


# Spring Boot Notes

<input checked="" type="checkbox"/> Favorite	<input type="checkbox"/>
 Tag	<u>Spring Boot</u>

## ▼ Spring vs Spring Boot

- Spring is a powerful, comprehensive framework that gives you a lot of flexibility, but can be a bit overwhelming to set up
- Spring Boot is a more opinionated, streamlined version of Spring that comes with a lot of built-in features to help you get started quickly and easily..

## ▼ Spring's Inversion of Control Container

Imagine you're hosting a big party. Usually, you'd have to organize everything yourself: decide who to invite, where to set up, what food to serve, and so on. But what if there was a magical party planner who could handle all of that for you? That's kind of what Spring's Inversion of Control (IoC) Container does for software.

In software development, there are often lots of different pieces that need to work together to make a program run smoothly. Instead of you having to manage and connect all those pieces manually, Spring's IoC Container does it for you.

Here's how it works:

1. **You define what needs to happen:** Just like telling the party planner your preferences for the party, you tell Spring what components your program needs and how they should work together. This could include things like what objects you need and how they should interact.
2. **Spring takes care of the details:** Once you've told Spring what you need, it handles all the behind-the-scenes stuff. It creates the objects you need, connects them together as you specified, and manages their lifecycle.

3. **You focus on your code:** With Spring's IoC Container handling the setup and management of components, you can focus more on writing the actual code for your program. You don't have to worry about the nitty-gritty details of how everything gets wired together.

So, in simple terms, Spring's IoC Container is like having a magical party planner for your software development. It takes care of all the setup and management tasks so you can focus on writing the code that makes your program awesome.

#### ▼ Spring Initializr

When starting a new Spring Boot application, Spring Initializr is the recommended first step. You can think of Spring Initializr like a shopping cart for all of the dependencies your application might need. It will quickly and easily generate a complete, ready-to-run Spring Boot application.

#### ▼ LAB: Spring Initializr

##### ▼ What is gradle

Gradle is an open-source build automation tool used primarily for Java projects, although it can also be used for other languages such as Kotlin, Groovy, and Scala. It automates the process of building, testing, and deploying software projects.

##### ▼ What is the gradle wrapper

- The wrapper is a script that allows you to run Gradle tasks without having to manually install Gradle on your system.
- When you run a Gradle project using the Gradle Wrapper, it automatically downloads and configures the correct version of Gradle specified by the project. This ensures that everyone working on the project uses the same version of Gradle, helping to maintain consistency and avoid compatibility issues.

#### ▼ API Contracts & JSON

##### ▼ Considerations for API development

- How should API consumers interact with the API?

- What data do consumers need to send in various scenarios?
- What data should the API return to consumers, and when?
- What does the API communicate when it's used incorrectly (or something goes wrong)?

#### ▼ API Contracts

Contracts → Methodologies for capturing agreed upon API behavior in documentation and code

- Consumer Driven Contracts
  - Consumers defining expectations for how an API should behave, typically through automated tests, which the provider must fulfill to ensure compatibility.
- Provider Driven Contracts
  - Defined by the API provider, specifying the expected behavior and constraints, ensuring consistency and reliability across consumer implementations.
- The provider and consumers do not have to share the same programming language, only the same API contracts.
- API contracts can be as simple as shared documentation to sophisticated contract management and validation frameworks.

#### ▼ Example

For the Family Cash Card domain, let's assume that currently there's one contract between the Cash Card service and all services using it.

##### Request

URI: /cashcards/{id}  
 HTTP Verb: GET  
 Body: None

##### Response:

HTTP Status:  
 200 OK if the user is authorized and the Cash Card

```
401 UNAUTHORIZED if the user is unauthenticated or
404 NOT FOUND if the user is authenticated and autl
Response Body Type: JSON
Example Response Body:
{
  "id": 99,
  "amount": 123.45
}
```

#### ▼ Importance of API Contracts

- API contracts are important because they communicate the behavior of a REST API.
- They provide specific details about the data being serialized (or deserialized) for each command and parameter being exchanged.

#### ▼ Testing First

##### ▼ Notes

##### ▼ Process of serialization

Imagine you have a really cool toy spaceship made of LEGO bricks. You decide to take it apart and put it in a box so you can store it or send it to a friend. This process of taking apart the spaceship and putting it in the box is like serialization. It turns the spaceship into something that can be easily stored or moved around.

Now, when your friend receives the box with all the LEGO pieces, they can take them out and put them back together to rebuild the spaceship. This is like deserialization. It transforms the LEGO pieces from the box back into the original spaceship shape, so your friend can play with it again.

