

A large, semi-transparent red arrow points from left to right, covering the left half of the slide.

**WELCOME BACK
LESSON 7**

<Creative
Software/>

Spring/Spring Boot
Crash Course

For TD Bank

MEET YOUR CRASH COURSE TEAM



TANGY F.
CEO



HAL M.
DEVELOPER



WILLIAM D.
DEVELOPER

AGENDA

LESSON 7

- 1 Recap Lesson of 6:
>>Rapid review of lesson six
- 2 Current Lesson- Unit Testing & Hibernate
- 3 Lab / Assessment and Q&A

UPDATE ABOUT YOUR YOUR ENVIRONMENT SET UP



The screenshot shows a GitHub repository page for 'Cre8tivePilot /'. The left sidebar has options like 'Clone', 'Create branch', 'Create pull request', 'Create fork', and 'Compare'. The main area shows a file named 'README.md' with the following content:

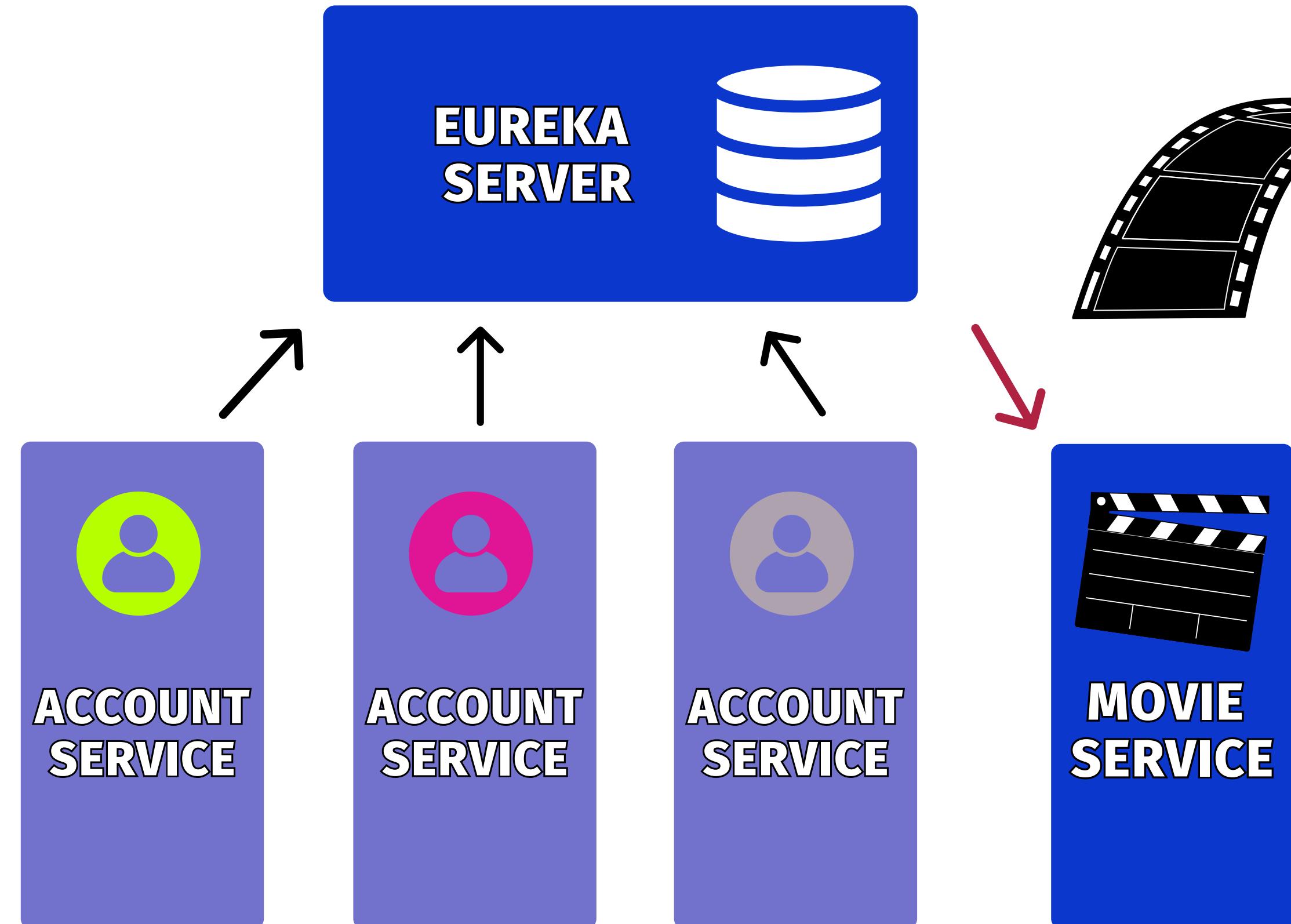
Don't configure the repository With this shell application in this project
Prompt: The purpose of this pre-assessment, is to get an understanding of your knowledge in SpringBoot. The pre-assessment is 30 minutes. Do not use Google or AI tools for this part. The instructions to send the code are below.
You will be creating a profile application using your current knowledge of Spring SpringBoot.
1. Set your time for 30 minutes.
2. Download the ZIP file with the project.
3. Create a new repository (using any service you want), and add the downloaded folders into the repository.
4. Open the "ProfileService" project in your IDE.
5. Add an optional dependency for lombok, from the group projectlombok
6. Add a dependency for hibernate-validator version 8.0.Final, from the group hibernate-validator
7. Fill out the Profile entity class and AuthRecord entity class using the comments in the shell, showing best practices for efficient programming
8. Fill out the controller class using the comments in the shell, following best practices for 12-factor app
9. Create a MySQL database. Create databases named ProfileData and SongData. Connect those databases to the ProfileService application.
10. Create a test that verifies the REST API endpoint that you created.
11. Run the application.
12. When time stops at 30 minutes, push to the repository.

