Java Spring Boot

Java Spring Boot

What is Spring (Boot)?

- Popular framework for Java Enterprise development
- Implements a model for modular code development
  - Declarative programming
  - Dependency injection => loose coupling
- Minimize boilerplate code
- Good testing integration
- Many modules provide additional functionality, such as security and DB integrations

> “**Spring makes Java**
> **Simple**
> **Modern**
> **Productive**”

Java Spring Boot

What is it not?

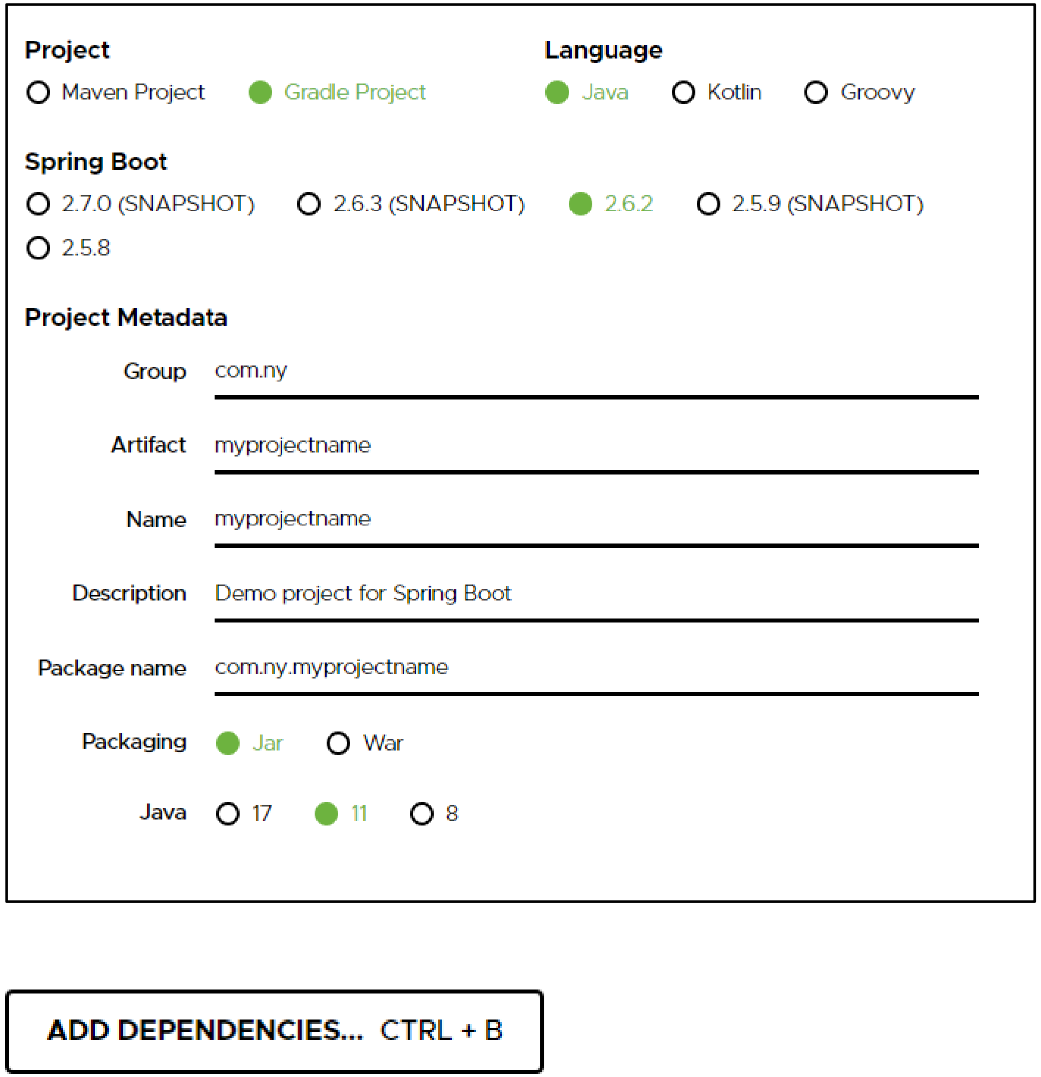
- It’s a Framework, not a new programming language!

➔ Its still just Java with additional features

Setup

Spring Initializr

- https://start.spring.io/
- Dependencies
  - Spring Web
  - Spring Data
  - PostgreSQL
  - Optional: Lombok
- Download and open in IntelliJ



Project

○ Maven Project ● Gradle Project

Language

● Java ○ Kotlin ○ Groovy

Spring Boot

○ 2.7.0 (SNAPSHOT) ○ 2.6.3 (SNAPSHOT) ● 2.6.2 ○ 2.5.9 (SNAPSHOT)
○ 2.5.8

Project Metadata

Group: com.ny

Artifact: myprojectname

Name: myprojectname

Description: Demo project for Spring Boot

Package name: com.ny.myprojectname

Packaging: ● Jar ○ War

Java: ○ 17 ● 11 ○ 8

Gradle

The modern build tool for Java

- Automates the compilation and building of Java code
- **build.gradle**: build script. configure build process, such as dependencies
- Plugins add functionality (e.g. testing)