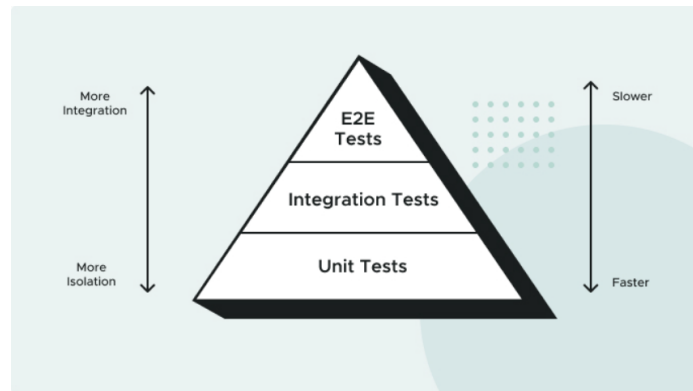
In the same way, serialization turns complex data from your computer into a format that can be easily saved or sent. And deserialization does the opposite—it turns that saved or sent data back into its original form, so it can be used in your program just like before. This way, you

can move data between different computers or platforms, just like sending your LEGO spaceship to your friend in another city.

### ▼ TDD

- Test Driven Development
- By asserting expected behavior before implementing the desired functionality, we're designing the system based on what we want it to do, rather than what the system already does.
- The tests guide you to write the minimum code needed to satisfy the implementation.

### ▼ Testing Pyramid



Unit Tests → A small "unit" of the system that's isolated from the rest of the system

Integration Tests → Exercise a subset of the system and may exercise groups of units in one test

End-to-End Tests → Exercises the system using the same interface that a user would, such as a web browser

### ▼ Red, Green, Refactor Loop

A fundamental practice in test-driven development (TDD)

1. Red: Write a failing test for the desired functionality.
2. Green: Implement the simplest thing that can work to make the test pass.

3. Refactor: Look for opportunities to simplify, reduce duplication, or otherwise improve the code without changing any behavior—to refactor.
4. Repeat!

## ▼ Implementing GET

### ▼ Notes

#### ▼ What is REST

Alright, imagine you're at a fast-food restaurant. You walk up to the counter and order a burger. The person at the counter takes your order and gives you a ticket with a number on it. You go sit down and wait for your number to be called.

In the world of computers, REST (Representational State Transfer) is kinda like that. It's a way for different computer programs or systems to talk to each other and exchange information, just like you ordered a burger at the counter.

Here's how it works:

1. **Ordering:** In REST, you "order" something by sending a message to a server, like ordering a burger at the counter. This message tells the server what you want to do, like get some data or update something.
2. **Ticket:** When the server gets your message, it does something with it, like getting the data you asked for or updating a record in a database. Then, it sends you back a response, just like when you get a ticket with a number on it at the restaurant.
3. **Waiting:** You wait for your number to be called at the restaurant, right? In REST, you wait for the server to send you back the response with the data or the result of your request.

REST is like a common language that computers use to talk to each other over the internet. It helps different systems understand each

other and work together smoothly, just like ordering a burger at your favorite fast-food joint.

#### ▼ What is REST API

Alright, think of a REST API like a menu at a restaurant. You know how a menu shows you all the different dishes you can order? Well, a REST API is like a menu for a web service or application.

Here's how it works:

1. **Menu Items:** On a menu, you have different items like burgers, salads, and drinks. Similarly, a REST API has different "endpoints" that represent different actions or resources you can interact with. For example, you might have endpoints for getting user information, creating a new post, or updating a product.
2. **Ordering:** When you want to do something, like get information about a user or add a new post, you make a request to the corresponding endpoint, just like ordering a dish from the menu.
3. **Response:** After you place your order, the restaurant prepares your food and brings it to your table. Similarly, when you make a request to a REST API, it processes your request and sends back a response with the information you asked for or confirming that the action was completed.

So, a REST API is basically a way for different software systems to communicate with each other over the internet by sending and receiving requests and responses, just like ordering and receiving food at a restaurant.