Longer Pre-assessment
During the course you had 10 mins to do a pre assessment. This is a take home assessment that will also help. We look forward to finding you sharing your results
Prompt:
The purpose of this pre-assessment, is to get an understanding of your knowledge in Spring Boot. The pre-assessment is 30 minutes. We ask that you avoid using Google or AI tools for this part. The instructions to send the code are below.



WE ARE BUILDING

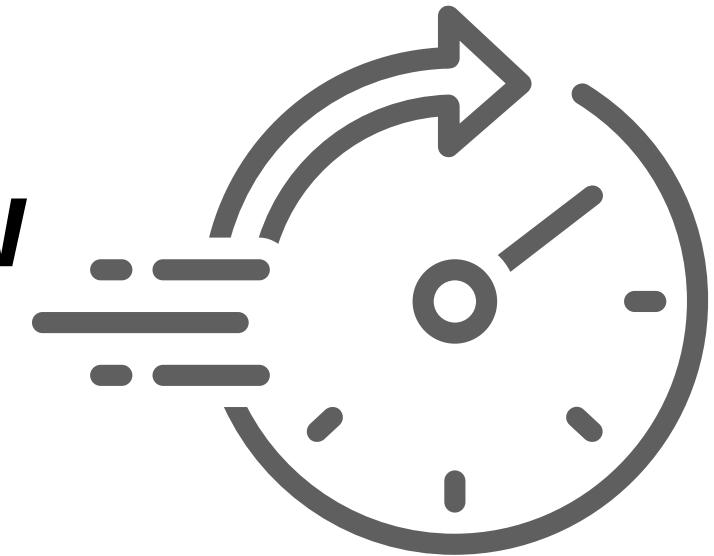
BITE SIZE
MOVIE REVIEW APP



<Cre8tive
Software/>
Devs.

