Microservices with Spring Boot & Spring Cloud

August 22 to 26, 2022

We have a ‘packed’ Agenda

- Spring Boot, Spring MVC & Spring Data
- Building a simple monolith
- Introducing REST APIs
- Break the monolith
- Communication between services using OpenFeign
- Introducing Microservices & Spring Cloud
- Exploring Spring Cloud - Cloud Capabilities
  - Configurations - Spring Cloud Config Server
  - Resilience - Resilience4j
  - Discovery - Eureka Service Discovery
  - Gateway - Spring Cloud API Gateway
  - Tracing - Spring Cloud Zipkin & Sleuth

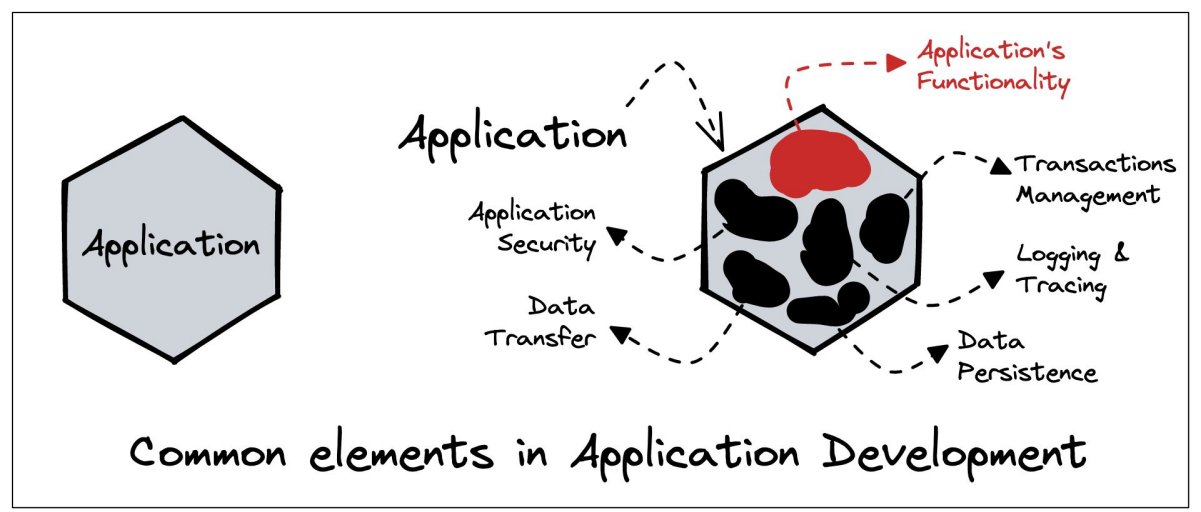
Subbu M

- 12 years and counting
  - Enterprise product design, architecture, development and deployment patterns and technologies
  - Cloud-native and cloud migration product design
  - Kubernetes
-

Spring Boot, Spring MVC & Spring Data

Application Development

- Every application is different, but every application is also same



What is Spring?

- an application framework that provides a foundational structure for developing an application
- makes application development effortless
- an ecosystem of application frameworks
  - Data, messaging, security, APIs, communication, testing, cloud, etc.
  - https://spring.io/projects
- The ecosystem revolves around Spring Core - that provides the Spring Application Context - the magical stage where the application play unwinds!

Inversion of Control (IoC)

- So, what is IoC?
- Usually the application controls the execution of the code through dependencies (instances, method calls)
- IoC means the framework control the application and its dependencies (dependency injection)