#### ▼ REST, CRUD and HTTP

- In a RESTful system, data objects are called Resource Representations.
- State → value
- Resource Representation → object

#### ▼ REST in Spring Boot

- ▼ Spring Annotations and Component Scan

- One of the main things Spring does is to configure and instantiate objects
  - Objects are spring beans
- Don't have to use the Java new keyword
  - Direct Spring to create Beans

#### ▼ Process

- Spring annotation directs Spring to create an instance of the class during Spring's Component Scan phase.
- Component Scan Phase happens at application startup
- Bean is stored in Spring's IoC container
- Bean can be injected into any code that requests it

#### ▼ Spring Web Controllers

- Requests are handled by controllers

```
@RestController
class CashCardController {
}
```

#### ▼ Repositories & Spring Data

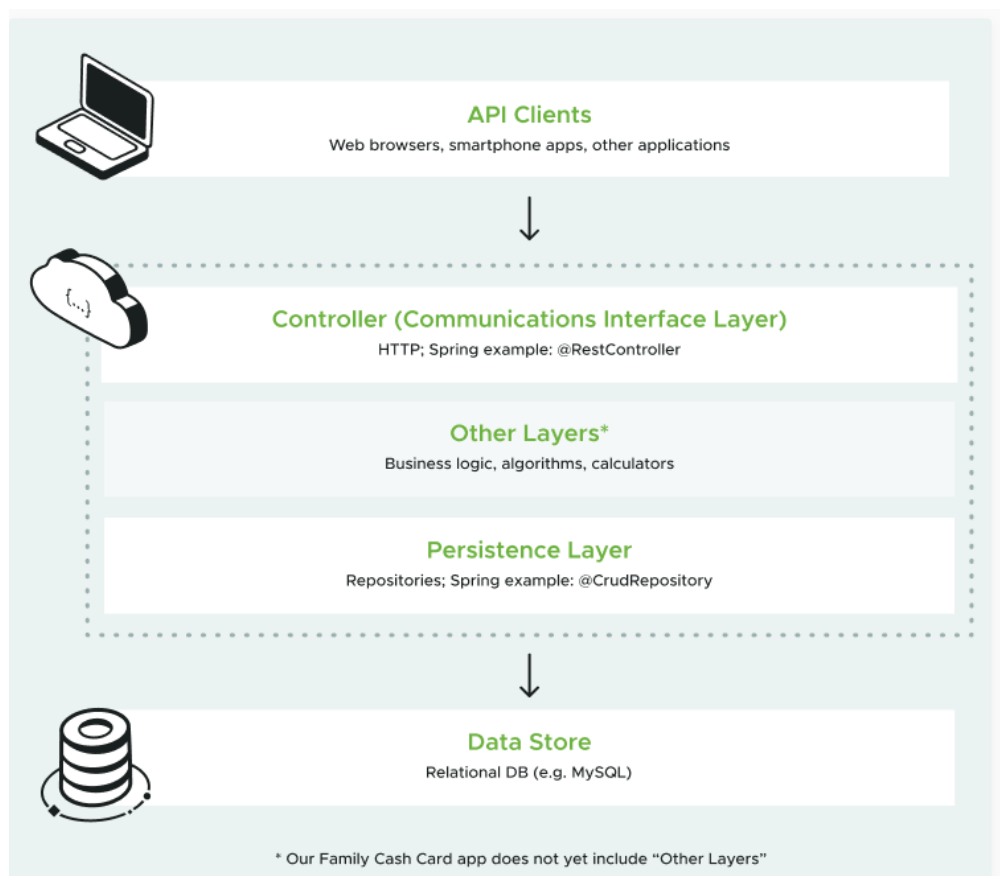
##### ▼ Controller-Repository Architecture

Separation of Concerns principle → Well-designed software should be modular, with each module having distinct and separate concerns from any other module

- In Spring, the Repository pattern is a design pattern used to abstract away the details of data access and manipulation from the rest of the application. It typically involves creating interfaces that define methods for accessing and managing data, without specifying how the data is actually stored or retrieved.