WELCOME TO LESSON 7

Rapid Review Trivia

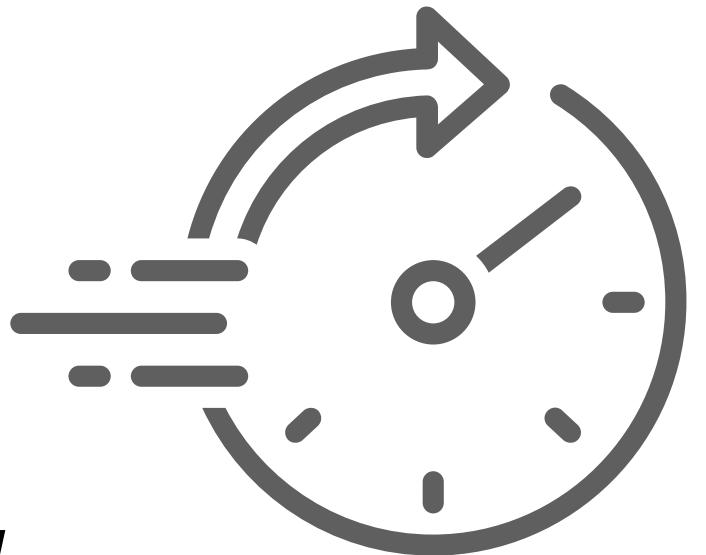


RAPID REVIEW



1. Which Spring class do we use to extract an entity from a REST API call?

WELCOME TO
LESSON 7

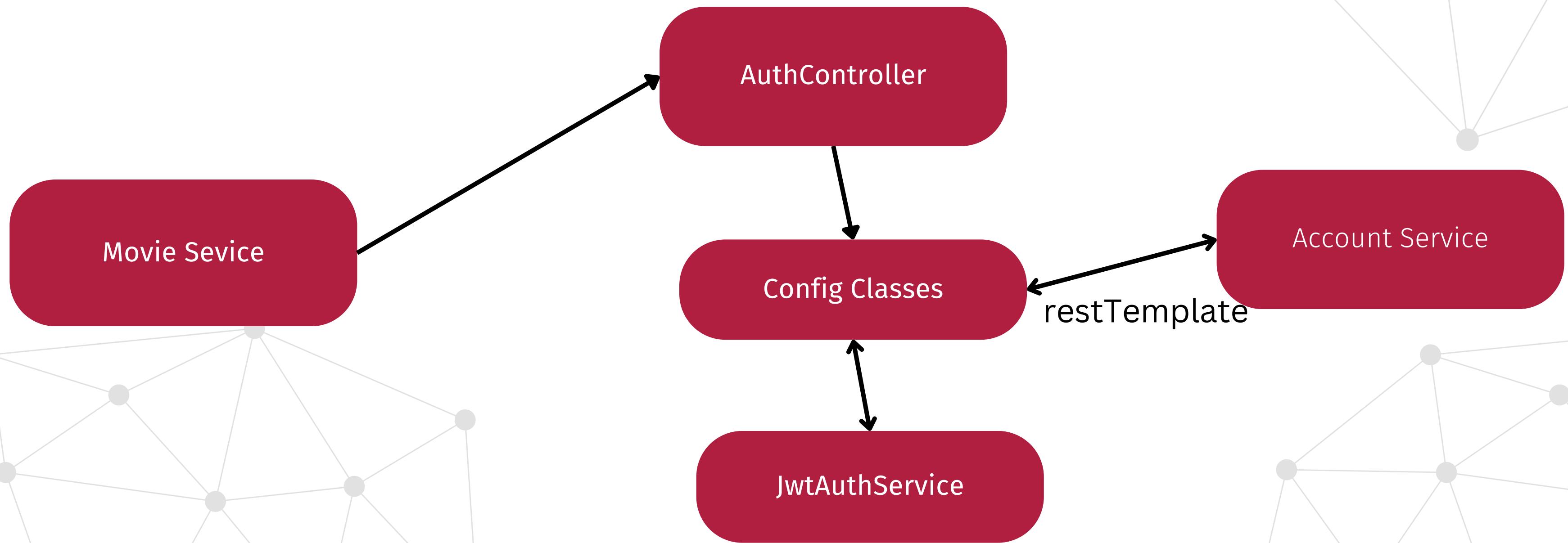


Rapid Review
of
LESSON 6

Rapid Review
JWT

THE FLOW OF MOVIE & ACCOUNT SERVICE WITH JWT

REVIEWING LESSON 6



LESSON 7

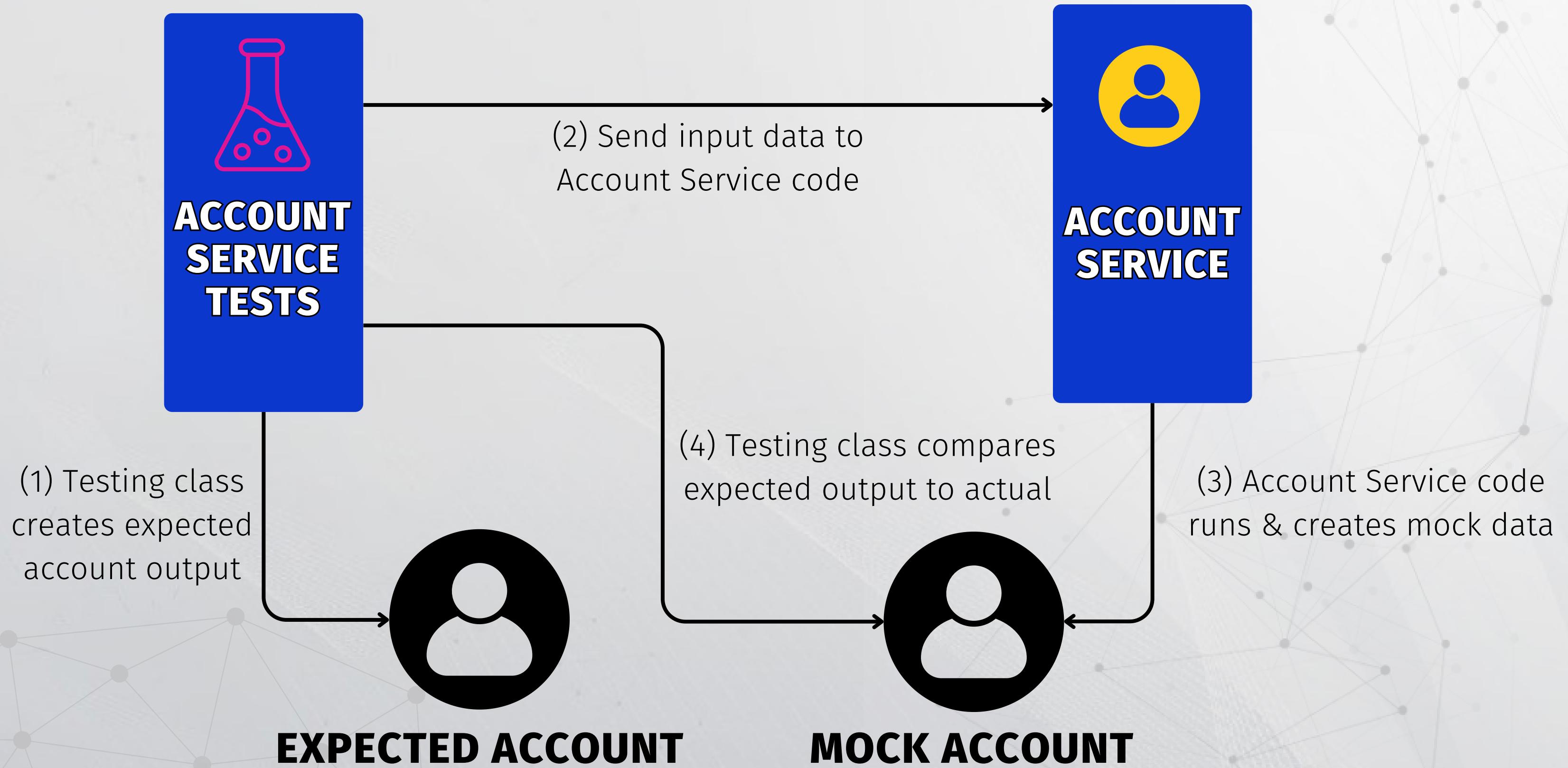
Unit Testing
&
Hibernate

LESSON 7

UNIT TESTING & HIBERNATE

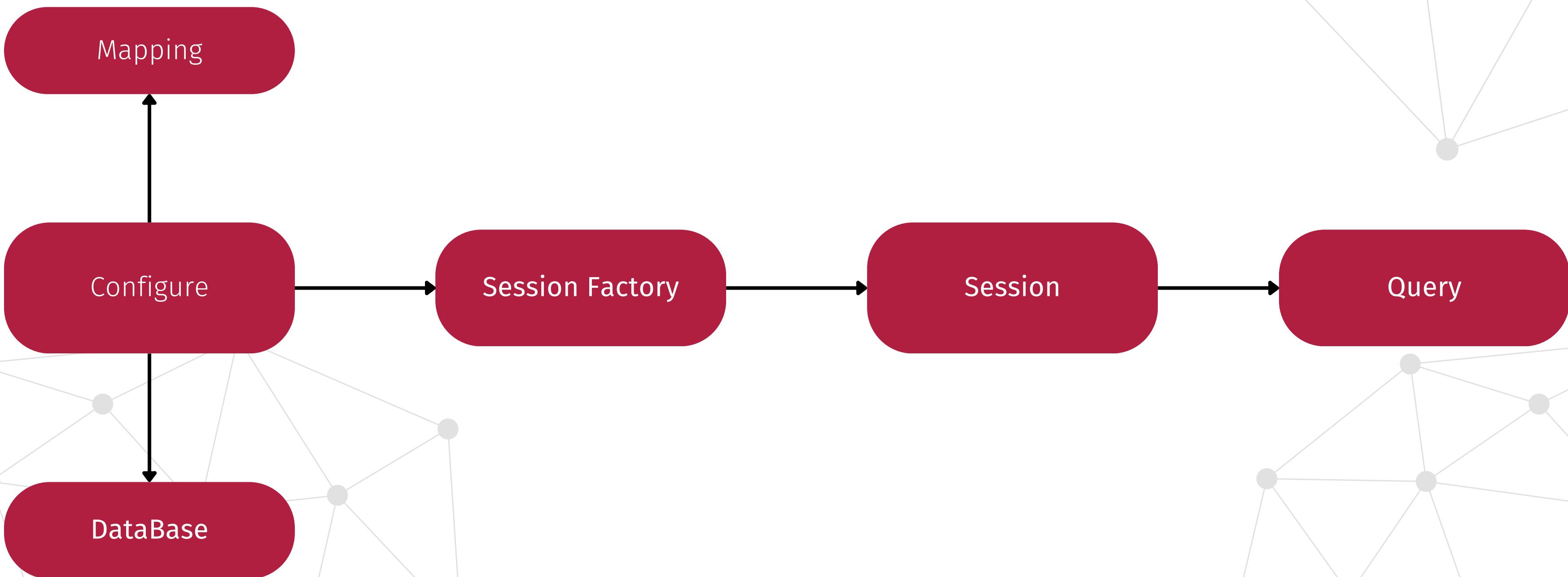
- 1 Implementation of Unit Test Cases
