WELCOME BACK
LESSON 7

Spring/Spring Boot Crash Course

For TD Bank

MEET YOUR CRASH COURSE TEAM

TANGY F.
CEO

HAL M.
DEVELOPER

WILLIAM D.
DEVELOPER

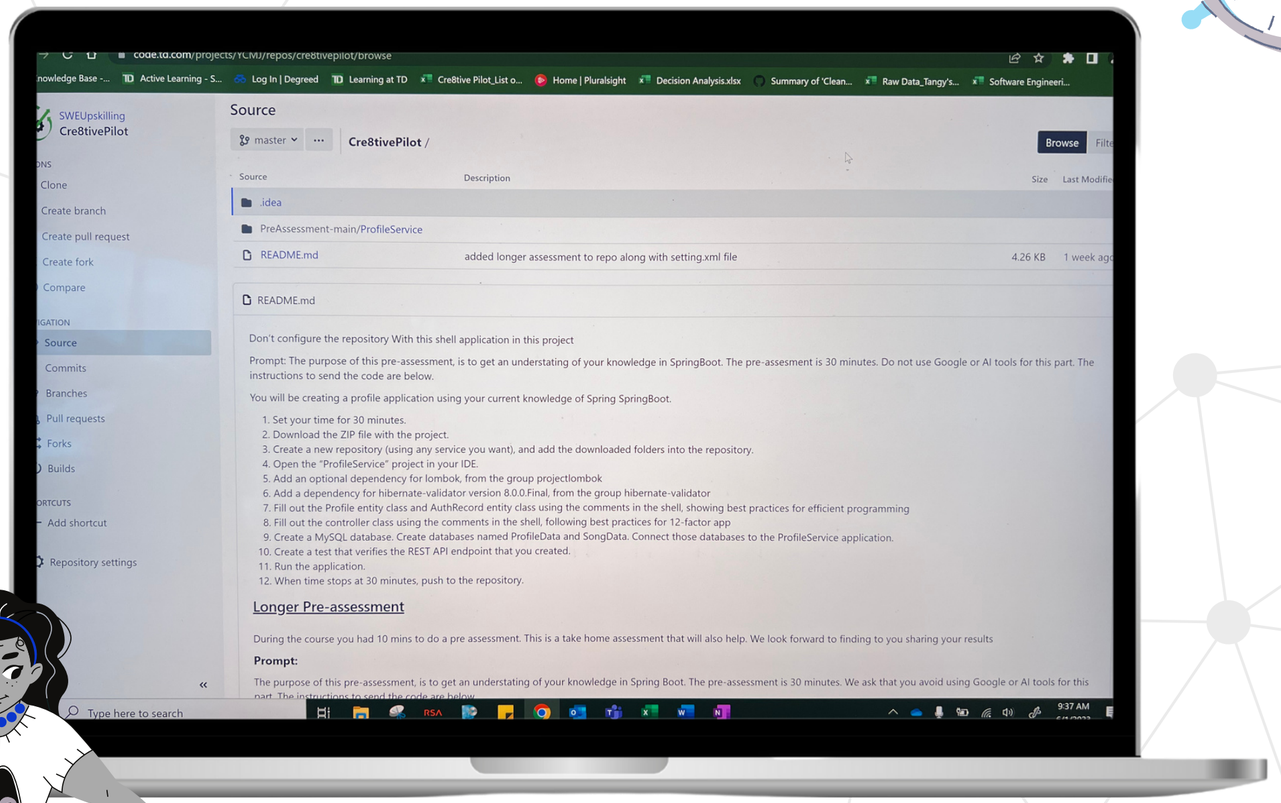
