Service Layer Mocks

Java Accelerator 7

Lesson 3.3

Learning Objectives

01 Refactor applications to use a service layer.

02 Use JUnit for unit and integration testing.

03 Test applications using mocks.

Service Layer

Service Layer

The Java object model of simple CRUD applications can be tightly coupled with the database structure.

Create	`INSERT INTO table (column1, column2, column3)`
Read	`SELECT * FROM table`
Update	`UPDATE table SET column1 = VALUE WHERE id = 1`
Delete	`DELETE FROM table WHERE id = 5`

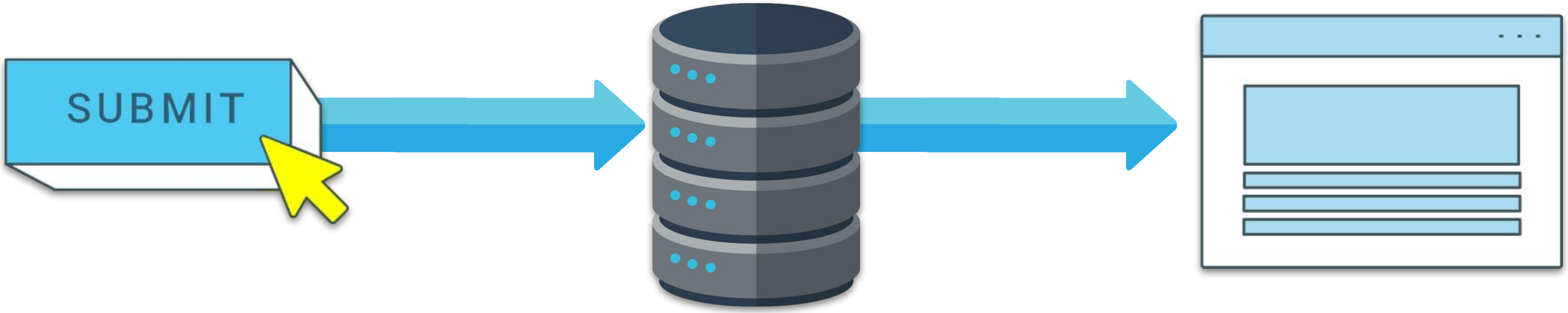
Service Layer

This model might be okay for simple applications, but it can be limiting for more complex applications.

