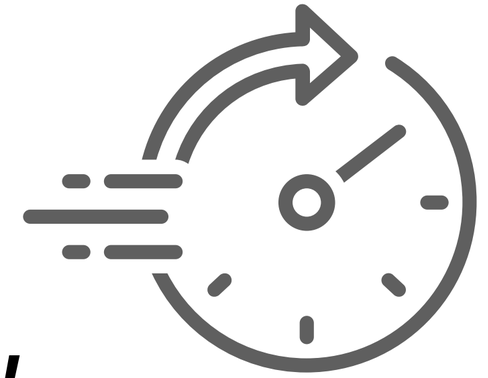


1. What are advantages & disadvantages of using an Integer as unique table index instead of a long?

2. What are the advantages of JWT?

WELCOME TO LESSON 6

Rapid Review of LESSON 5



Rapid Review

Spring Security II

LESSON 6

Authorization using JWT

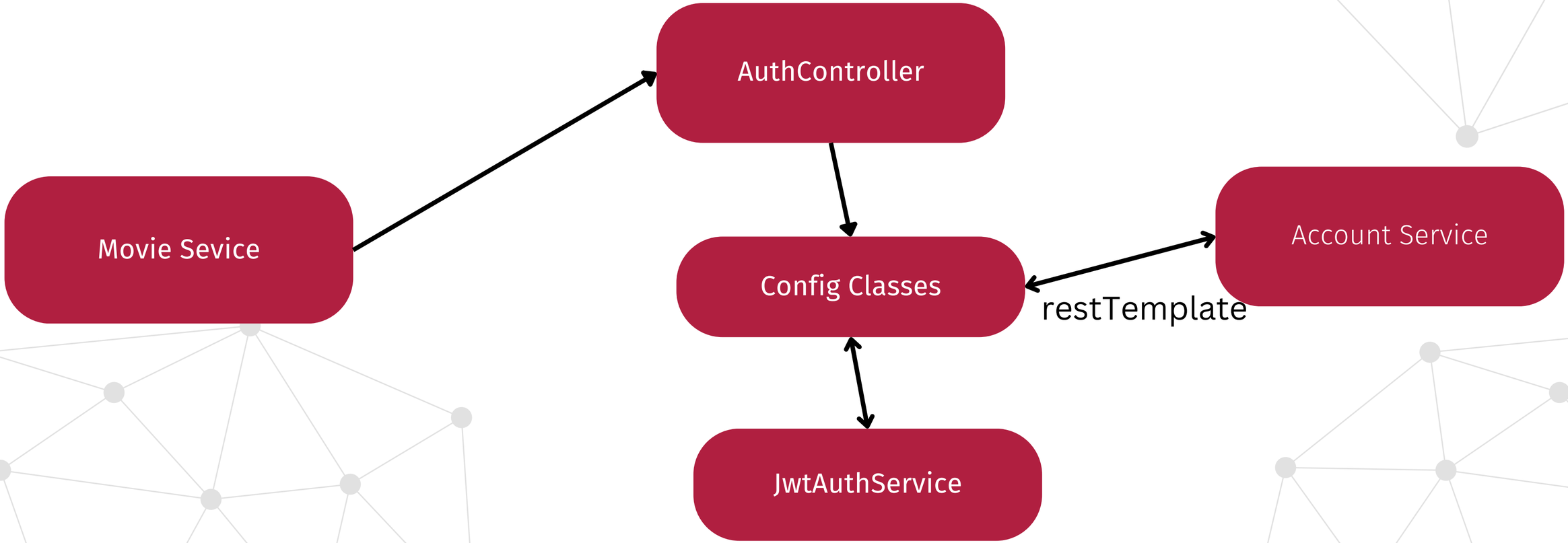
LESSON 6

AUTHORIZATION USING JWT

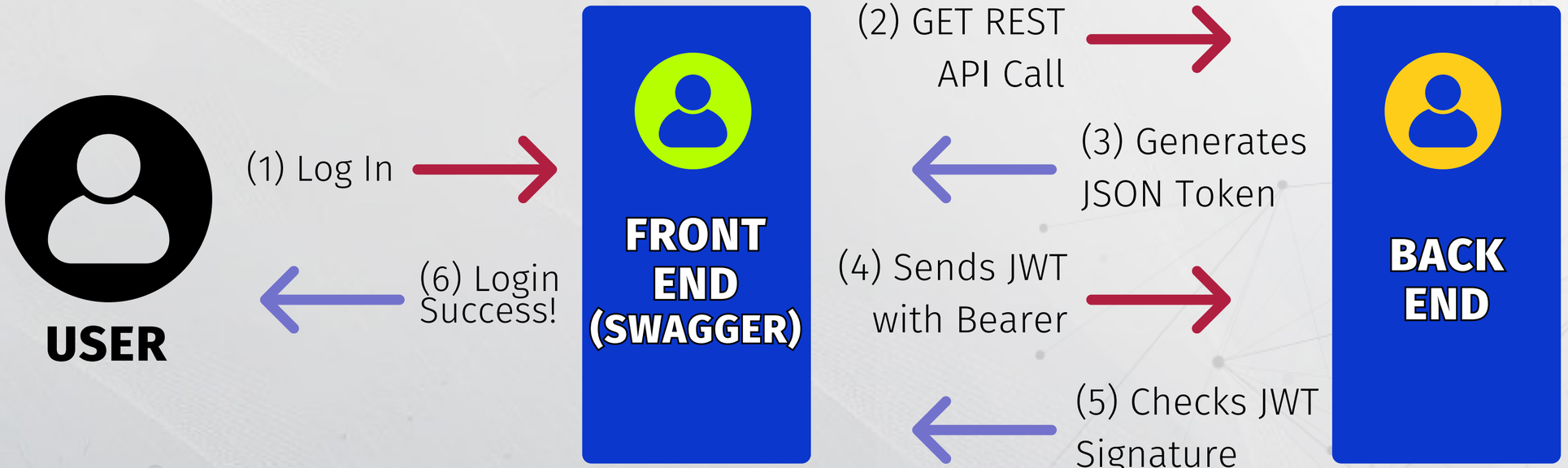
1. Recap of how to generate JWT token
2. Implementing AuthFilter to enable JWT
3. Modification of the services enabling PreAuthorization