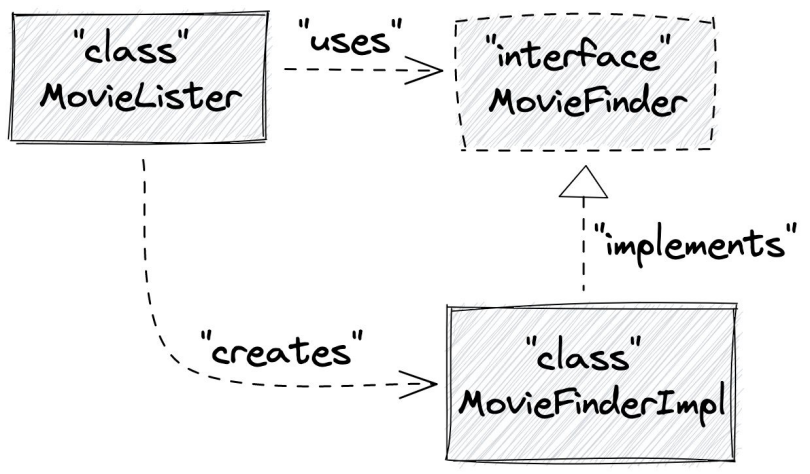
Let us explore the true meaning of IoC, as many find it confusing

- What control are we inverting?

What control are we inverting?

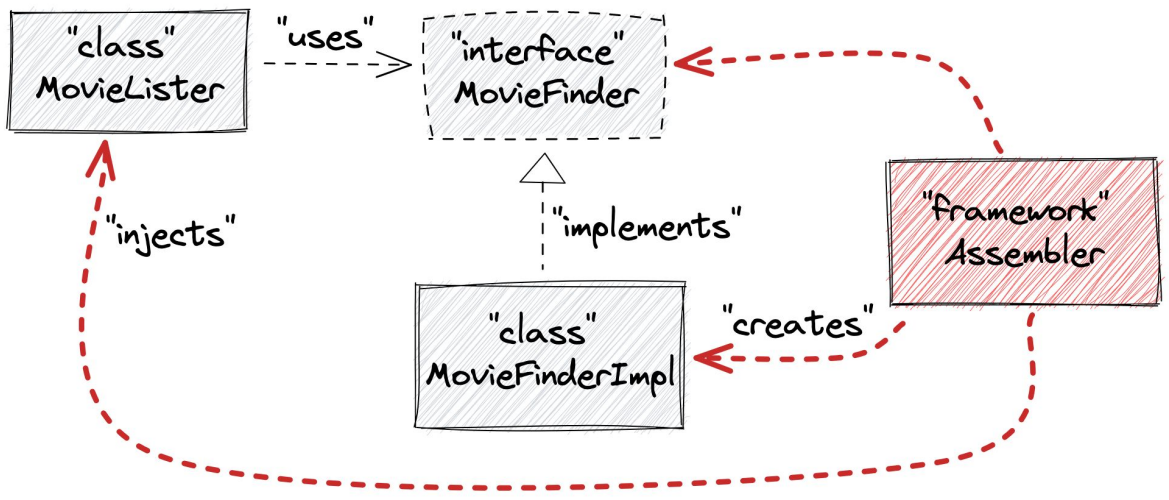
- Martin Fowler's article: https://martinfowler.com/articles/injection.html

- The MovieLister class is dependent on both the MovieFinder interface and upon the implementation.
- Desirable:
  - The implementation is not linked at compile time, but plugged in at some later time



The basic idea of Dependency Injection

- a separate object, an assembler, populates a field in the lister class with an appropriate implementation for the finder interface, resulting in a dependency diagram



Spring Application Context

- Spring Application Context
  - The dependency injection assembler
  - a memory location in which we add the object instances that we want Spring to manage
  - almost everything in a Spring application happens through the context

Lab

- Placing an object in the Spring Application Context
- Placing multiple objects of the same type in the context