Users interact with a **Controller**

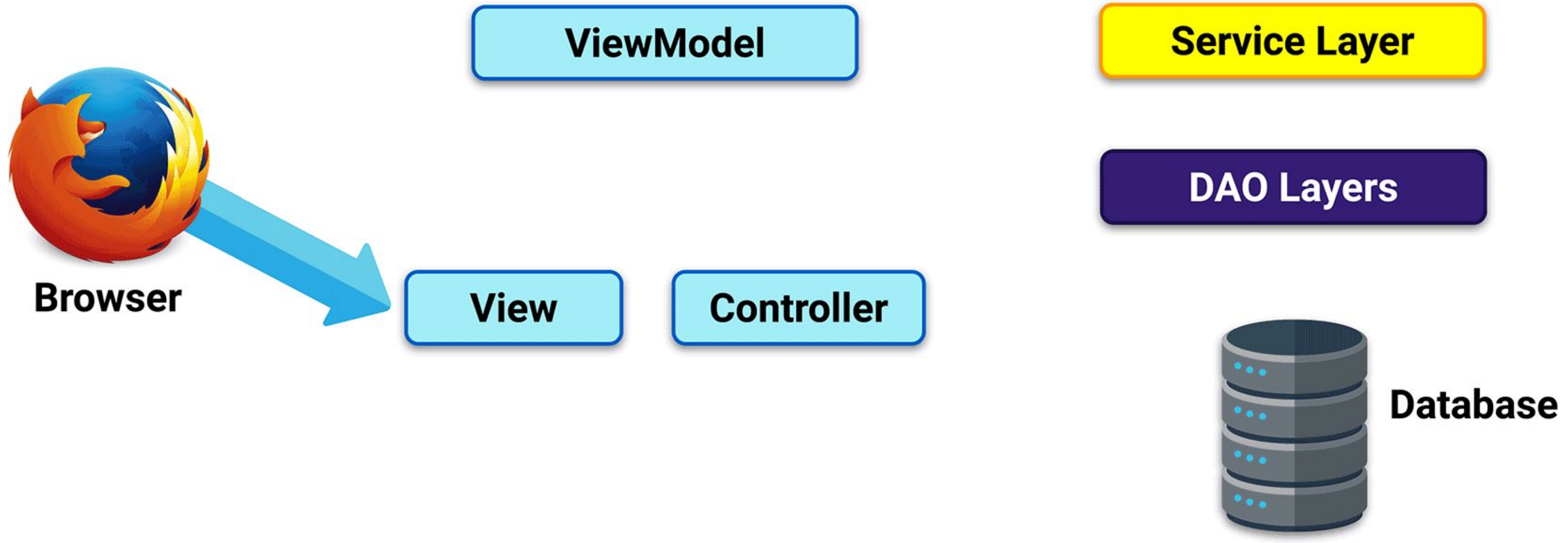
to make changes to a **Model**

which is then shown in the View.



Service Layer

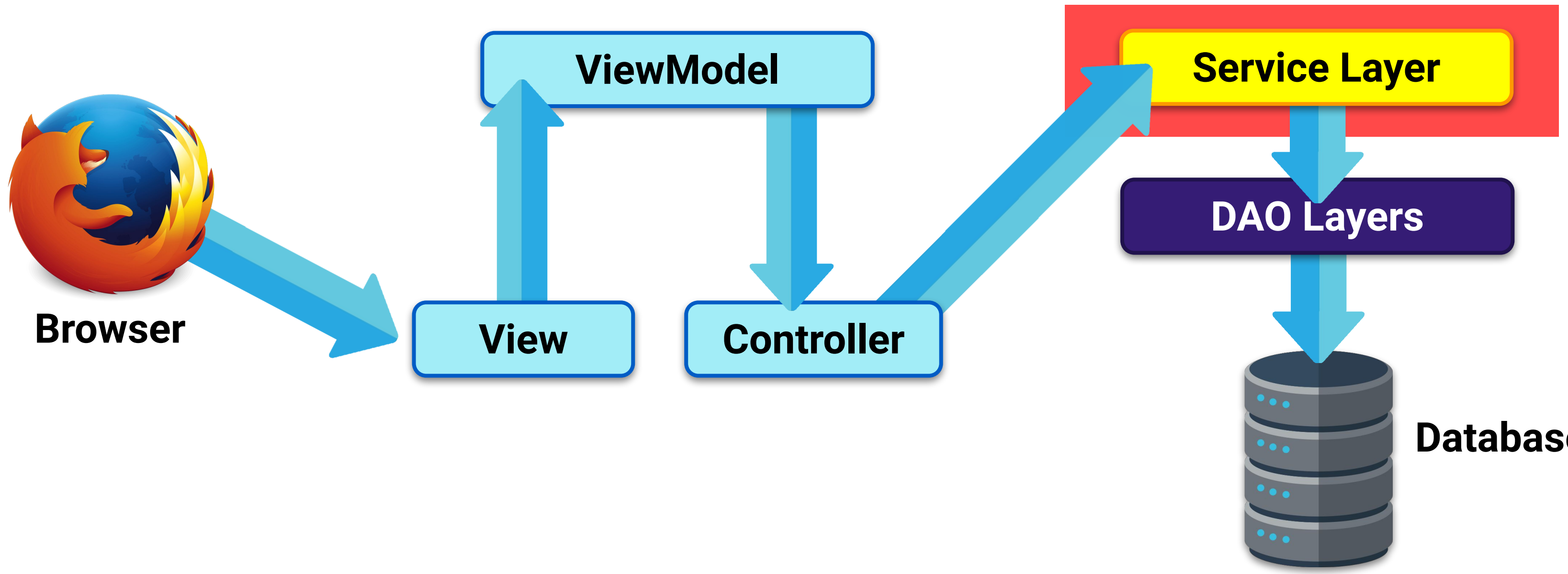
The **service layer** can validate the `ViewModel` data, apply other business rules/logic, and translate between the `ViewModel` and the DTOs (Data Transfer Objects) associated with the database model.