- Alternative: Maven

Spring Vocabulary

- Inversion of Control
- Dependency Injection
- Beans
- Annotations

Dependency Injection

Spring Vocabulary

- **Problem**: We don’t want to deal with all the background utility classes our App might need

Dependency Injection

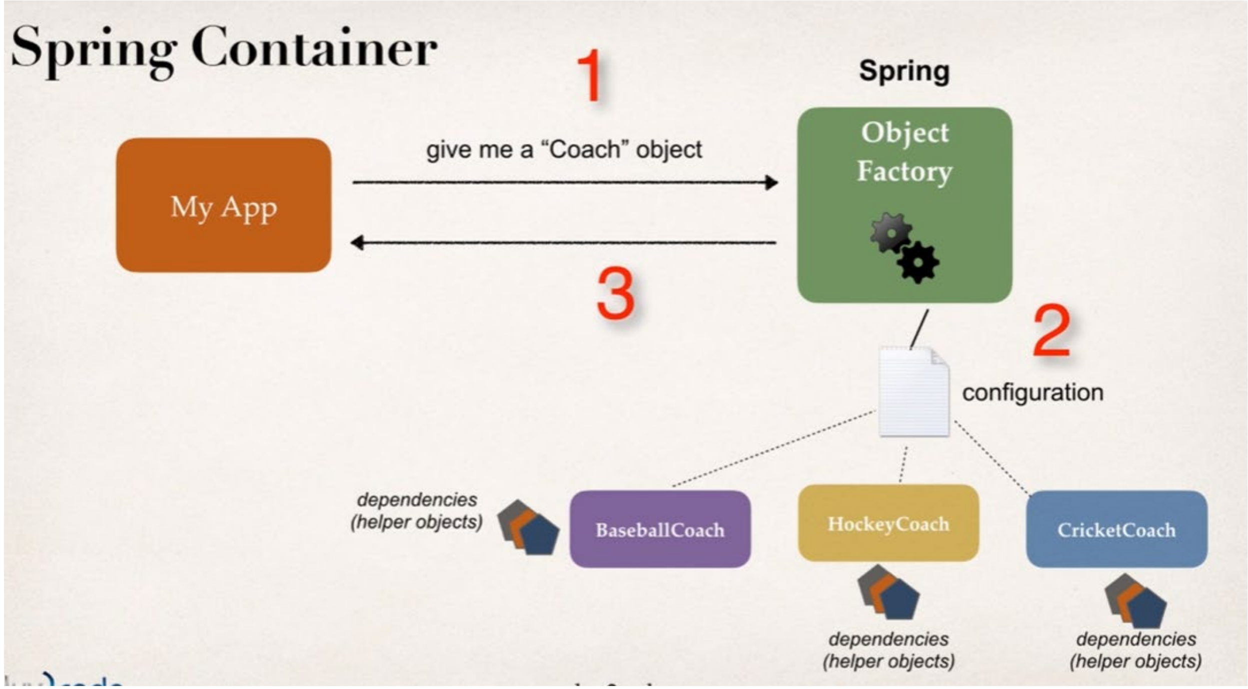
What is it?

- **Idea**: Search and “inject” objects that provide a necessary function instead of declaring them
- Connecting objects with other objects (“injecting”) is done by an **assembler** in the background

Dependency Injection

What is it?

- **Idea**: Search and "inject" objects that provide a necessary function instead of declaring them



Inversion of Control

What is it?

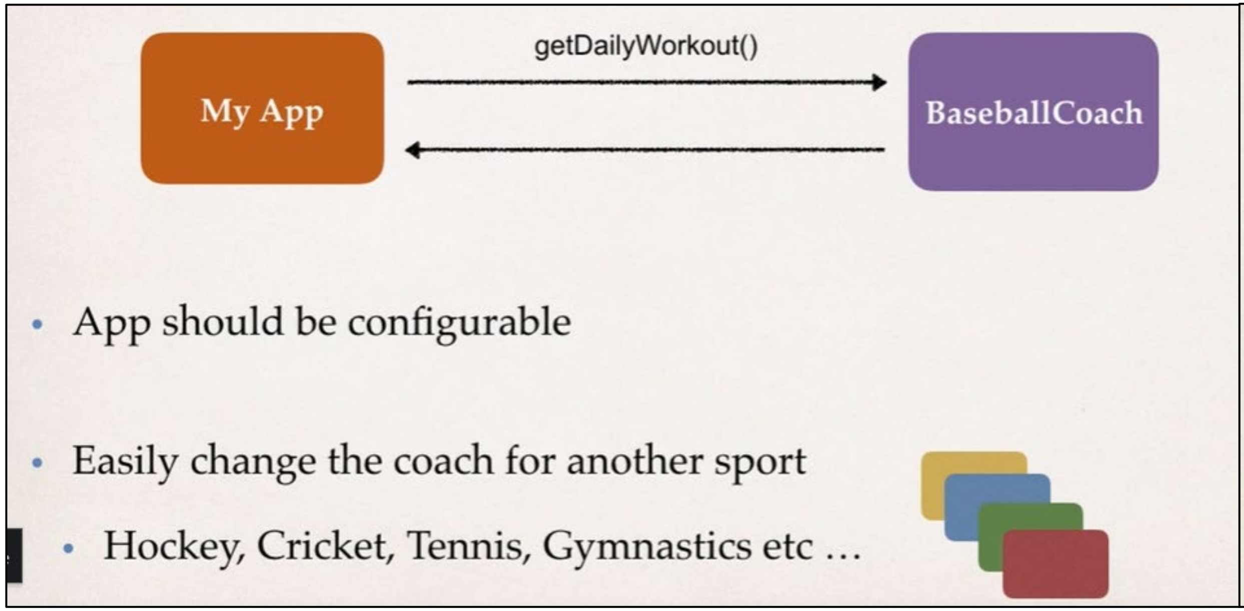
- **Idea: Outsourcing the construction and management of dependency objects**
  - Not the programmer is responsible for the handling of dependencies, but Spring
- This ensures **low coupling** of classes
  - Decoupling the execution of a task from its implementation
  - Making it easier to switch between different implementations
  - Greater modularity of a program
  - Easier to test a program

Inversion of Control

By Dependency Injection

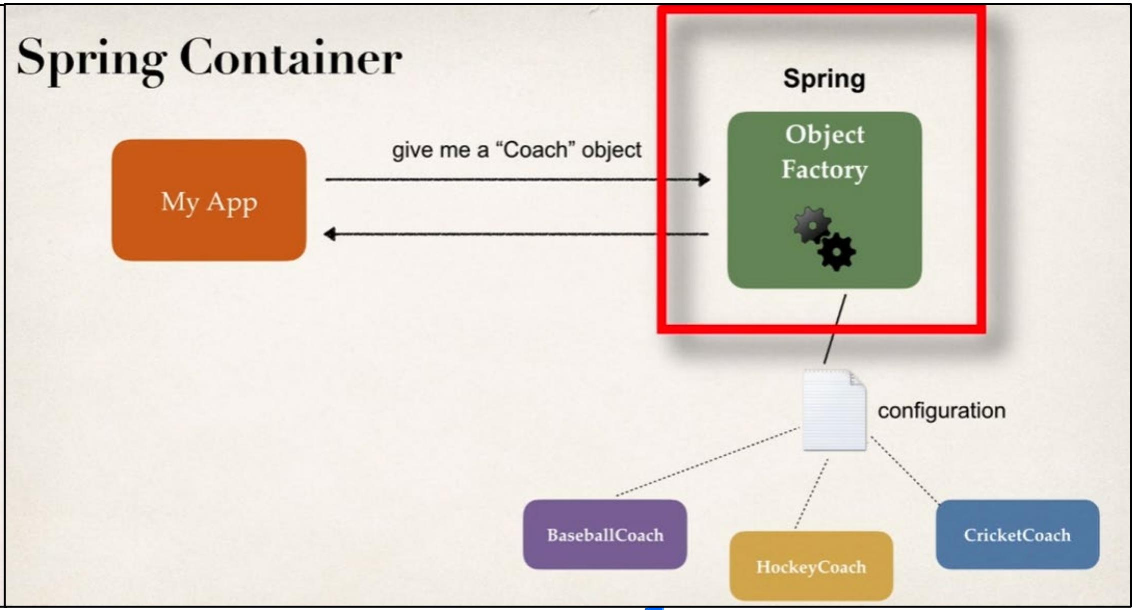
- Outsourcing the construction and management of objects
  - Handled by a «Object Factory»

Without IoC



- App should be configurable
- Easily change the coach for another sport
  - Hockey, Cricket, Tennis, Gymnastics etc …

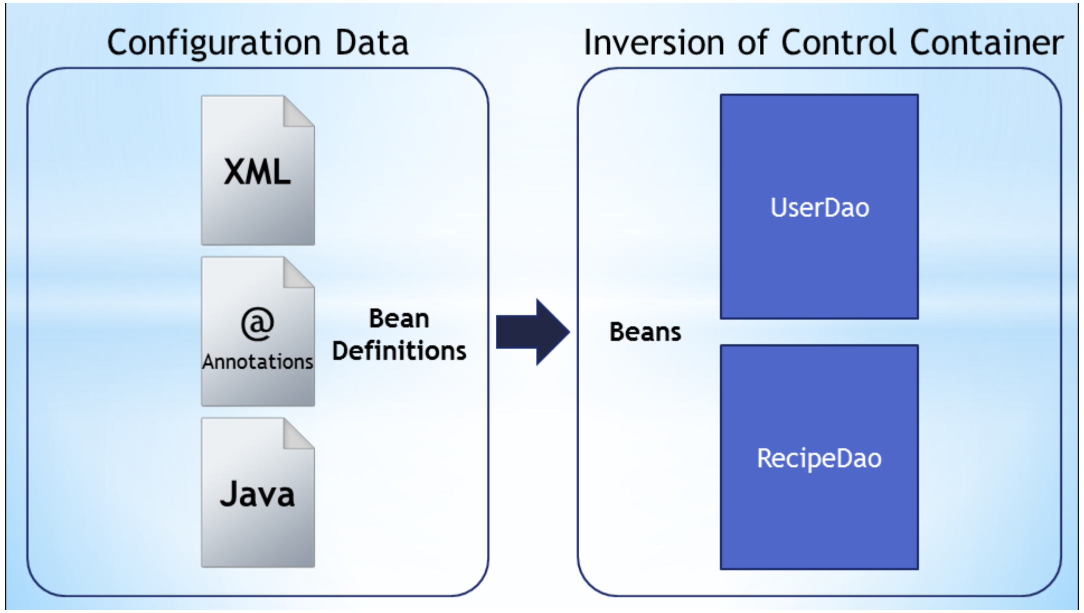
With IoC



Spring «Bean»

What is it?

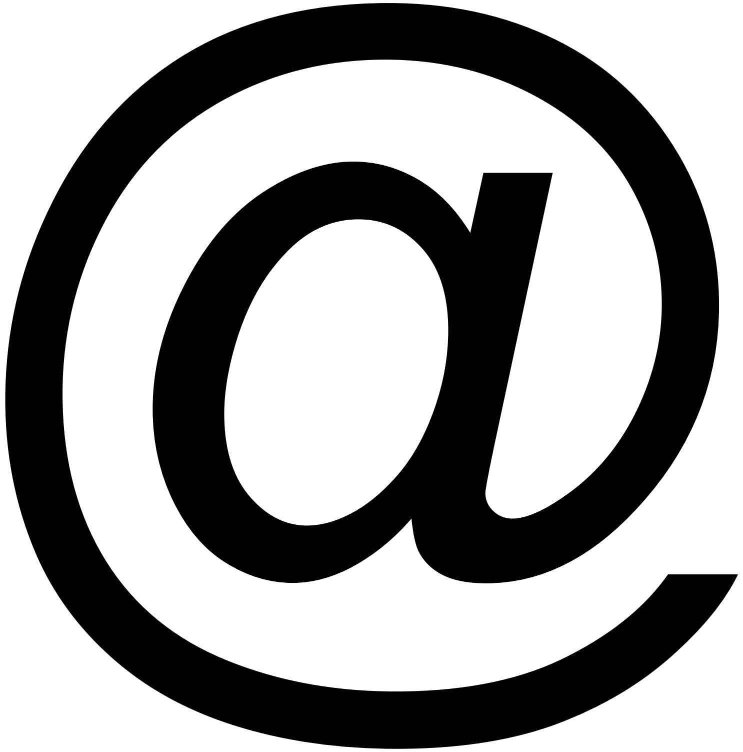
- A "Spring Bean" is simply a **Java object that is handled by Spring**
- When Java objects are created by the Spring Container, then Spring refers to them as "Spring Beans"