Lab

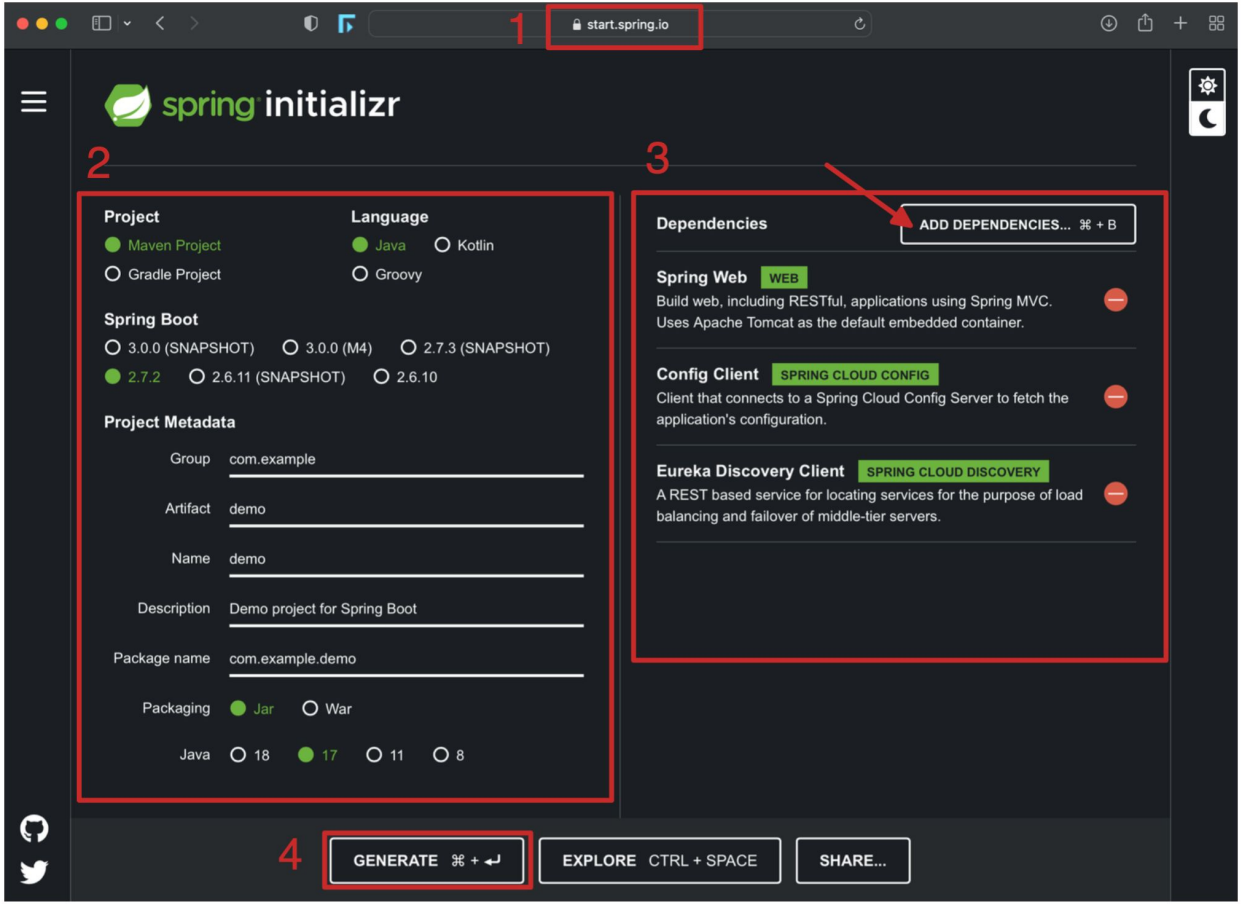
- Wiring Beans - @Component
- Wiring between two @Components
- Wiring for abstractions
- @Service
- @Repository
- Bean Scopes - Singleton & Prototype

Spring Boot Introduction

- a tool for implementing modern Spring applications
- Why Spring Boot?
  - In your microservices journey, you will be creating & update microservices applications more often
  - Configurations are a huge part of developing modern applications like microservices
  - Spring Boot lets you concentrate on developing the business functionalities, by helping eliminate the code related to dealing with configurations

Spring Boot - notable features

- Project Creation:
  - simplified and faster project creation
- Dependency starters:
  - dependency starters group dependencies for specific areas of application development, like persistence or messaging or stream processing
- Auto-configuration:
  - based on the dependencies added to the project, Spring Boot defines default configurations.