This pattern—`ViewModel`, controller, service layer, DTO, and DAOs—works quite well for many applications of different sizes.

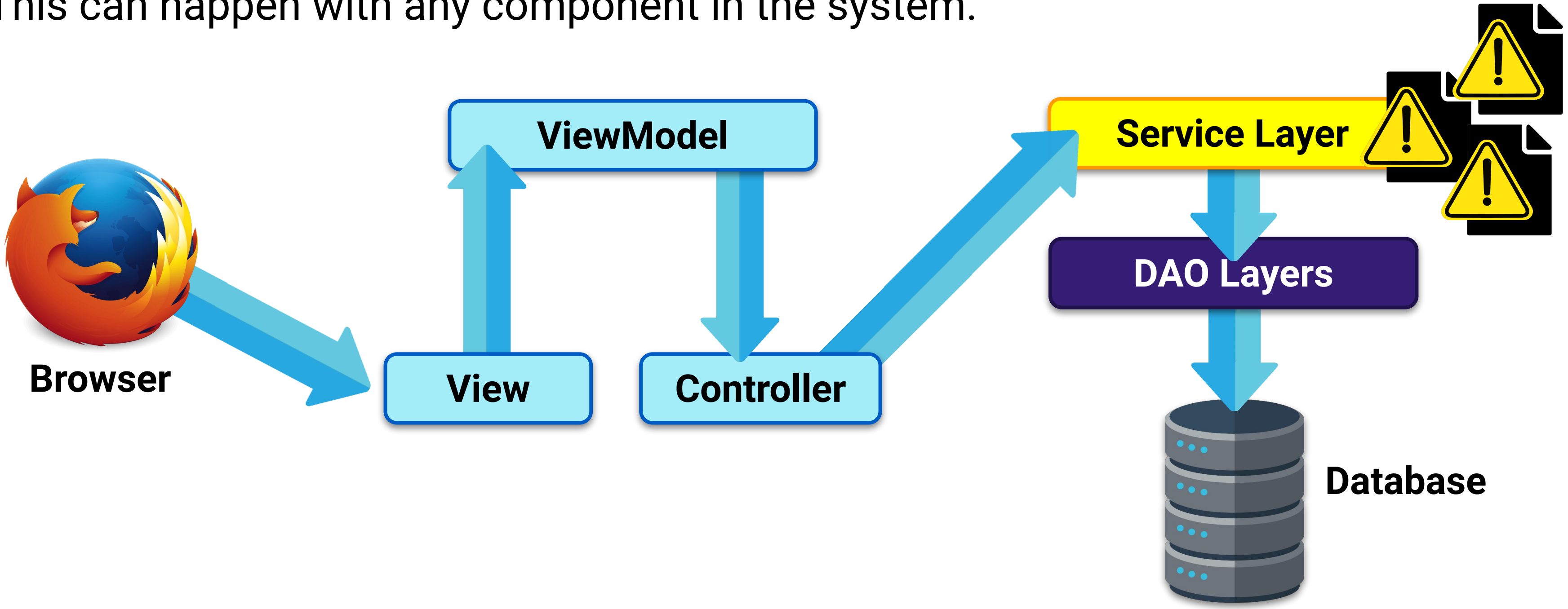
Potential Pitfalls of the Service Layer

The service layer components grow in an uncontrolled manner as the application grows, eventually becoming an unruly mess that has too many responsibilities.



