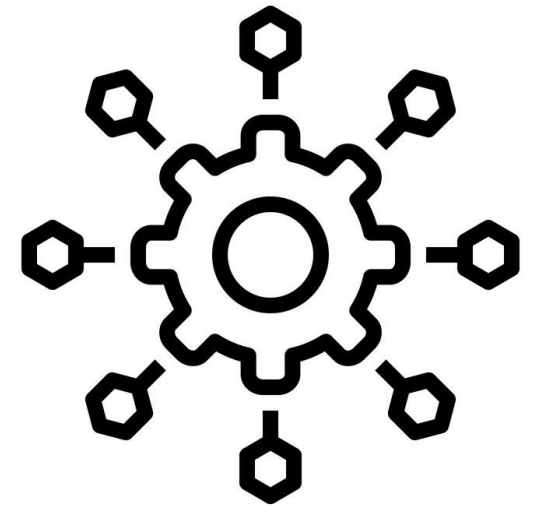




SBM-1 Spring Boot
Microservices



Inversion of control (IoC)

- With Inversion of Control we want to decouple some dependencies
- Goal is to improve simplicity
- Goal is to improve testability

Dependency injection (DI)

- “Dependency Injection (DI) is a form of inversion of control, where implementation are passed into an object by an IoC container”
- Different types of Injection types:
 - Field-based Injection
 - Method-based Injection
 - Constructor-based Injection

Dependency injection - **field**

```
public class InjectionClass {  
    private Object field;  
}
```

Dependency injection – method/setter

```
public class InjectionClass {  
    private Object field;  
  
    void setField (Object field) {  
        this.field = field;  
    }  
}
```

Dependency injection - constructor

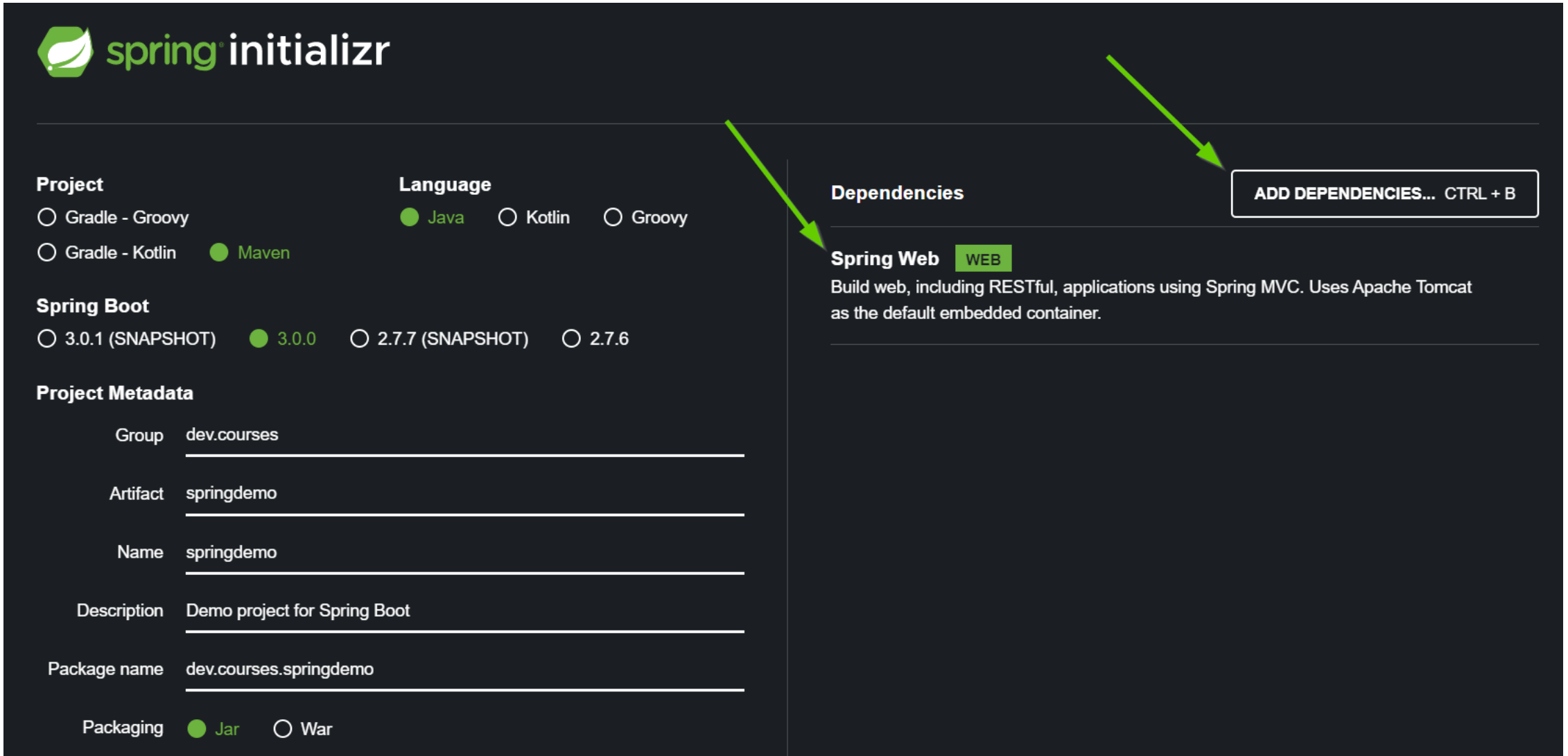
```
public class InjectionClass {  
    private final Object field;  
  
    public InjectionClass(Object field) {  
        this.field = field;  
    }  
}
```