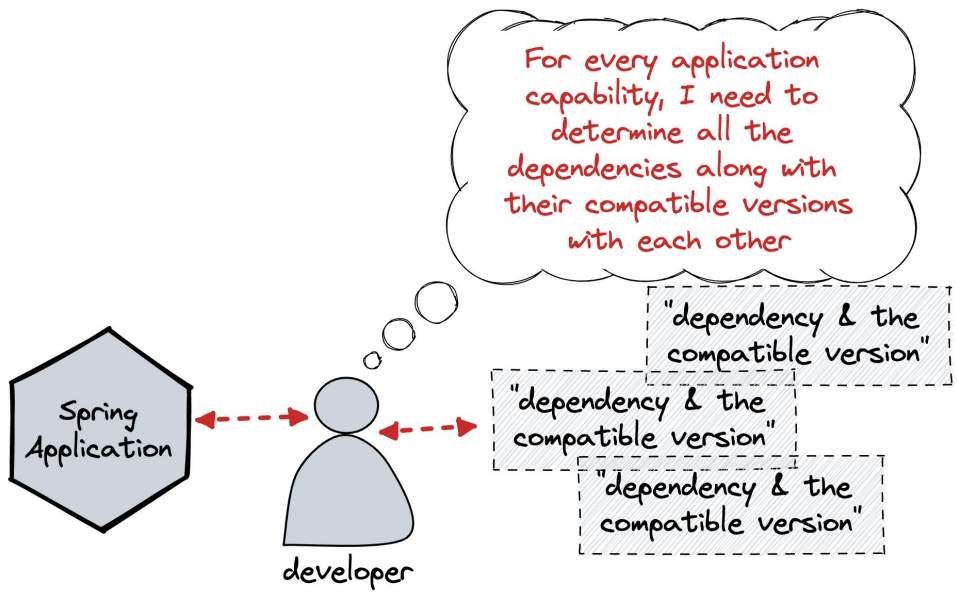
Spring Boot - How To (A 5-step process)

Lab

- Creating the first simple Spring Boot application
  - How it looks!
  - 5 things to note!

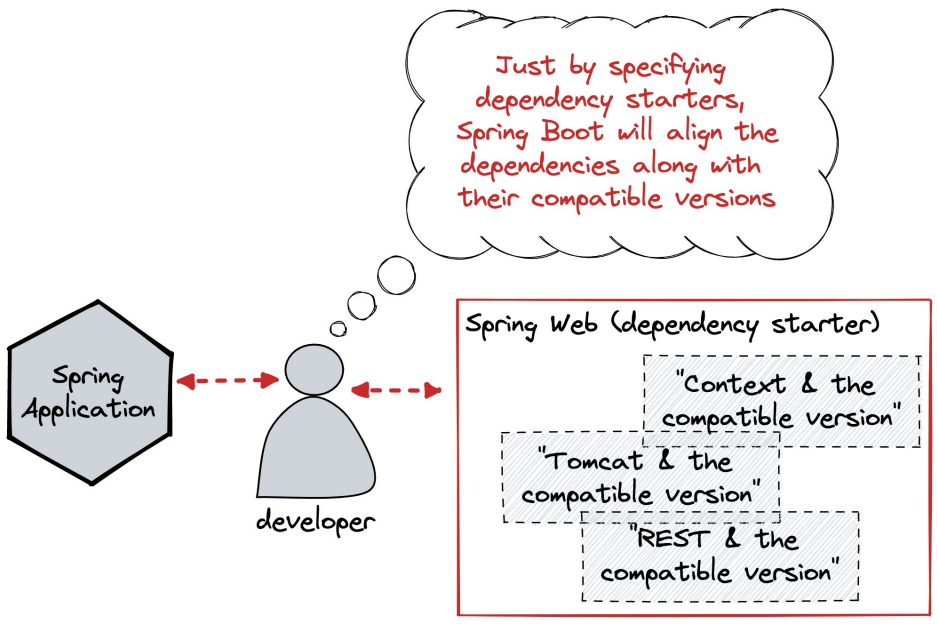
Dependency Starters

- Without Dependency Starters - the normal way of application building



Dependency Starters

- A dependency starter is a group of compatible dependencies that we add to the application for a specific purpose or specific areas of application development