LESSON 7

AGENDA

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

Creative Devs. Software

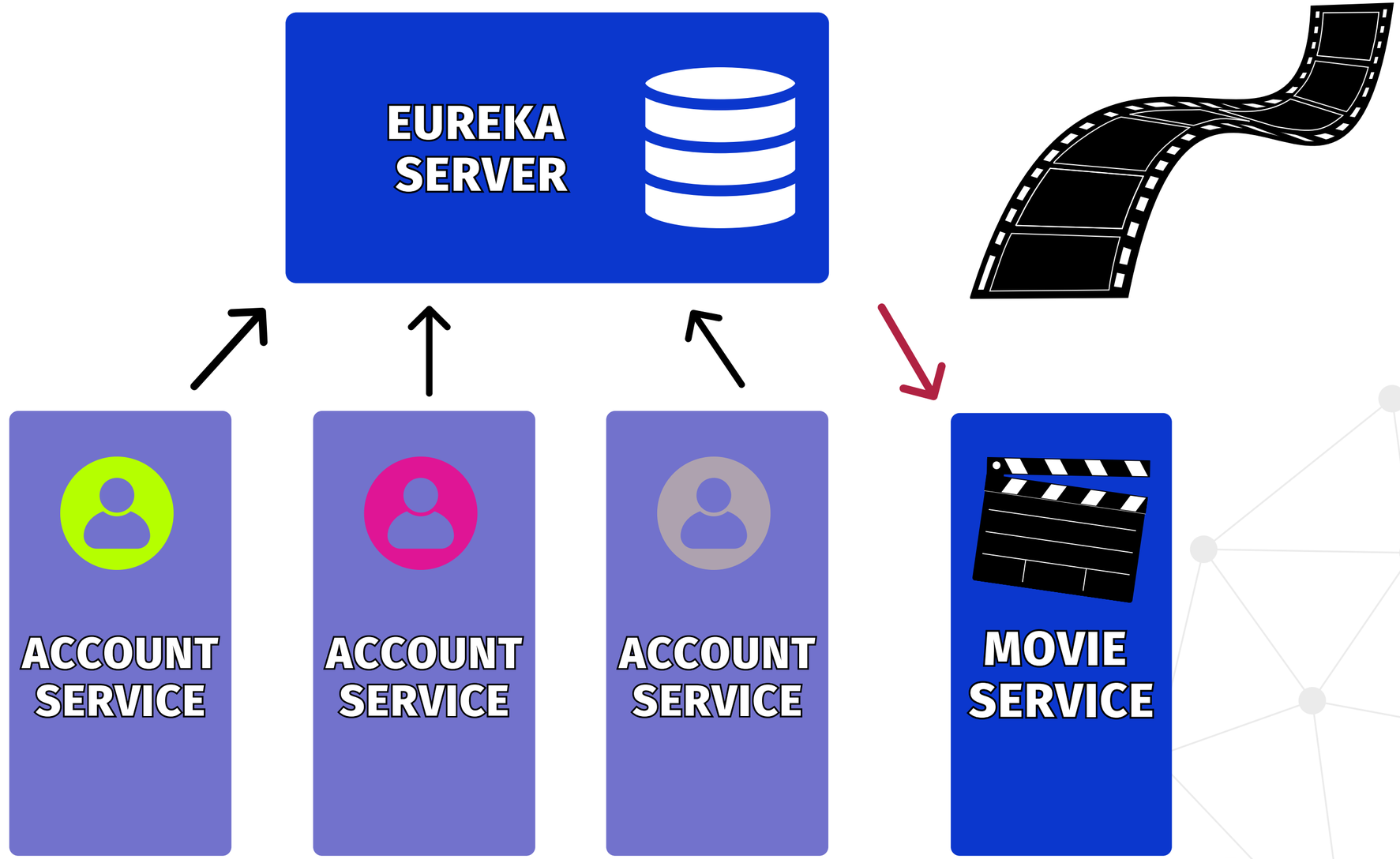
UPDATE ABOUT YOUR YOUR ENVIRONMENT SET UP