Springboot

Generating a new project



Spring initializr - <https://start.spring.io/>



The image shows the Spring Initializr web form with several green arrows pointing to specific elements: one to the 'Project' section, one to the 'Language' section, one to the 'Spring Web' dependency, and one to the 'ADD DEPENDENCIES...' button.

spring initializr

Project

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ Maven

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 3.0.1 (SNAPSHOT) ☒ 3.0.0 ☐ 2.7.7 (SNAPSHOT) ☐ 2.7.6

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Dependencies

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

ADD DEPENDENCIES... CTRL + B

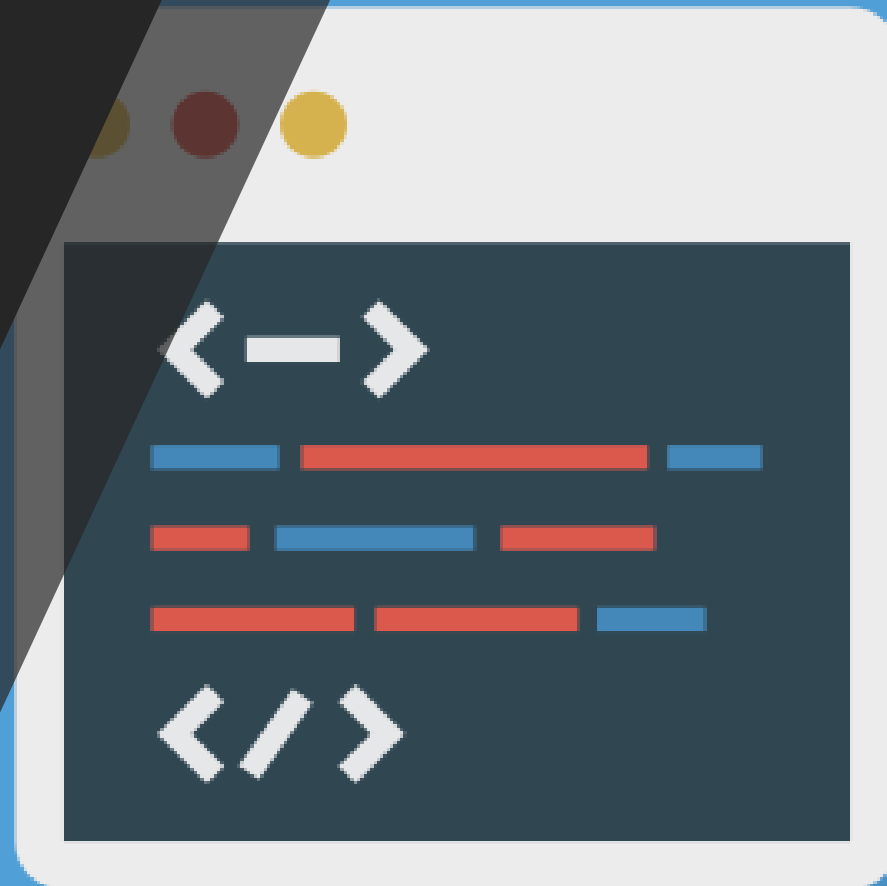
Github repo

- <https://github.com/spring-guides/gs-spring-boot>



Assignment

Dependency injection



Dependency injection - field/annotation

```
@Component
public class MyClass {

    private final Dependency1 dependency1;
    private final Dependency2 dependency2;