Convention over configuration

Spring Boot chose to run the application in an embedded tomcat on port "8080"

```
Tomcat initialized with port(s): 8080 (http)
Starting service [Tomcat]
Starting Servlet engine: [Apache Tomcat/9.0.65]
Initializing Spring embedded WebApplicationContext
Root WebApplicationContext: initialization completed in 1229 ms
Tomcat started on port(s): 8080 (http) with context path ''
Started FirstSimpleBootApplication in 2.614 seconds (JVM running for 3.106)
```

server.port=8181

```
Tomcat initialized with port(s): 8181 (http)
Starting service [Tomcat]
Starting Servlet engine: [Apache Tomcat/9.0.65]
Initializing Spring embedded WebApplicationContext
Root WebApplicationContext: initialization completed in 1044 ms
Tomcat started on port(s): 8181 (http) with context path ''
Started FirstSimpleBootApplication in 2.007 seconds (JVM running for 2.448)
```

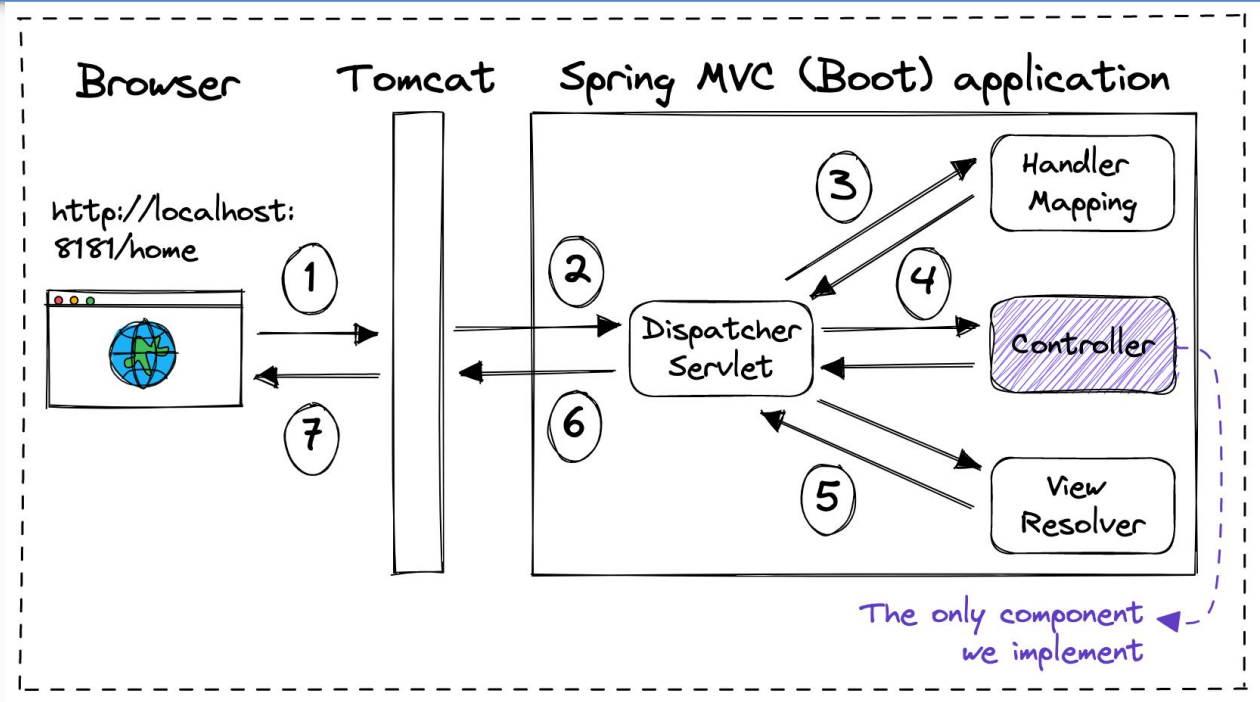
Spring Boot & Spring MVC

- The “Spring Web” dependency starter brings in all the components to build a Spring web application
- Model View Controller - A proven pattern for web applications
  - Model - conduit between controllers and data store
  - Controller - logic for handling input or requests to specific pages
  - View - handles data passed via the controller, then presents the data onscreen
- With Spring Boot, all happens in 2 steps: as simple as that

Lab

- Creating a static web application

Web Application - How things work



Spring Boot - Dynamic Web Application

- Dynamic web apps need dynamic web pages
- There are many proven ways in Java to build dynamic web pages
- Template engines are less intrusive, and you will be writing HTML + something to that add dynamism to the web page
- Thymeleaf is the preferred template engine by Spring Boot - Spring MVC

Lab

- Creating a dynamic web application

Spring Boot - Sending Data through request

- Use the following ways to send data through the HTTP request
  - request parameter
  - path variable
  - request header
  - request body
- We will try the first two options:
  - request parameter
  - path variable