- 2 Hibernate integration with Spring Boot
- 3 Features of Session in Hibernate

HOW MOCKITO WORKS



HIBERNATE OVERVIEW

LESSON 7



Test Automation

JUnit

A framework of Java used to implement automation tests which gives us the project reliability

Mokito

A framework which enables to test the service with a mock objects required for the data framing and testing of JUnit test cases

Hibernate

Hibernate

It is a ORM which maps Java objects to relational DB and perform the HQL for CRUD operations

Session

12-FACTORS & DESIGN PATTERN



Factor 2: Dependencies

Note for how the Hibernate functions we use do not require Hibernate to be installed on the system running the app.

Also note how we declare our testing dependencies to only cover testing scopes.



Factor 10: Development/Production Parity

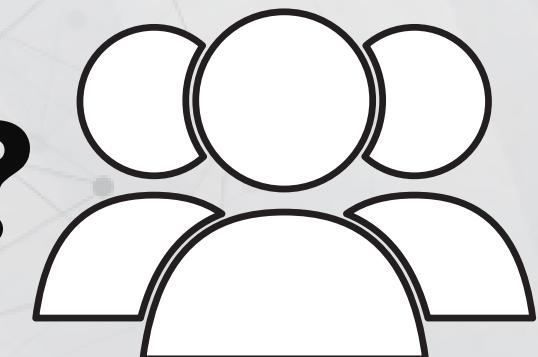
Watch for how we use mocked instances to run our actual production code during testing.



Factor 11: Logging

Take note of where we output our logs during testing - we use stdout, not a log file.

