Slide Note

Credit Card

- How is this credit card different from others?

Menu

00:01:00:00:18

01/ 18

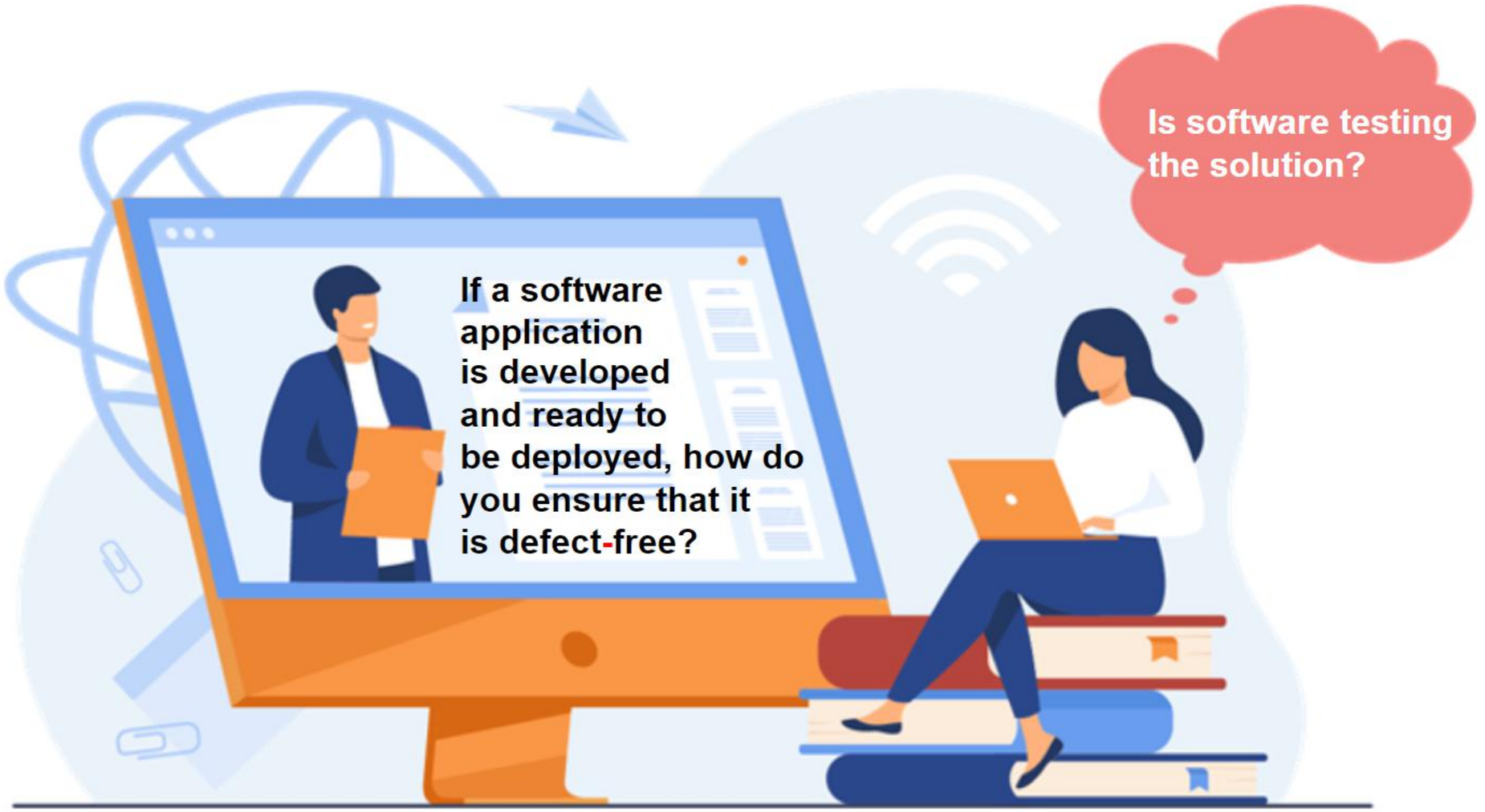
Slide Note

Display Screen

- How do these problems impact consumers and businesses?
- When should these defects be identified?
- What steps must companies take to avoid such occurrences?

Slide Note

Product Testing



Slide Note

Think and Tell

- The controller layer is dependent on HTTP methods. Do you think you should use actual HTTP methods while testing?
- To test controller layer methods, can you use the actual object of the service layer or a substitute?
- Should you use the actual server for testing the controller layer?

Slide Note

Test RESTful Services at Controller Layer by Using Testing Tools (JUnit, Mockito)

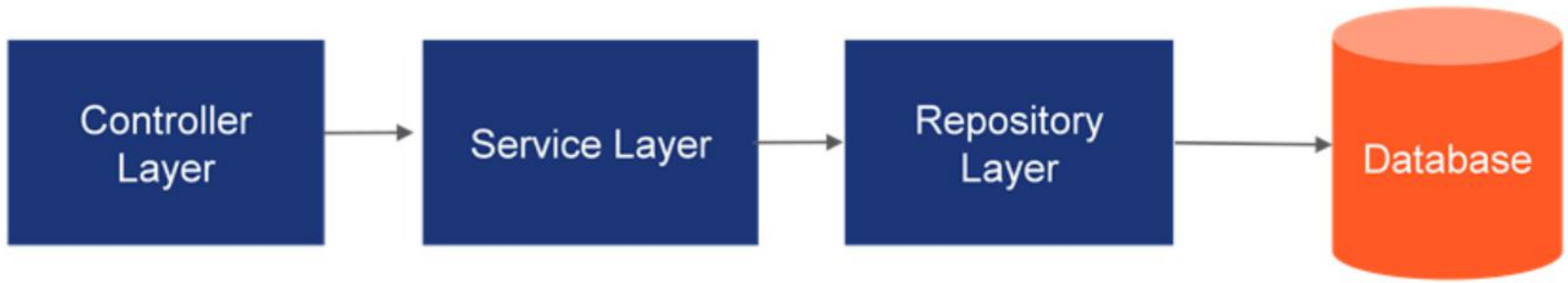
Slide Note

Learning Objectives

- Explore MockMvc
- Implement controller layer testing

Slide Note

Layers in Spring Boot Application



Slide Note

Quick Check

To unit test the controller layer, which layer should we mock?

1. Service Layer

2. Repository Layer

Slide Note

Quick Check: Solution

To unit test the controller layer, which layer should we mock?

1. Service Layer

2. Repository Layer

Slide Note

Explore MockMvc

Slide Note

What Is MockMvc?

- MockMvc has been around since Spring 3.2.
- MockMvc is mainly used to test the code of the controller layer.
- It provides a powerful way to mock Spring MVC for testing MVC web applications.
  Through MockMvc, you can send mock HTTP requests to a controller and test how the controller behaves without running the controller within a server.
- MockMvc testing is needed for:
  - Content negotiation headers: To produce only application/JSON content.
  - Response codes: To check if the response code matches the expected one.
  - JSON serialization/deserialization: To validate JSON is deserialized and correctly converted into the response body.

Slide Note

Implementing Controller Layer Testing

Slide Note

Testing the Controller Layer

- The test class needs to be annotated with @ExtendWith(MockitoExtension.class).
- The @ExtendWith annotation integrates the Spring TestContext Framework into JUnit 5's Jupiter programming model.
- To write the test cases, you need MockMvc object.
- The controller layer has unit test cases, and it will mock the service layer.
- Here, he service layer is annotated with @Mock and the controller layer is annotated with @InjectMock.

```
@ExtendWith(MockitoExtension.class)
class CustomerControllerTest {


    private MockMvc mockMvc;
    @Mock
    private CustomerServiceImpl customerService;
    @InjectMocks
    private CustomerController customerController;
```

Slide Note

Testing the Controller Layer

- Here, the `when` method of Mockito is used to mock the service layer and set an expectation.
- In this code, the expectation is that when `customerService saveCustomerDetails()` method is called with any customer object as an argument, then it will return the saved object.
- By using the `mockMvc` object, you call "`perform`" method to make a mock call to the API post method.
- Post takes the API endpoint as an argument. Use the same endpoint that was set in RequestMapping in the controller.
- Here, `(.andExpect(status().isCreated())` is a basic check for 201 status.

```
@PostMapping("customer")
public ResponseEntity<?> saveCustomer(@RequestBody Customer customer) throws CustomerAlreadyExistsException {
    try {
        customerService.saveCustomerDetail(customer);
        responseEntity = new ResponseEntity(customer , HttpStatus.CREATED);
    } catch (CustomerAlreadyExistsException e) {
        throw new CustomerAlreadyExistsException();
    }
    catch (Exception e)
    {
        responseEntity = new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }

    return responseEntity;
}
```

```
@Test
public void givenCustomerReturnSavedCustomer() throws Exception {
    when(customerService.saveCustomerDetail(any())).thenReturn(customer1);
    mockMvc.perform(post( urlTemplate: "/api/v1/customer")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(jsonToString(customer1)))
            .andExpect(status().isCreated()).andDo(MockMvcResultHandlers.print());
    verify(customerService,times( wantedNumberOfInvocations: 1)).saveCustomerDetail(any());
}
```

Slide Note

Testing the Controller Layer

A supermarket chain needs to perform an analysis of customer data to determine how it can expand its business to new locations.
They have outsourced the work to ABC Company to create the application. ABC Company has completed the application and tested the REST endpoints using Postman.
Now they want to automate the testing purpose so they will start with the repository, service, and controller layers.

Check the solution here.

DEMO

Slide Note

Quick Check

Which annotation is added on the controller layer while testing it?

1. @InjectMocks
2. @Mock
3. @Autowired

Slide Note

Quick Check: Solution

Which annotation is added on the controller layer while testing it?

1. @InjectMocks
2. @Mock
3. @Autowired