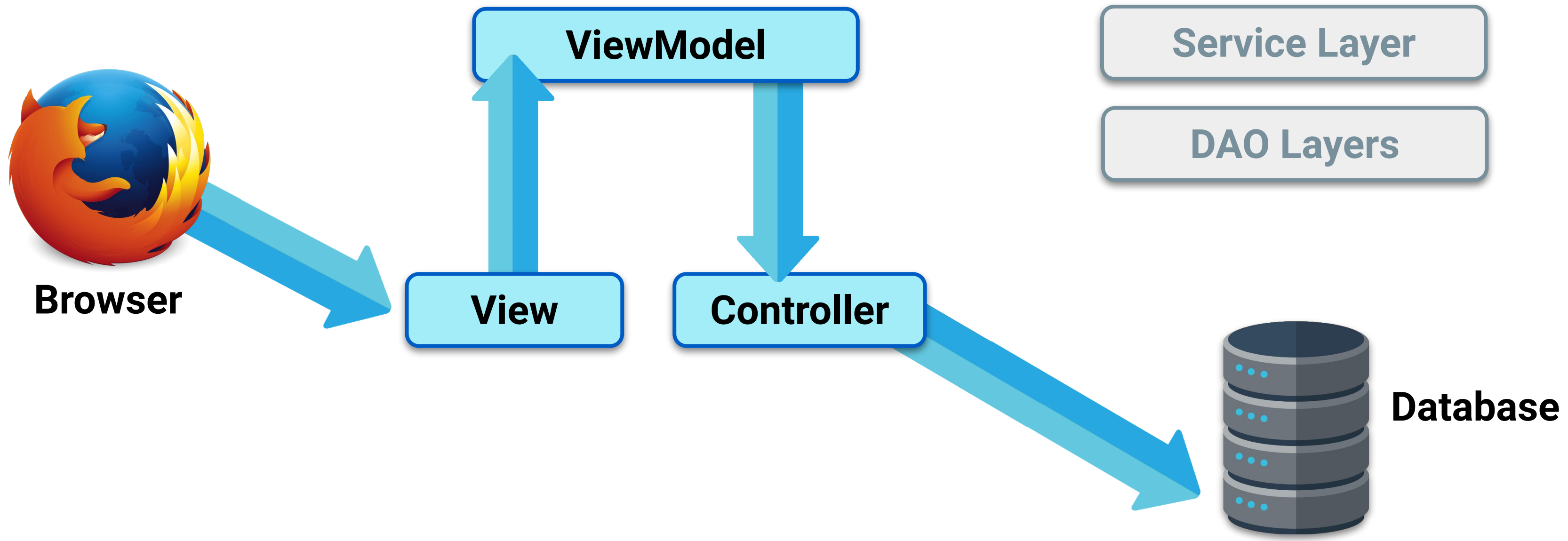
Potential Pitfalls of the Service Layer

Business rules can become scattered around or copied around the service layer if you aren't disciplined with your development practices. This can happen with any component in the system.



Potential Pitfalls of the Service Layer

If we don't use the service layer, the Java object model remains directly coupled to the database structure. The degree to which this is a problem depends on the application requirements and complexity.



Time to Code

Service Layer Implementation

Suggested Time:

Using Mock Objects

Using Mock Objects

Complex systems need several levels of testing:

- Unit testing
- Integration testing
- System testing
- Performance testing
- User acceptance testing

Using Mock Objects

About the tests:

01

Most of these levels of testing involve testing the dependencies, interaction, and configuration of the various parts of the system.

02

Unit testing is different: we want to remove the external dependencies and just concentrate on the code under test.

The Code Under Test

We want to only deal with the code under test for the following reasons:

- This allows us to write our tests narrowly, without needing to consider the behavior of external dependencies.
- Some external dependencies are not **deterministic**. In other words, we can't predict the returned values from interactions with these systems—making it almost impossible to create good tests.
- Some external dependencies are slow to respond or respond asynchronously. We want unit tests to run quickly and efficiently.
- It might be very expensive in configuration or compute time to run external systems.

Using Mock Objects

Using mock objects for unit testing allows us to address these issues:

01

We use mock objects to simulate the behavior of an external dependency.

02

We configure mock objects so that they give known responses to a particular set of parameters.

03

These calls are quick and deterministic.

Mockito

Mockito

Mockito is one of the most popular and easy-to-use Java mocking frameworks—and the framework we use in this course.

Project status

Please see the release notes page.

Mockito

The Mockito library is already included in the start dependencies for the Spring Boot data-driven web service projects.

Mockito has a lot of features, but we will concentrate on the mocking functionality—specifically these methods:

- `mock`
- `doReturn`
- `when`

Mockito

The Mockito framework can be used:

- To create mocked implementations of interfaces and classes.
- To mock individual methods tailored to individual test cases.
- To ensure code test coverage using Spy functionality.
- To inject mocked versions of dependencies into contexts such as Spring Boot.
- In conjunction with other testing frameworks such as JUnit and MockMVC.

Mockito

Time to <code>