Creative Devs. Software

WE ARE BUILDING

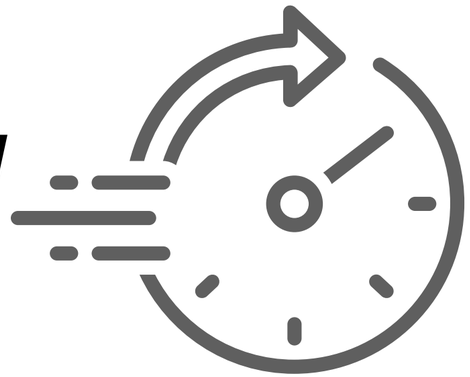
BITE SIZE
MOVIE REVIEW APP



Creative Devs. Software

WELCOME TO LESSON 7

Rapid Review Trivia

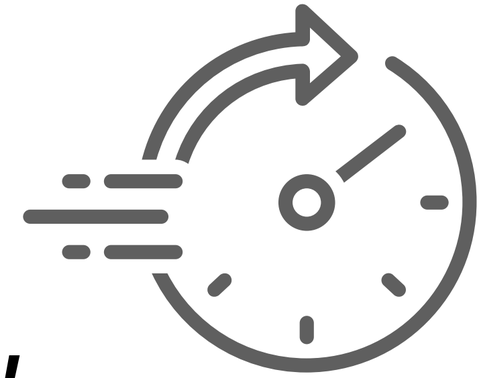


RAPID REVIEW

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

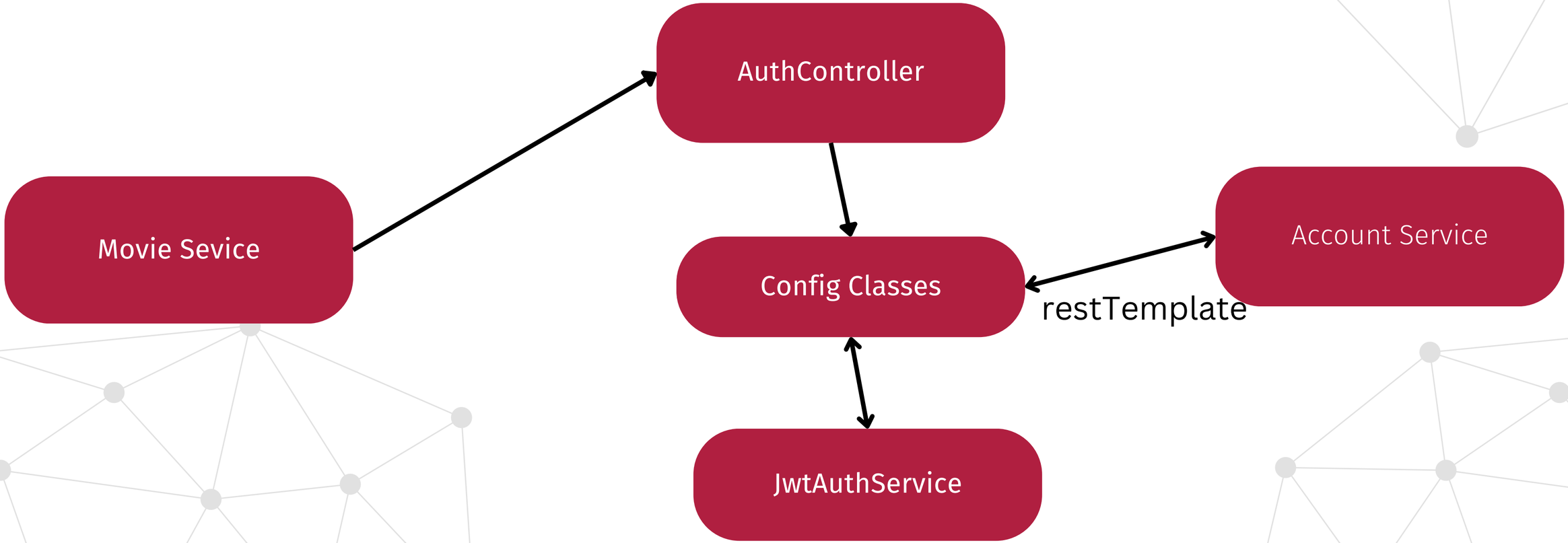


Rapid Review