    public MyClass(Dependency1 dependency1, Dependency2 dependency2) {
        this.dependency1 = dependency1;
        this.dependency2 = dependency2;
    }
}
```



```
@Component
public class MyClass {

    @Autowired
    private Dependency1 dependency1;
    @Autowired
    private Dependency2 dependency2;

}
```

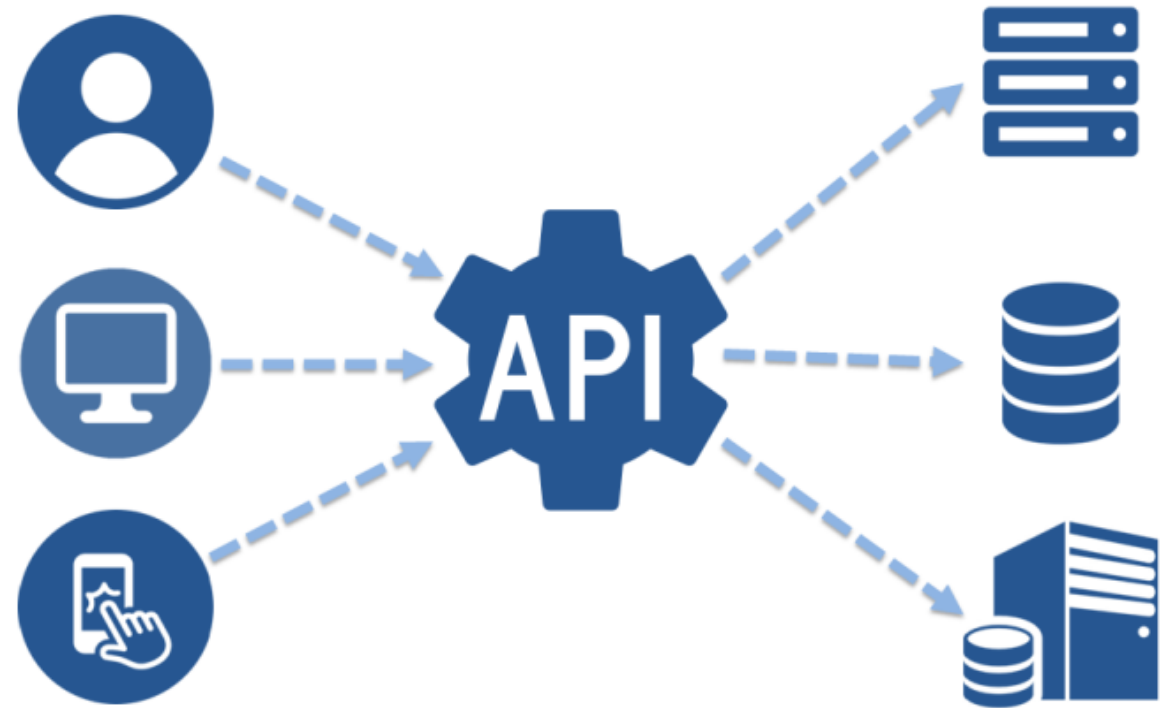
Why Should I Use Constructor Injection?

- **All required dependencies are available at initialization time:** The IoC container makes sure that all the arguments provided in the constructor are available before passing them into the constructor. This helps in preventing the infamous **NullPointerException**.
- **Identifying code smells:** Constructor injection helps us to identify if our bean is dependent on too many other objects. If our constructor has a large number of arguments this may be a sign that our class has too many responsibilities. We may want to think about refactoring our code to better address proper separation of concerns.

Why Should I Use Constructor Injection?