1. What Design Pattern did we use?



CODING EXERCISE #1



```
@Test  
public void testCreateAccount() throws Exception {  
    AccountRequestDto requestDto = new AccountRequestDto();  
    requestDto.setName("John Doe");  
    requestDto.setEmail("john.doe@example.com");  
    requestDto.setMobNo("1234567890");  
  
    AccountResponseDto responseDto = new AccountResponseDto();  
    responseDto.setId(1L);  
    responseDto.setName("John Doe");  
    responseDto.setEmail("john.doe@example.com");  
  
    when(accountService.createAccount(any(AccountRequestDto.class))).thenReturn(responseDto);  
  
    mockMvc.perform(post(urlTemplate: "/rest/account")  
        .contentType(MediaType.APPLICATION_JSON)  
        .content(asJsonString(requestDto)))  
        .andExpect(status().isOk())  
        .andExpect(jsonPath(expression: "$.id", is(value: 1)))  
        .andExpect(jsonPath(expression: "$.name", is(value: "John Doe")))  
        .andExpect(jsonPath(expression: "$.email", is(value: "john.doe@example.com")));  
  
    //Code Goes Here  
}
```

The code to the left is meant to run a Mockito test that creates a profile. Using Mockito methods, write one line of code that verifies if our rest API called the service function with the correct argument.

CODING EXERCISE #1



```
@Test
public void testCreateAccount() throws Exception {
    AccountRequestDto requestDto = new AccountRequestDto();
    requestDto.setName("John Doe");
    requestDto.setEmail("john.doe@example.com");
    requestDto.setMobNo("1234567890");

    AccountResponseDto responseDto = new AccountResponseDto();
    responseDto.setId(1L);
    responseDto.setName("John Doe");
    responseDto.setEmail("john.doe@example.com");

    when(accountService.createAccount(any(AccountRequestDto.class))).thenReturn(responseDto);

    mockMvc.perform(post(urlTemplate: "/rest/account")
        .contentType(MediaType.APPLICATION_JSON)
        .content(asJsonString(requestDto)))
        .andExpect(status().isOk())
        .andExpect(jsonPath(expression: "$.id", is(value: 1)))
        .andExpect(jsonPath(expression: "$.name", is(value: "John Doe")))
        .andExpect(jsonPath(expression: "$.email", is(value: "john.doe@example.com")));
}

//Code Goes Here
}
```

verify(accountService,times(1)).createAccount(any(AccountRequestDto.class));

CODING EXERCISE #2



Below is a section of our pom.xml file that declares dependencies for Spring Starter Security, Hibernate Validator, and JUnit testing. Which part of this code is incorrect?

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>8.0.0.Final</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <scope>runtime</scope>
</dependency>
```

CODING EXERCISE #2



Below is a section of our pom.xml file that declares dependencies for Spring Starter Security, Hibernate Validator, and JUnit testing. Which part of this code is incorrect?

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>8.0.0.Final</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <scope>runtime</scope>
</dependency>
```

JUnit is declared as a runtime dependency, when it should be a testing dependency.
`<scope>testing</scope>`

SETTING UP YOUR ENVIRONMENT



The screenshot shows a GitHub repository page for 'SWEUpskilling Cre8tivePilot'. The main content is a README.md file with the following text:

Don't configure the repository With this shell application in this project

Prompt: The purpose of this pre-assessment, is to get an understanding of your knowledge in SpringBoot. The pre-assessment is 30 minutes. Do not use Google or AI tools for this part. The instructions to send the code are below.

You will be creating a profile application using your current knowledge of Spring SpringBoot.

1. Set your time for 30 minutes.
2. Download the ZIP file with the project.
3. Create a new repository (using any service you want), and add the downloaded folders into the repository.
4. Open the "ProfileService" project in your IDE.
5. Add an optional dependency for lombok, from the group projectlombok
6. Add a dependency for hibernate-validator version 8.0.0.Final, from the group hibernate-validator
7. Fill out the Profile entity class and AuthRecord entity class using the comments in the shell, showing best practices for efficient programming
8. Fill out the controller class using the comments in the shell, following best practices for 12-factor app
9. Create a MySQL database. Create databases named ProfileData and SongData. Connect those databases to the ProfileService application.
10. Create a test that verifies the REST API endpoint that you created.
11. Run the application.
12. When time stops at 30 minutes, push to the repository.

Longer Pre-assessment

During the course you had 10 mins to do a pre assessment. This is a take home assessment that will also help. We look forward to finding you sharing your results

Prompt:

The purpose of this pre-assessment, is to get an understanding of your knowledge in Spring Boot. The pre-assessment is 30 minutes. We ask that you avoid using Google or AI tools for this part. The instructions to send the code are below.



THANK YOU



Crash Course

We will see you tomorrow