Lab

- Getting the colour as request parameter
- Getting the gift as path variable

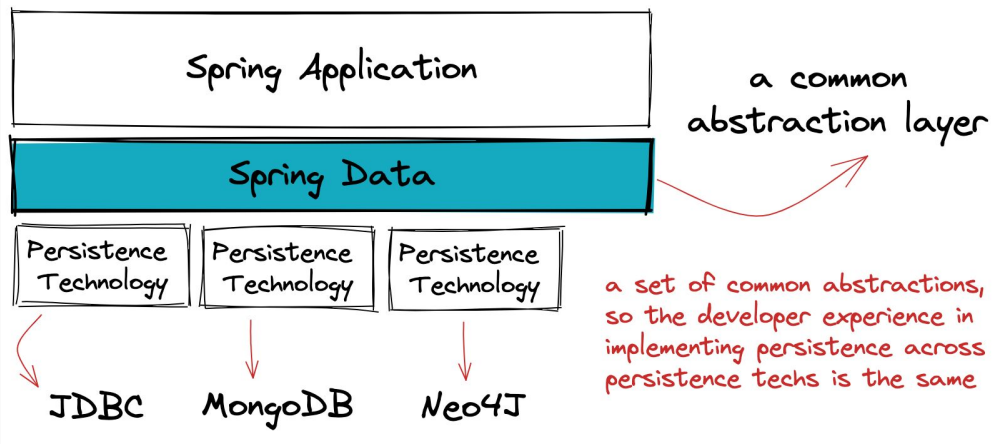
Building a monolith

Lab

- A simple library - monolith application with in-memory storage

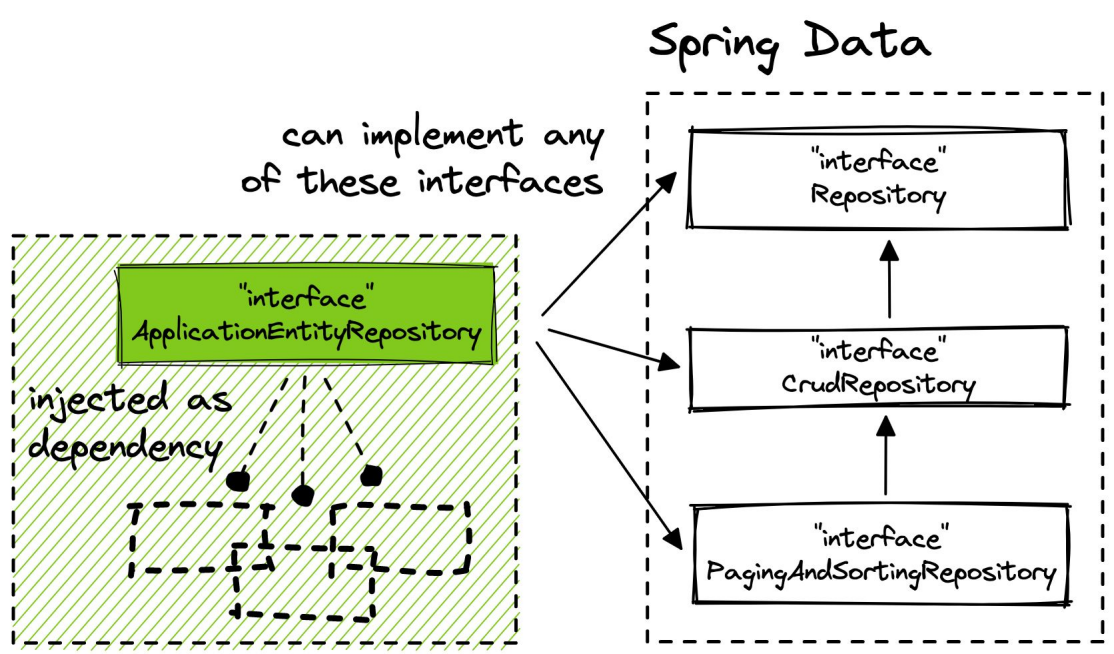
Spring Data - a run through

- provides common set of abstractions for various persistence technologies, so the approach to implement persistence across persistence technologies is similar