- **Preventing Errors in Tests:** constructor injection simplifies writing unit tests. The constructor forces us to provide valid objects for all dependencies. Constructor injection ensures that our test cases are executed only when all the dependencies are available. It's not possible to have half created objects in unit tests

REST API



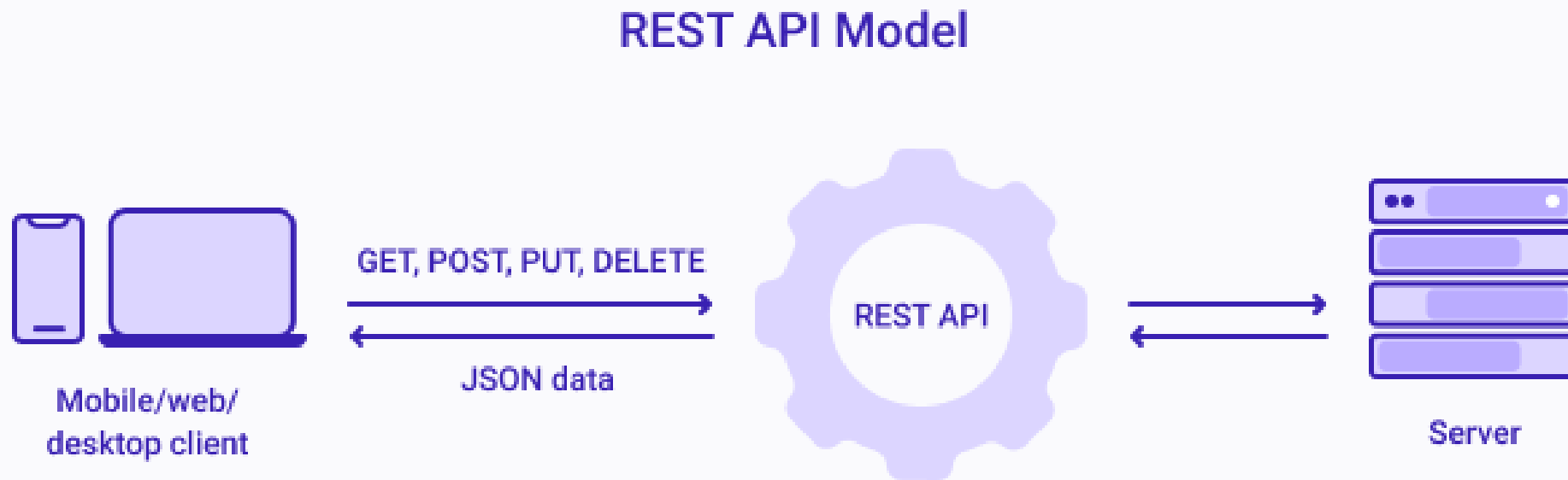
What is an API?

API stands for **application programming interface**, which is a set of definitions and protocols for building and integrating application software

What is a REST API? (Representational State transfer)

- A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services
- **RESTful**: it has to conform to these criteria:
 - A **client-server architecture** made up of clients, servers, and resources, with requests managed through **HTTP**
 - **Stateless client-server communication**, meaning no client information is stored between get requests and each request is separate and unconnected
 - Cacheable data that streamlines client-server interactions
 - A uniform interface between components