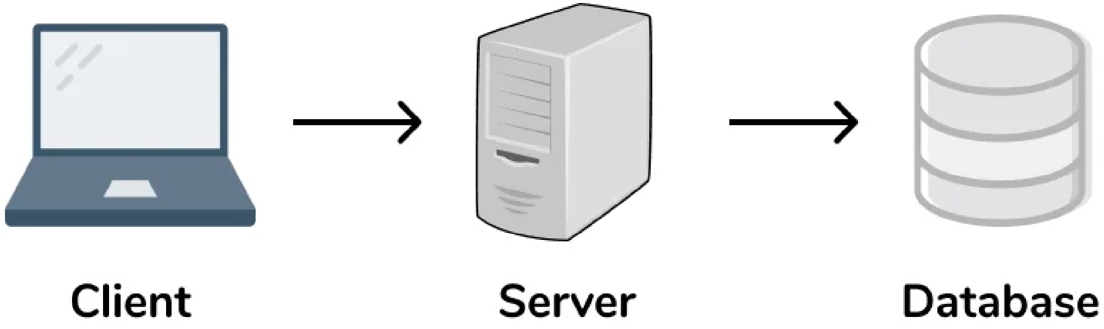
Java Annotations

For Dependency Injection

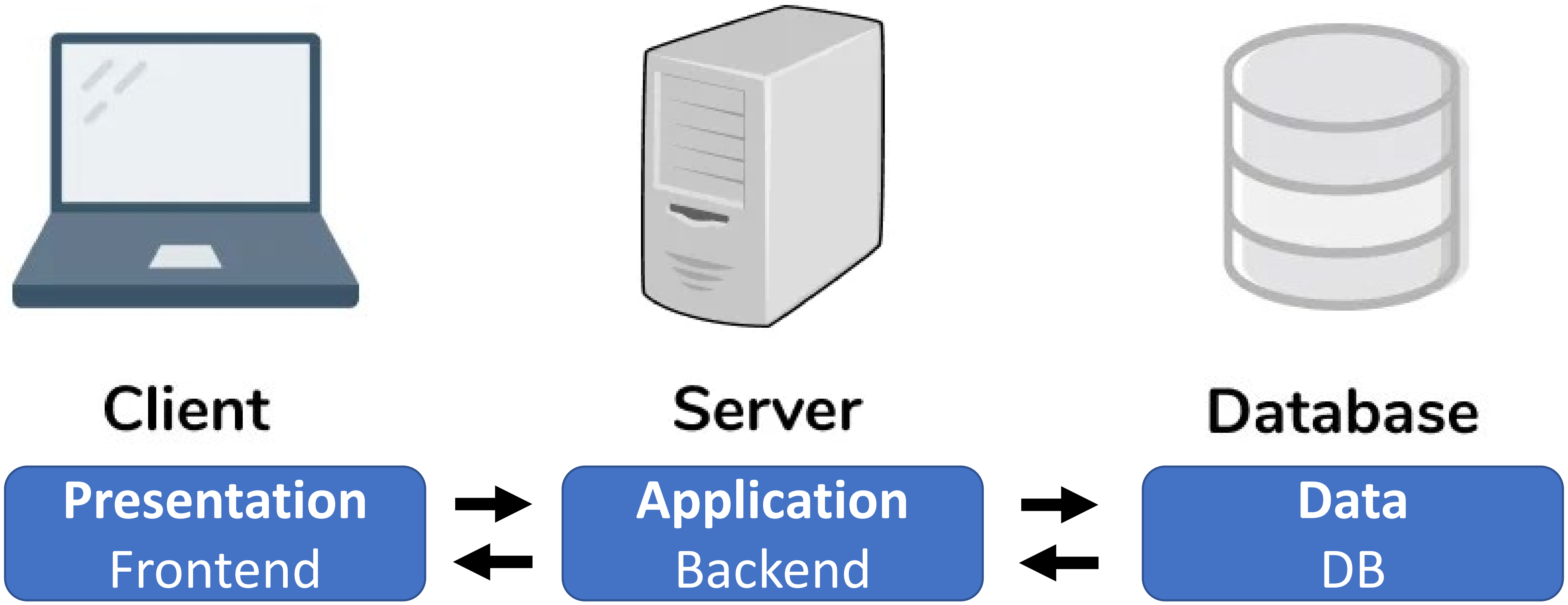
- Act as Labels/Markers added to java classes, methods and variables and **provide metadata about the class and expand it with additional functionality**
- Implemented as functions, that take classes/methods/etc. as input and return a modified version.
- Processed at compile-time OR run-time