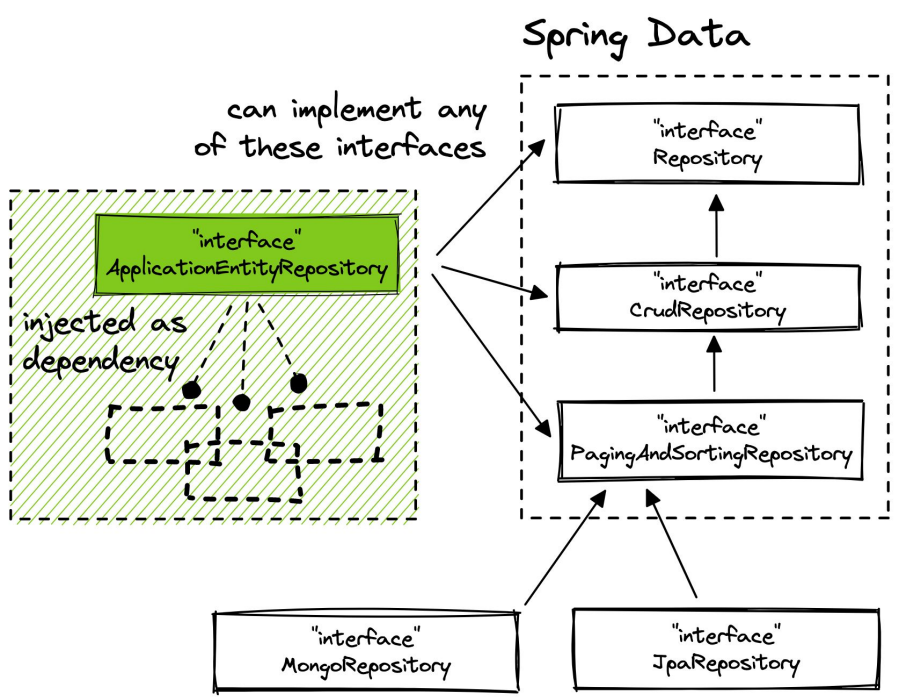
Spring Data - how does it work?

- one dependency starter for each persistence technology
- few common abstractions that can be extended for persisting application data



Spring Data - new age persistence?

- Apart from the common capabilities, Spring Data modules provide specific contracts for specific technology capabilities
  - https://spring.io/projects/spring-data



Spring Data - how to!