REST API



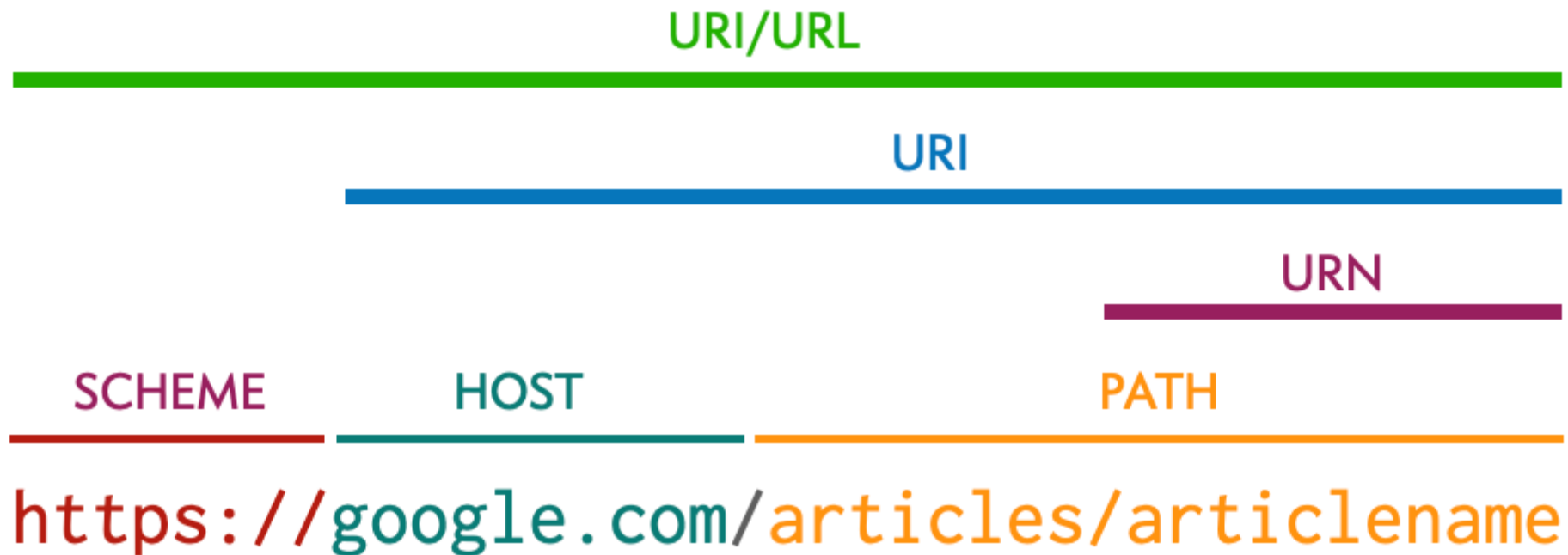
HTTP verbs

HTTP Verb	CRUD
POST	Create
GET	Read
PUT	Update/Replace
PATCH	Update/Modify
DELETE	Delete

JSON

```
{
  "firstName": "BCD",
  "lastName": "ABC",
  "gender": "Female",
  "age": 24,
  "address": {
    "streetAddress": "345",
    "city": "LA",
    "state": "CA",
    "postalCode": "394221"
  },
  "phoneNumbers": [
    { "type": "home", "number": "7383627627" }
  ]
}
```

HTTP request



REST endpoint - CRUD examples

- REST endpoint **HTTP verb + server host URL + entity**
- Get all users **GET http://my-server/users**
- Get user by ID = 15 **GET http://my-server/users/15**
- Get user by name = John **GET http://my-server/users/john**
- Create a user **POST http://my-server/users**
 - Body: {name="John", age=25}
- Update user by ID = 12 **PUT http://my-server/users/12**
 - Body: {name="Leo", age=32}
- Delete user by ID = 2 **DELETE http://my-server/users/2**

REST endpoint - CRUD examples

- Get address of user ID = 3
 - GET /users/3/addresses
- Get book ID = 5 from the library name = Amsterdam
 - GET /libraries/Amsterdam/books/5
- **Use plurals for entities**
 - GET /user/12 -> GET /**users**/12
- **Don't use verbs in URLs.** Use HTTP verbs instead
 - POST /users/createNewUser -> POST /users
 - GET /articles/{article id}/generateBanner -> GET /articles/{article id}/banners




REST endpoint – parameters and body

- **Request parameters:** /users?field1=value1&field2=value2
 - GET /users?name=John&age=15
 - GET /users?pageSize=20&pageNumber=3
- **Path variables:** /users/{user id}
 - GET /users/15
 - GET /users/10/addresses/5
- **Body:**
 - JSON data mainly for POST and PUT
 - GET can't have a body

REST endpoints - exercises

- “Give me users with the name Paul”
 - Solution: `GET /users?name=Paul`
 - “Add a book in the library with title = Memories and pages = 123”
 - `POST /libraries/books`
 - Body = `{"title": "Memories", "pages": 123}`
 - “Make a payment with amount = 1000, currency = USD”
- > Fix this junior endpoint: `GET /payment/1000/USD`
- `POST /payments`
 - Body = `{"amount": "1000", "currency": "USD"}`

HTTP response codes - ranges

	HTTP code range	Type
	100 - 199	Informational response
	200 - 299	Successful responses
	300 - 399	Redirection messages
	400 - 499	Client error responses
	500 - 599	Server error responses

HTTP codes - success

HTTP code	Description
200	OK
201	Created
202	Accepted
204	No Content

HTTP response codes - client errors

HTTP code	Description
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found

HTTP response codes - server errors

HTTP code	Description
500	Internal Server Error
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout

Error handling - risks

- Hide information (security)
- Balance between inform the client and protecting data structures
- Agreements in a company about them and how to react on them

Postman

Postman is an API platform for developers to design, build, test and iterate their APIs



POSTMAN

Free public APIs without authentication

- <https://mixedanalytics.com/blog/list-actually-free-open-no-auth-needed-apis/>
- <https://apipheny.io/free-api/>

Install Postman

Call a public API

Assignment



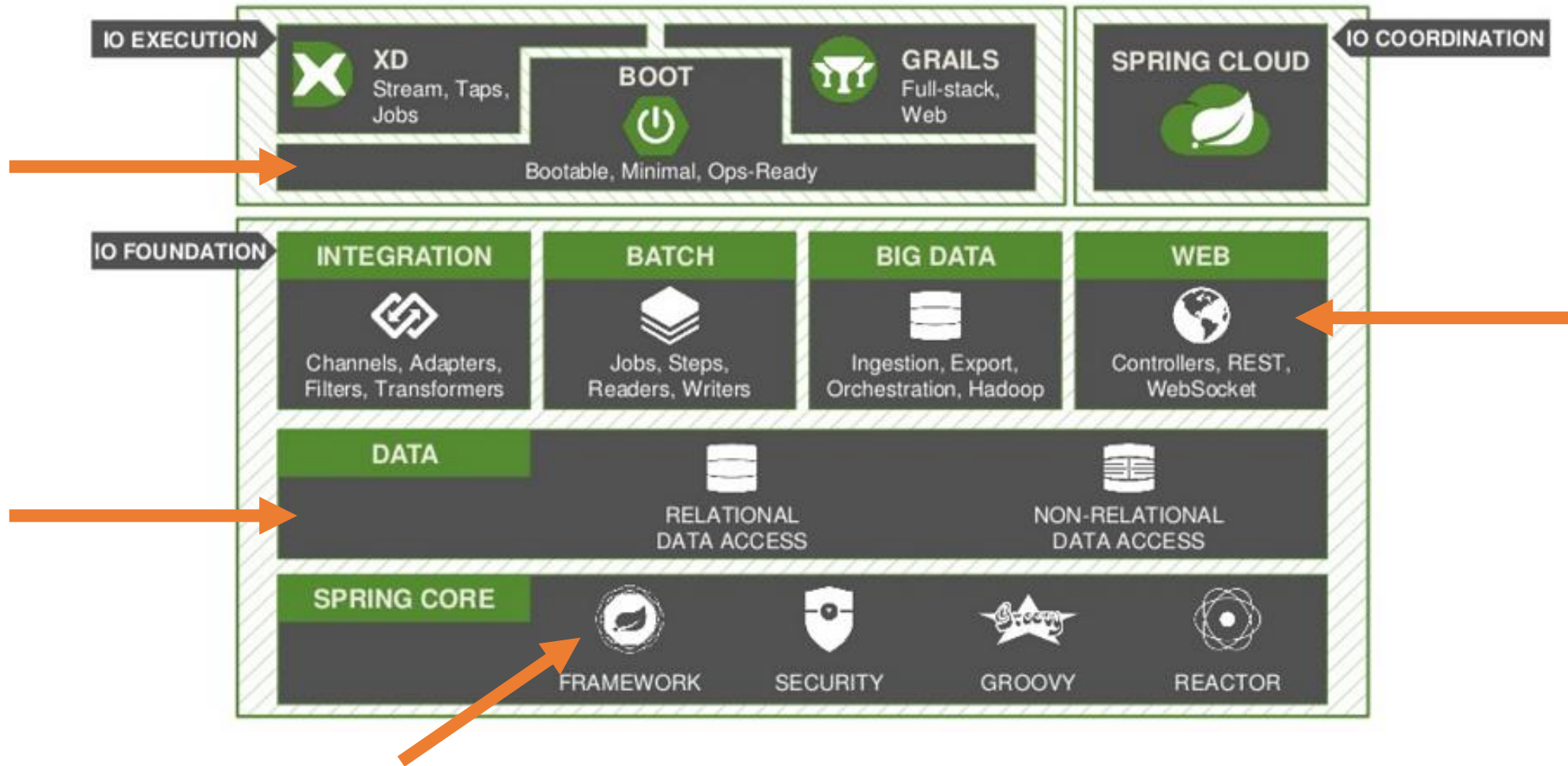
Spring



How can Spring help us?