3-Tier Architecture

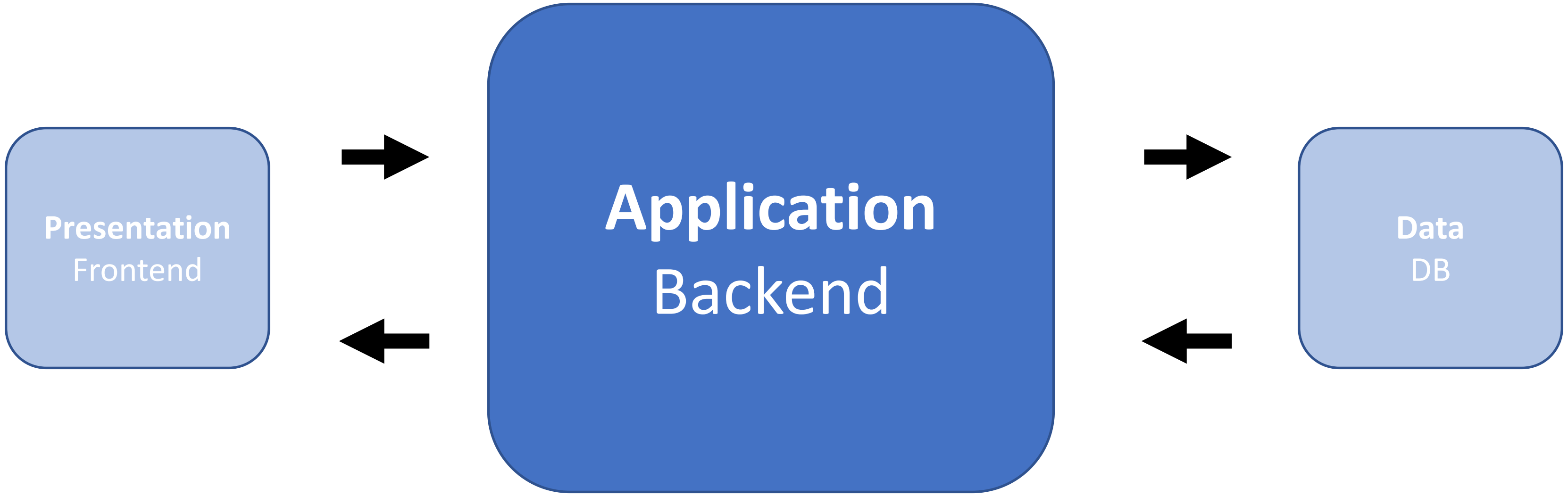


3-Tier Structure of a Full Stack Application



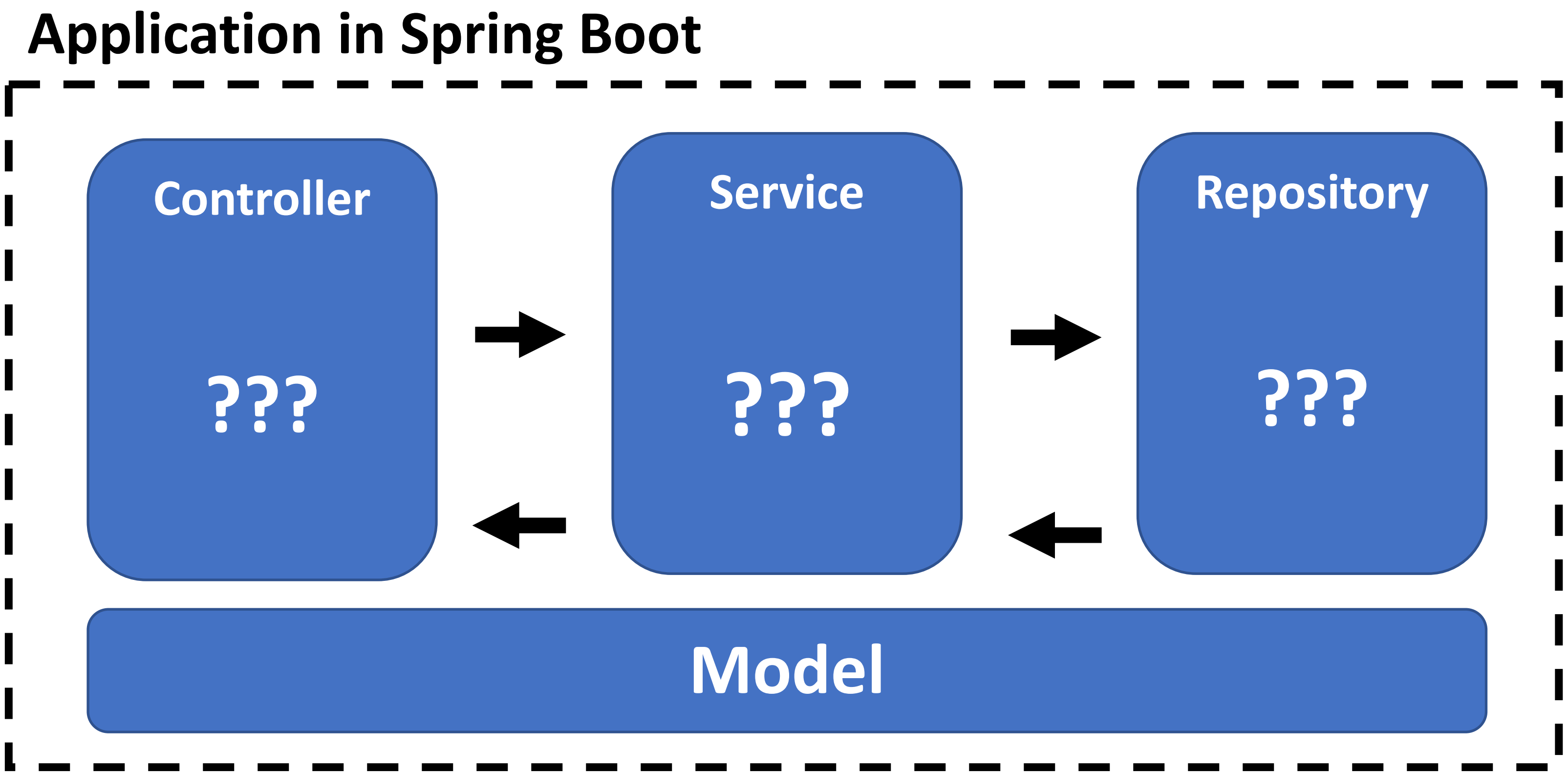
NOSERYOUNG

3-Tier Structure of a Full Stack Application



NOSERYOUNG

3-Tier Structure of a Spring Backend



Application in Spring Boot

Controller

???

Service

???

Repository

???

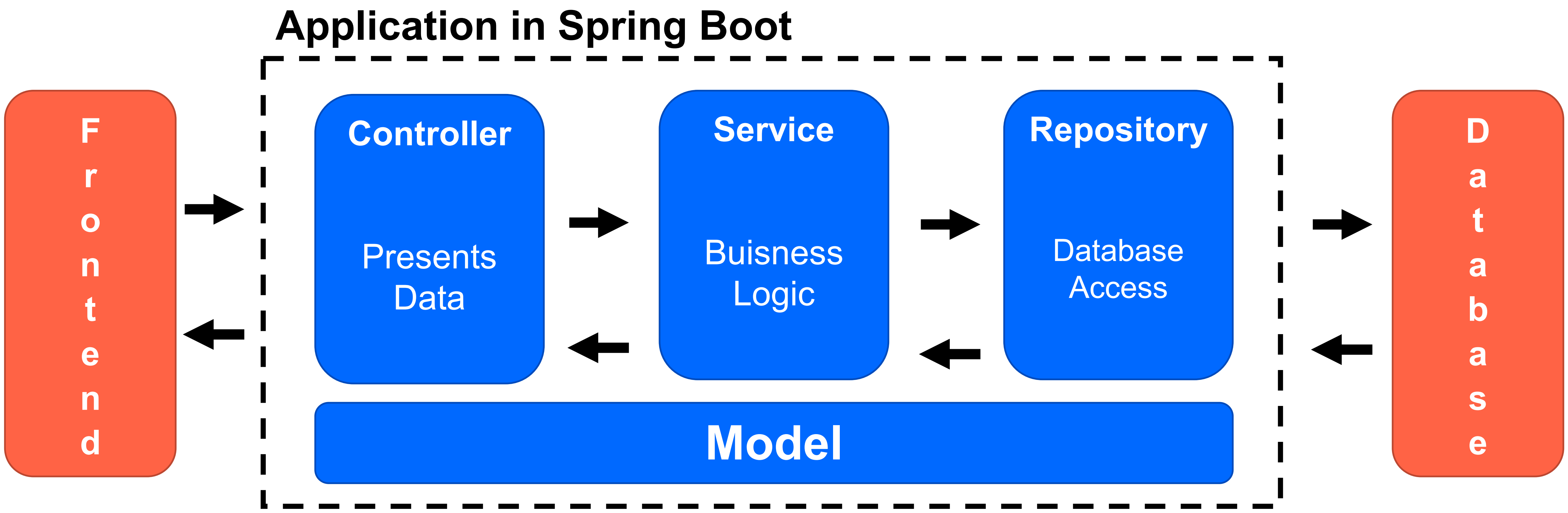
Model

3-Tier Structure of a Spring Backend

Help

- https://spring.io/learn
- https://www.baeldung.com/
- https://stackoverflow.com/

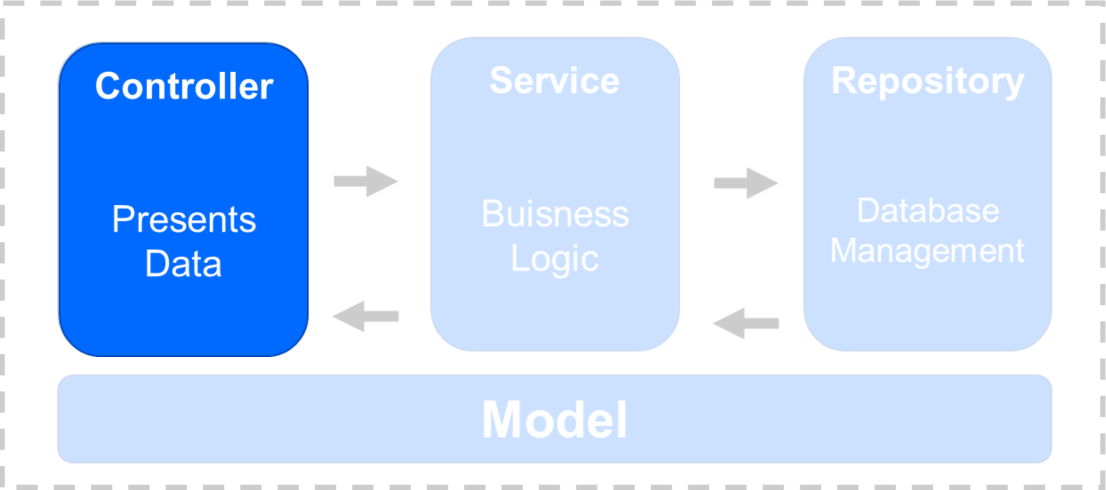
3-Tier Structure of a Spring Backend



NOSERYOUNG

Controller Class

- **Implements REST endpoints**
- Returns a **HTTP(S) response** to the frontend
- **Uses services** to generate the response data

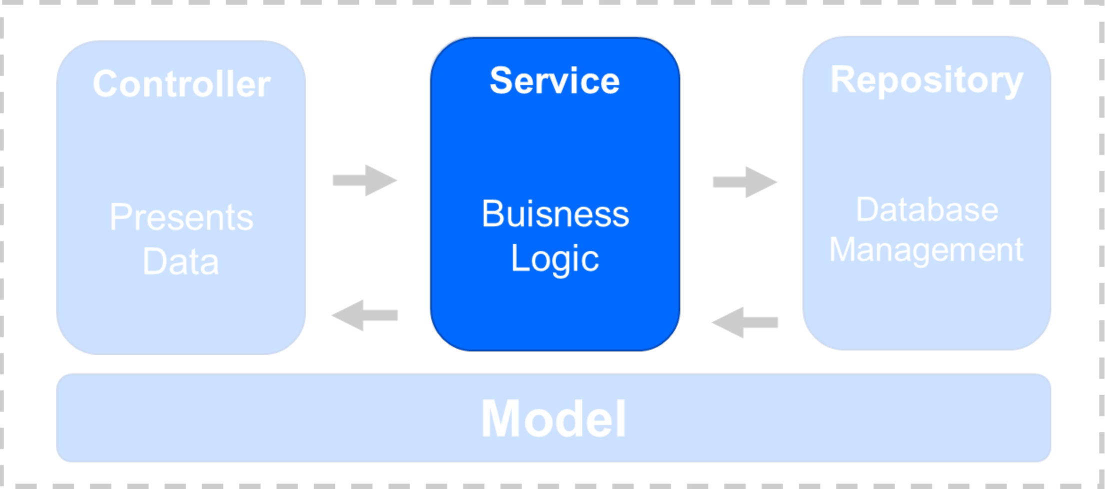


Service Class

- Contains **business logic**
- **Uses repositories** to gather necessary data
- Can interact with other services

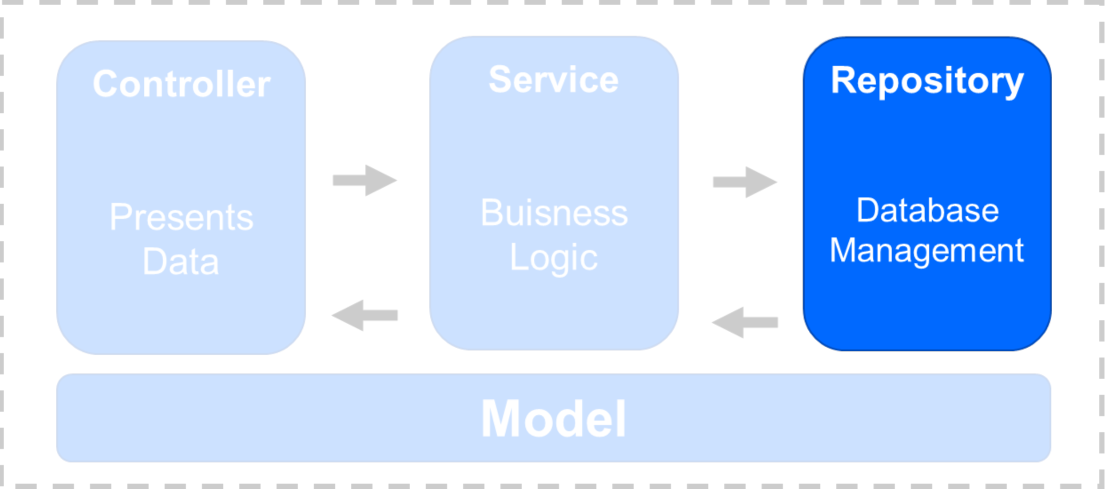
Business Logic:

Encodes the real-world business rules that determine how data can be created, stored, and changed.



Repository Class

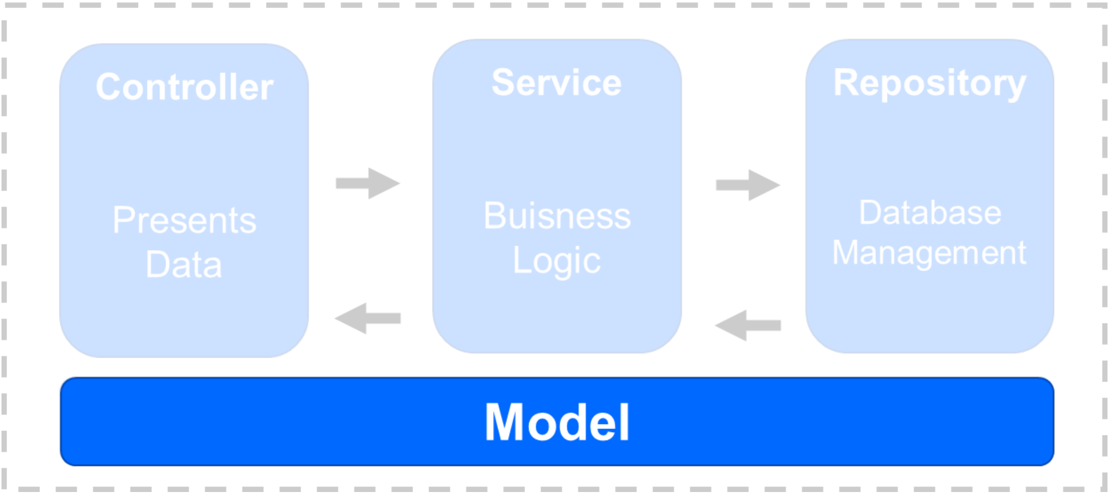
- **Accesses and modify the database.**
- Easy starting point: **Interface that inherits JpaRepository**
  - Standard CRUD operations are already implemented by Spring Data



Model Class / Entitiy

Mapping Java Objects to Data tables

- **A Java-Class that is mapped to a table in a database**
- One model class for each (non-intermediate) table
- Class variables are columns of tables



Validation

- Conditions are defined on **Entity,** Validation in **Controller**
- Automatically validate User input using *Hibernate validator.* Dependency: implementation 'org.springframework.boot:spring-boot-starter-validation’
- Help: https://www.baeldung.com/spring-boot-bean-validation

```
@PostMapping("/car")
ResponseEntity<String>
addCar(@Valid @RequestBody
Car car) {
```

```
public class Car {

@NotNull (message =
„manufacturer is mandatory“)
private String manufacturer;

@NotNull
@Size(min = 2, max = 14)
private String licensePlate;

@Min(2)
private int seatCount; // ...
}
```

Documentaion

Swagger

- Swager can **automatically generate HTML Documentation** for any Java project and continuously update it.
- Setup:
  - implementation 'org.springdoc:springdoc-openapi-ui:1.6.6’
- Attention: Swagger tries to serve Documentation on the base URL (may conflict with Endpoints)
- Help: https://www.baeldung.com/spring-rest-openapi-documentation