JWT

THE FLOW OF MOVIE & ACCOUNT SERVICE WITH JWT

REVIEWING LESSON 6



LESSON 7

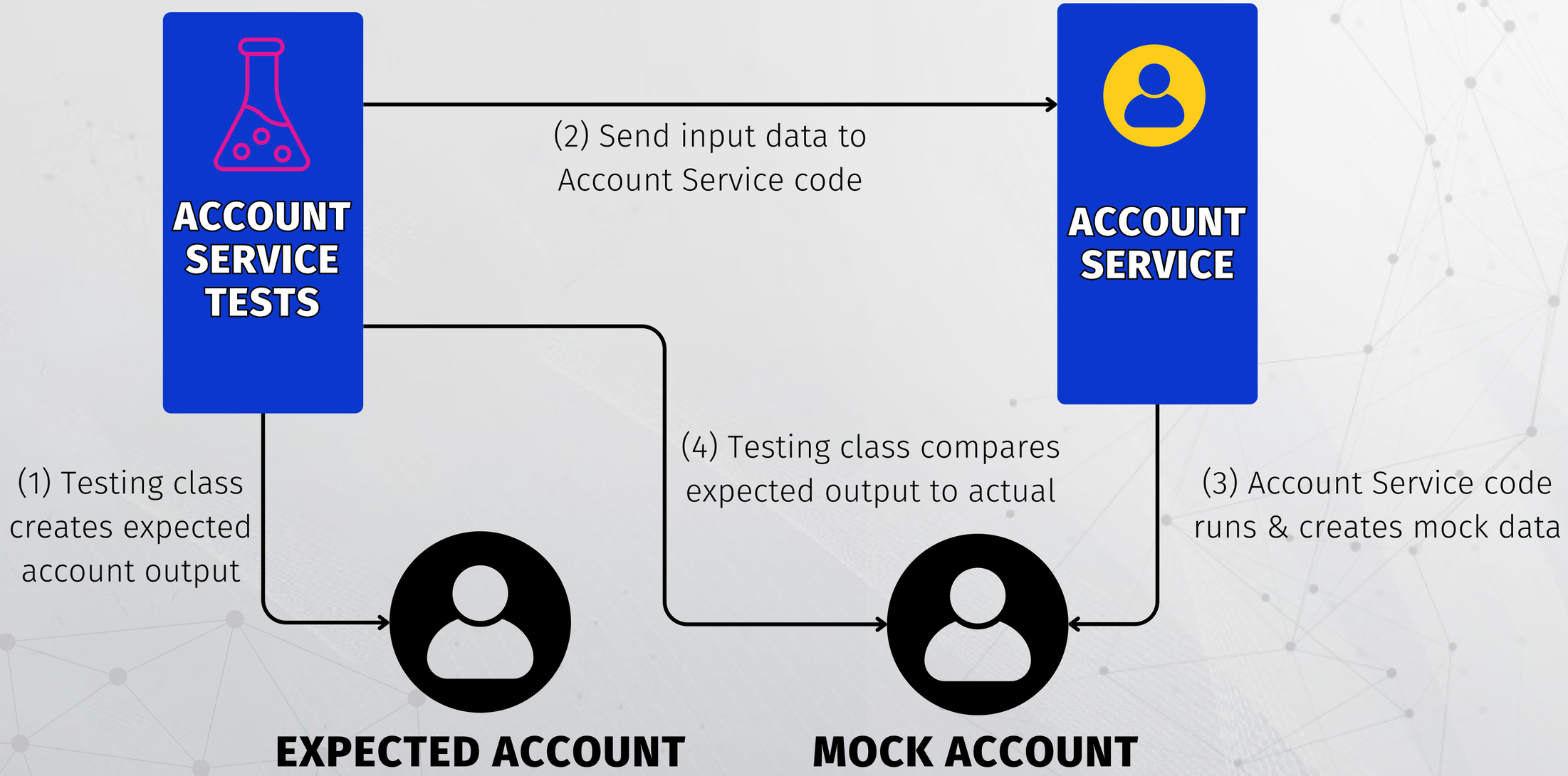
Unit Testing & Hibernate

LESSON 7

UNIT TESTING & HIBERNATE

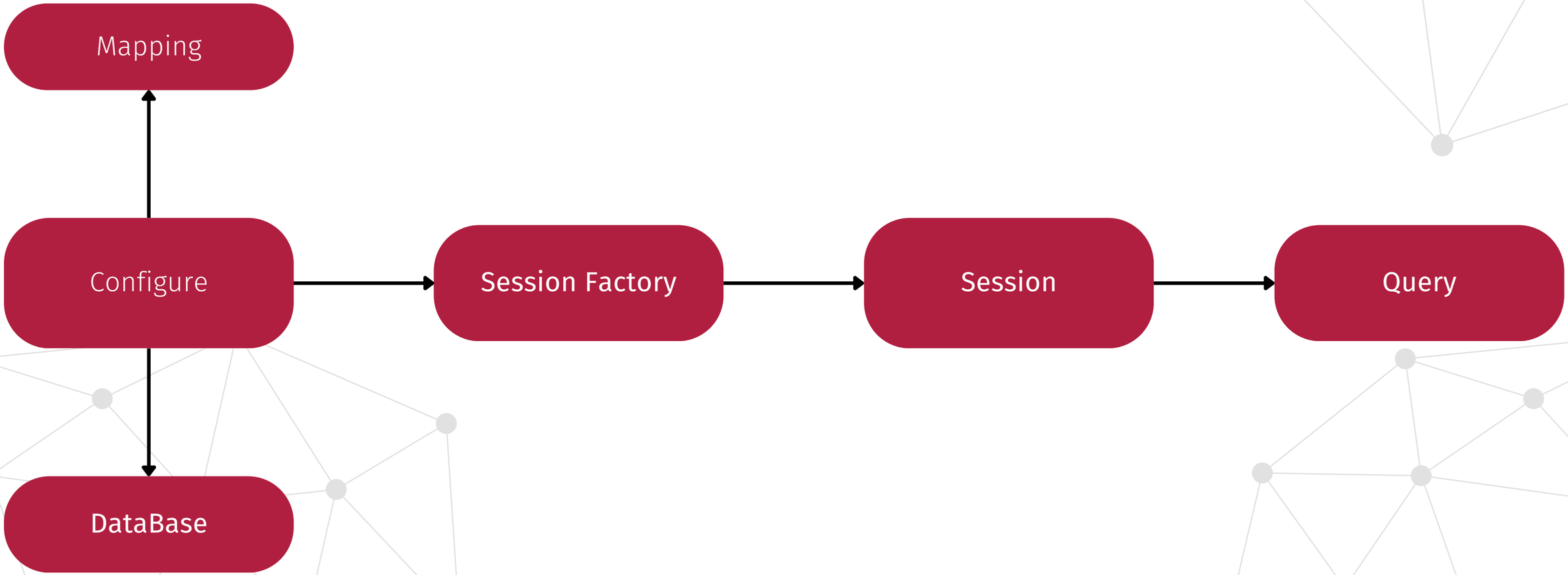
1. Implementation of Unit Test Cases
2. Hibernate integration with Spring Boot
3. Features of Session in Hibernate

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

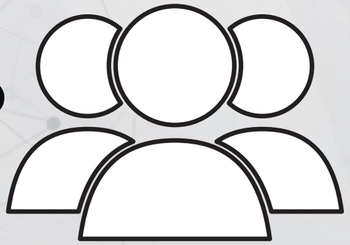
Watch for how we use mocked instances to run our actual production code during testing.

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>
```

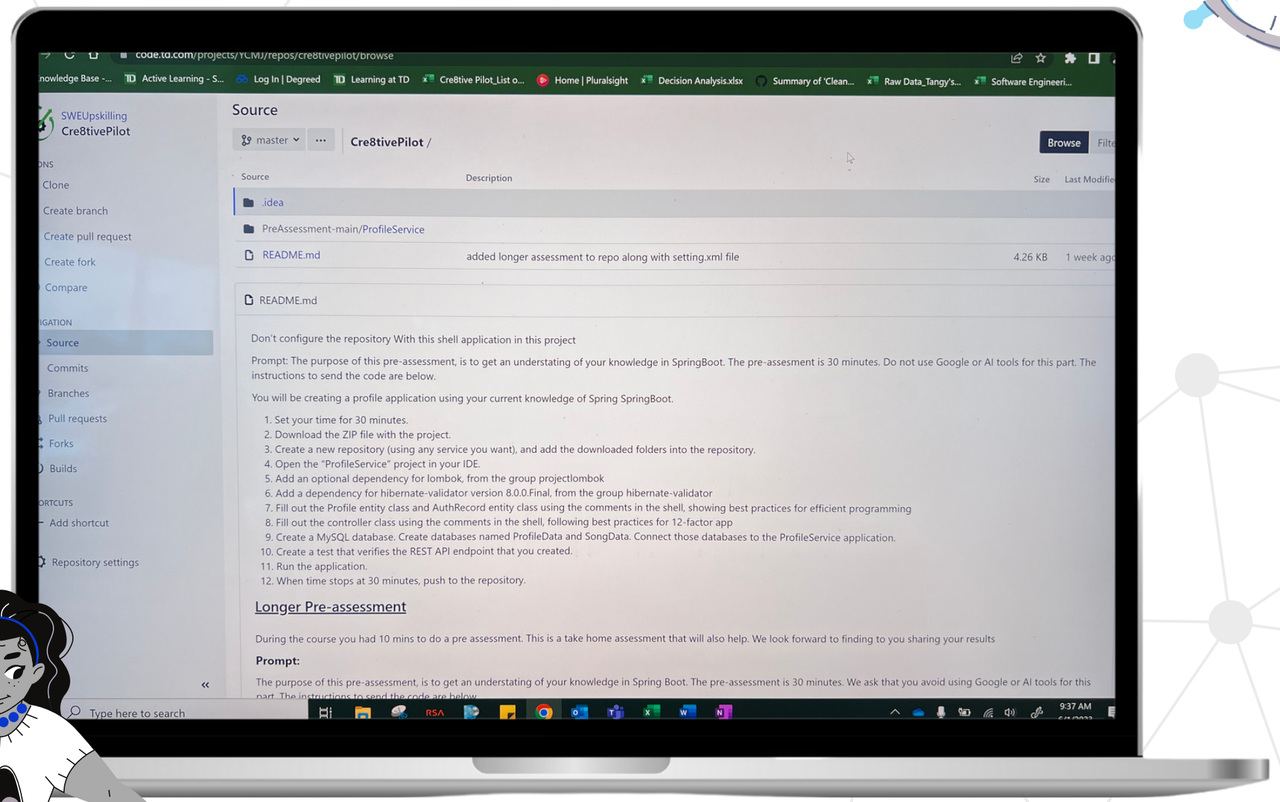
CODING EXERCISE #2

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



Cre8tive Devs. Software

THANK YOU

Crash Course

We will see you tomorrow