- Start up the application
- Initialize all the beans
- Run the automated tests

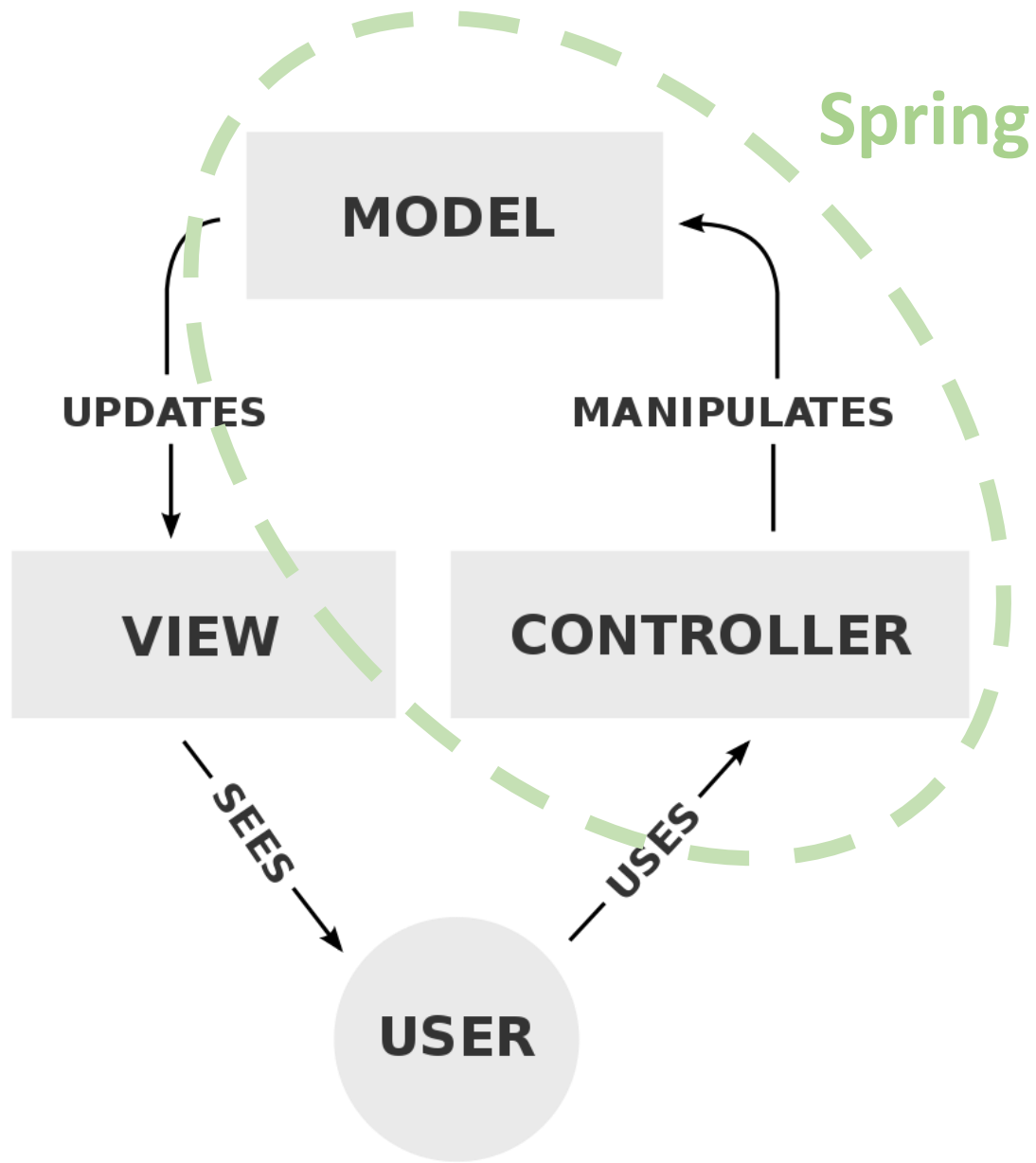
Spring ecosystem - what will we use today



Assignment

Make our first Controller - Hello World





Model–View–Controller (MVC)

MVC is a software architectural pattern commonly used for developing user interfaces that divide the related program logic into three interconnected elements