THE FLOW OF MOVIE & ACCOUNT SERVICE WITH JWT

LESSON 6



BEARER AUTHENTICATION WITH SWAGGER



IMPLEMENTATION OF JWT

LESSON 6

Authorization

Performing authorization using JWT for selected methods

Advantages of JWT

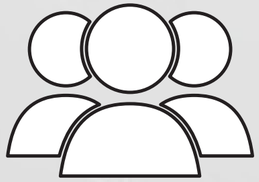
Benefits of implementing JWT

12-FACTORS & DESIGN PATTERNS

12 Factor App

Factor 8: Concurrency

Take note of how our JWT Bearer Authentication data is handled on shutdown.

Design Patterns 

Share with us what they are or how they were used?

Singleton Pattern

Chain of Responsibility Pattern

DESIGN PATTERNS

Bridge Pattern

- Is used when we need to decouple an abstraction from multiple varying implementations.
- It combines interfaces and abstract classes
  - Use classes implementing an interface for low level specific behavior that varies
  - Use an abstract class to defines the contract for the client code. It contains a reference to the implementation and delegates the low-level operations to the implementation.
  - A subclass of the abstract class implements common logic among all varying low level logic
- The client class instantiate both, the low level class and subclass, passing low level class to subclass via parameter.

CODING EXERCISE #1

The below code is from our Movie Service application. Write one line of code to obtain an Account entity using the REST API Endpoints of our Account Service application. Assume that the Account entity exists in both applications, and that ACCOUNTURL correctly refers to the port that Account Service is being run on.

```
@Autowired
AccountInfoUserDetailsService accountInfoUserDetailsService;

public static final String ACCOUNTURL = "http://ACCOUNT-SERVICE/rest/account/getaccountbyusername/";

@Autowired
RestTemplate restTemplate;

public Account getAccount() {
    String username = UserUtil.getLoggedInUsername();
    Account account = //Code Goes Here
    return account;
}
```

CODING EXERCISE #1

The below code is from our Movie Service application. Write one line of code to obtain an Account entity using the REST API Endpoints of our Account Service application. Assume that the Account entity exists in both applications, and that ACCOUNTURL correctly refers to the port that Account Service is being run on.

```
@Autowired
AccountInfoUserDetailsService accountInfoUserDetailsService;

public static final String ACCOUNTURL = "http://ACCOUNT-SERVICE/rest/account/getaccountbyusername/";

@Autowired
RestTemplate restTemplate;

public Account getAccount() {
    String username = UserUtil.getLoggedInUsername();
    Account account = //Code Goes Here
    return account;
}
```

```
restTemplate.getForObject(ACCOUNTURL + "/" + username, Account.class);
```

CODING EXERCISE #2

Below is a snippet of code from our JWT Service, with functions to generate tokens, create tokens, validate tokens, and check if a token is expired. Which part of this code is incorrect?

```
private Boolean isTokenExpired(String token) { return extractExpiration(token).before(new Date()); }

public Boolean validateToken(String token, UserDetails userDetails) {
    final String username = extractUsername(token);
    return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
}

public String generateToken(String userName) {
    Map<String, Object> claims = new HashMap<>();
    return createToken(claims, userName);
}

private String createToken(Map<String, Object> claims, String userName) {
    return Jwts.builder()
            .setSubject(userName)
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + expiryTime))
            .signWith(getSignKey(), SignatureAlgorithm.HS256).compact();
}
```

CODING EXERCISE #2

This code never sets the claims of the JWT, which will make it impossible for the service to check the validity of tokens. The correct createToken method is as follows:

```
private String createToken(Map<String, Object> claims, String userName) {
        return Jwts.builder()
                .setClaims(claims)
                .setSubject(userName)
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + expiryTime))
                .signWith(getSignKey(), SignatureAlgorithm.HS256).compact();
    }
```

THANK YOU

Crash Course

We will see you Tomorrow