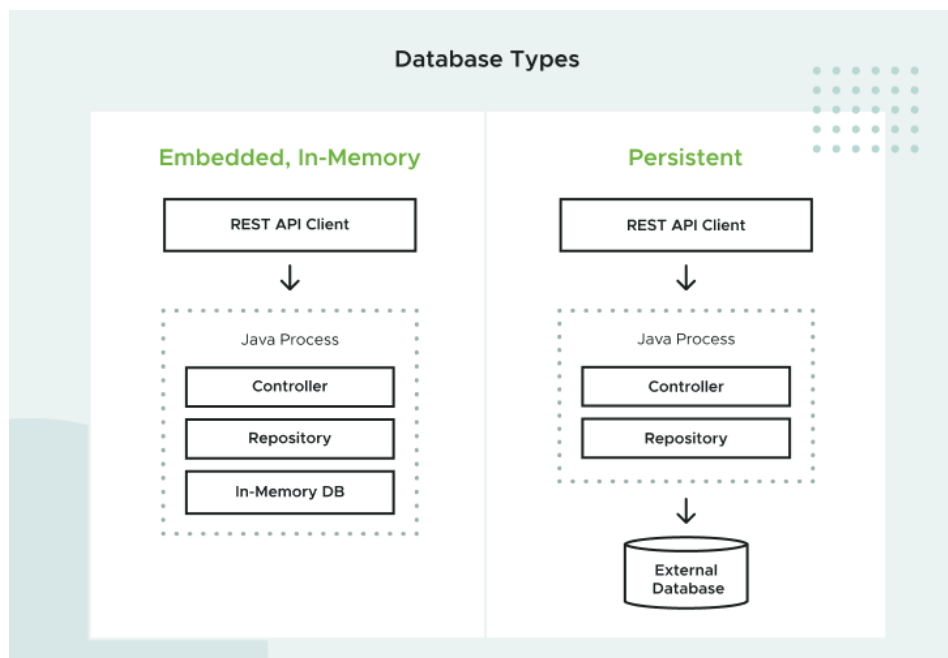


- By using the Repository pattern in Spring, you can achieve a clean separation of concerns between your business logic and data access logic. This makes your code more modular, easier to test, and less tightly coupled to the underlying data storage technology. Additionally, Spring provides built-in support for implementing the Repository pattern through features like Spring Data JPA, which simplifies the creation of repository interfaces and provides automatic implementation based on conventions.
- A common architectural framework that divides these layers, typically by function or value, such as business, data, and presentation layers, is called Layered Architecture.
- The Repository is the interface between the application and the database, and provides a common abstraction for any database, making it easier to switch to a different database when needed.



#### ▼ Choosing a database

- Embedded, in memory
  - Embedded → it's a Java library, so it can be added to the project just like any other dependency
  - In-memory → stores data in memory only, as opposed to persisting data in permanent
  - Largely compatible with production-grade relational database management systems (RDBMS) like MySQL, SQL Server



- Pros
  - Can develop without installing separate RDBMS
  - Ensure database is in the same state on every test run
- Cons
  - You need persistent database for live production application
  - Dev-Prod Parity
    - Application may behave differently when running in-memory database and when running in production.

#### ▼ Auto Configuration

- Need to add 2 dependencies for full database functionality
  - Spring Data dependency
  - Specific data provider, H2

#### ▼ Spring Data's CrudRepository

```
interface CashCardRepository extends CrudRepository<CashCard, Long> {
}
```

```
cashCard = cashCardRepository.findById(99);
```

- The `CrudRepository` interface defines generic methods for CRUD operations such as `save`, `findById`, `findAll`, `delete`, etc. However, it doesn't contain the implementation of these methods. Instead, it declares method signatures that need to be implemented by concrete repository interfaces or classes.
- When you use `cashCardRepository.findById(99)`, Spring Data JPA dynamically generates the implementation for the `findById` method based on the entity type (`CashCard`) and the primary key type (`Long`). Under the hood, it translates this method call into a corresponding SQL query to retrieve the `CashCard` entity with the specified ID from the database.

#### ▼ What is the steel thread approach

Imagine building a really cool, complex structure like a skyscraper, but instead of using giant steel beams, you're using something much smaller, like threads. These threads are like tiny strands of steel, really thin but strong.

In software engineering, the "steel thread approach" is a way of designing and building software by focusing on small, strong, and essential parts first, just like those threads. Instead of trying to tackle everything all at once, you start

by creating the most important and basic parts of the software, making sure they're solid and dependable.

Once you have these essential parts, you gradually build upon them, adding more features and complexity, just like adding more threads to make a stronger structure. This approach helps ensure that even if something goes wrong or changes later on, the core parts of the software remain strong and reliable.

So, in simpler terms, the steel thread approach is about starting small, making sure the basics are really solid, and then gradually building up from there to create something robust and dependable, just like building a skyscraper with tiny, strong threads.

#### ▼ Idempotence and HTTP

*Idempotency* is a property of HTTP methods.

A request method is considered *idempotent* if the intended *effect* on the server of multiple identical requests with that method is the same as the *effect* for a single such request. And it's worthwhile to mention that idempotency is about the *effect* produced on *the state of the resource* on the server and not about the *response status code* received by the client.

To illustrate this, consider the `DELETE` method, which is defined as idempotent. Now consider a client performs a `DELETE` request to delete a resource from the server. The server processes the request, the resource gets deleted and the server returns `204`. Then the client repeats the same `DELETE` request and, as the resource has already been deleted, the server returns `404`.

Despite the different status code received by the client, the effect produced by a single `DELETE` request is the same effect of multiple `DELETE` requests to the same URI