- add the dependency to represent the underlying persistence technology
  - For JDBC, add Spring Data JDBC
  - For JPA, add Spring Data JPA
  - For MongoDB, add Spring Data MongoDB
- extend the interface to represent the capabilities needed
  - CrudRepository
  - PagingAndSortingRepository
  - JpaRepository
  - MongoRepository

Lab

- A simple library - monolith application, persisting in H2

REST APIs &
Building REST services

REST using Spring Boot

```java
@Controller
public class LibraryAPIController {
    @RequestMapping(path = "/library",
            method = RequestMethod.GET)
    @ResponseBody
    public String sayHello() {
        return "Welcome to our Library";
    }
}
```

```java
@RestController
public class LibraryAPIController {
    @GetMapping("/library")
    public String sayHello() {
        return "Welcome to our Library";
    }
}
```

```java
@Controller
public class LibraryAPIController {
    @GetMapping("/library")
    @ResponseBody
    public String sayHello() {
        return "Welcome to our Library";
    }
}
```

A bit more REST using Spring Boot

```
@RestController
public class LibraryAPIController {
    @GetMapping("/library")
    public Book sayHello() {
        return new Book("Wings of Fire",
                "APJ Abdul Kalam", 400,
                LocalDate.now());
    }
}
```

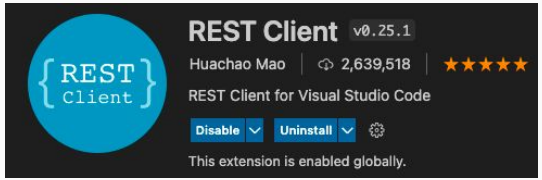
- We can return objects
- Spring Boot defaults the object representation to JSON (uses Jackson)

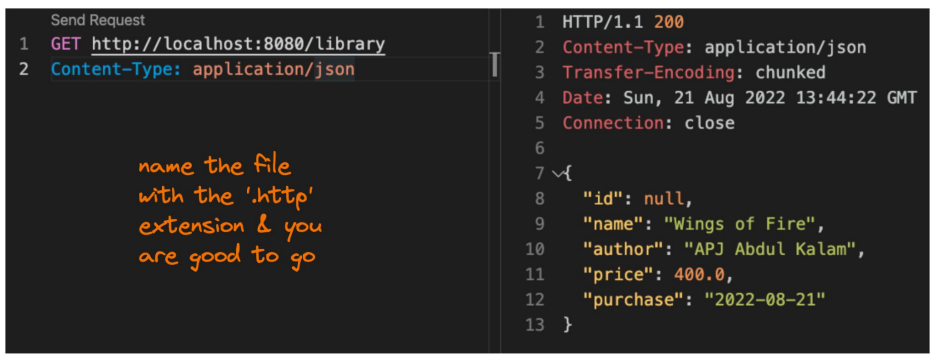
As you can see, we need a better tool to REST!

Better tool to REST!

- There are many tools (open-source) available to do REST better
  - PostMan is the most famous
- In fact, we can write our own tool to play REST

We will use Visual Source Code with REST Client plugin





A bit more REST using Spring Boot

```
@RestController
public class LibraryAPIController {
    @GetMapping("/library")
    public Book sayHello() {
        return new Book("Wings of Fire",
                "APJ Abdul Kalam", 400,
                LocalDate.now());
    }
}
```

- We can return objects
- Spring Boot defaults the object representation to JSON (uses Jackson)

As you can see, we need a better tool to REST!

Breaking the monolith

Communication between services using OpenFeign

Introducing Microservices & Spring Cloud

Configurations - Spring Cloud Config Server

Resilience - Resilience4j

Discovery - Eureka
Service Discovery

Gateway - Spring Cloud API Gateway

Tracing - Spring Cloud Zipkin & Sleuth

Materials

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5

Homework

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua

1. Incididunt ut labore et dolore
2. Consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua