Week- 4

Backend Engineering Launchpad

Pawan Panjwani

Airtribe

Backend Development Rest API and Java Frameworks for Backend Development

Airtribe

Agenda

- What is API and introduction to Rest API
- REST API and HTTP verbs
- Modelling restful endpoints
- Frameworks for java web applications
- Live coding
- Question and answers

Airtribe

“An API is a precise specification of the programming instruction and standards to access a web based software or web tool...

What is an API?

- Acronym for Application program interface
- Acts as a bridge between the programmer and application
- Takes specific requests predefined when created
- Verifies the request and processes the data
- Performs the processing and returns the response to the user

What is REST

- REST stands for representational state transfer
- REST is not a framework but an architectural principle
- REST uses http which is oriented around verbs and resources (GET, POST, PUT, DELETE)

What is REST (continued…)

- Verbs are then applied to resources
- Request consists of headers, body and methods
- Response consists of headers, status code and body
- Data represented via XML/JSON/HTML

Key Principles of REST

- Stateless
- Client Server Architecture
- Uniform Interfaces
- Resource Oriented

API architectural styles

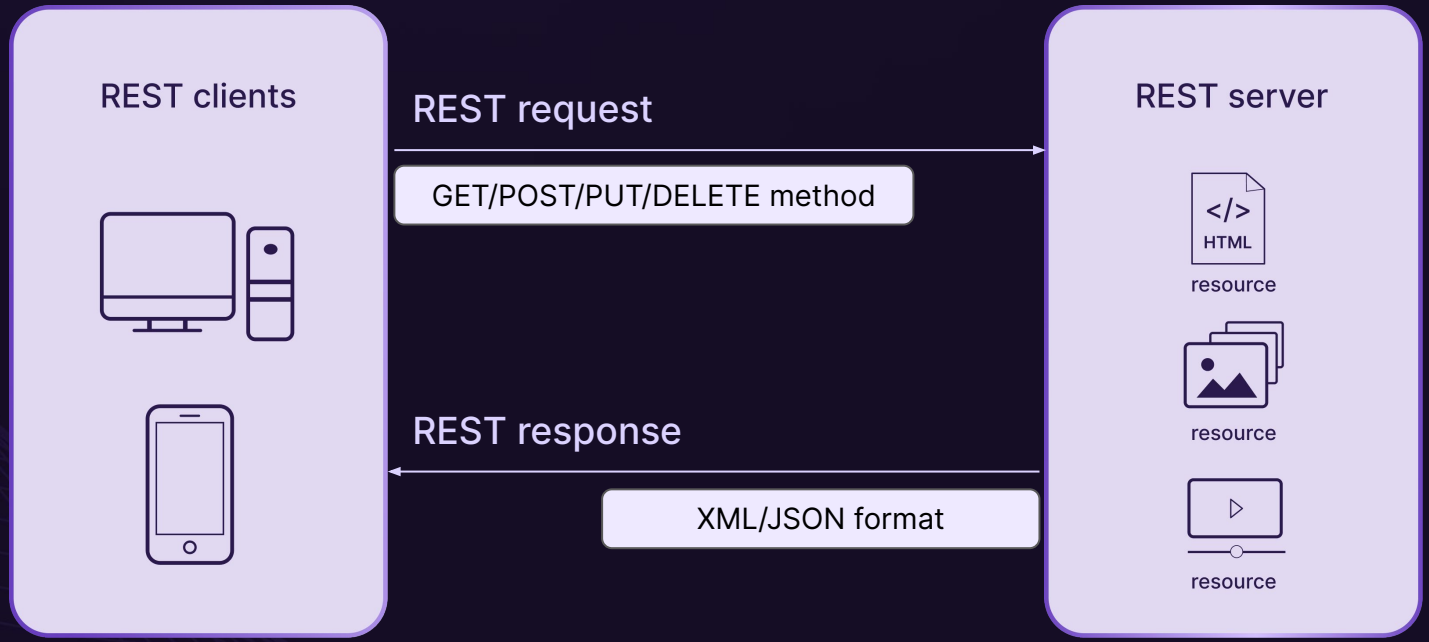
	REST	SOAP	RPC	GraphQL
Organized in terms of	compliance with six architectural constraints	enveloped message structure	local procedure calling	schema & type system
Format	XML, JSON, HTML, plain text,	XML only	JSON, XML, Protobuf, Thrift, FlatBuffers	JSON
Learning curve	Easy	Difficult	Easy	Medium
Community	Large	Small	Large	Growing
Use cases	Public APIs simple resource-driven apps	Payment gateways, identity management CRM solutions financial & telecommunication services, legacy system support	Command and action-oriented APIs; internal high performance communication in massive micro-services systems	Mobile APIs, complex systems, micro-services

Credit: Altexsoft

Airtribe

Rest API architecture

REST API IN ACTION



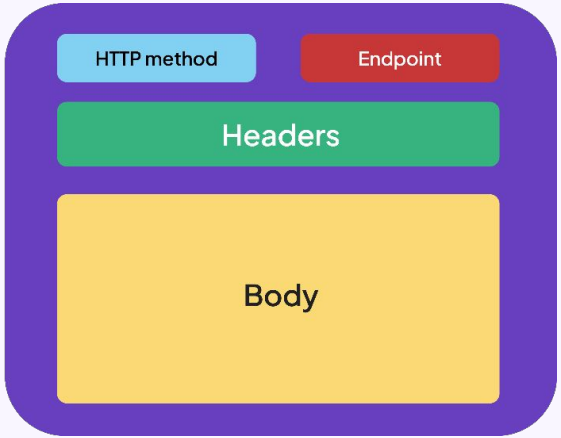
Credit: Altexsoft

Airtribe

REST API request example

```
POST /api/2.2/sites/9a8b7c6d-54f-3a2b-1c0d-98f7a6b5c4d/users HTTP/1.1
HOST: my-server
X-Tableau-Auth: 12ab34cd56ef78ab90cd12ef34ab56cd
Content-Type: application/json

{
"user": {
 "name": "NewUser1",
 "siteRole": "Publisher"
  }
}
```



Credit: Altexsoft

Airtribe

REST API response example

```
HTTP/1.1 200 OK (285ms)
Date: Fri, 21 Apr 2017 10:27:20 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.1e-fips PHP/7.0.16
X-Powered-By: PHP/7.0.16
Content-Length: 109
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json; charset=UTF-8
```

{"status":"success","message":"City List","data":[{"city_name":"Visakhapatnam"},{"city_name":"Vijayawada"}]}

Airtribe

HTTP response example

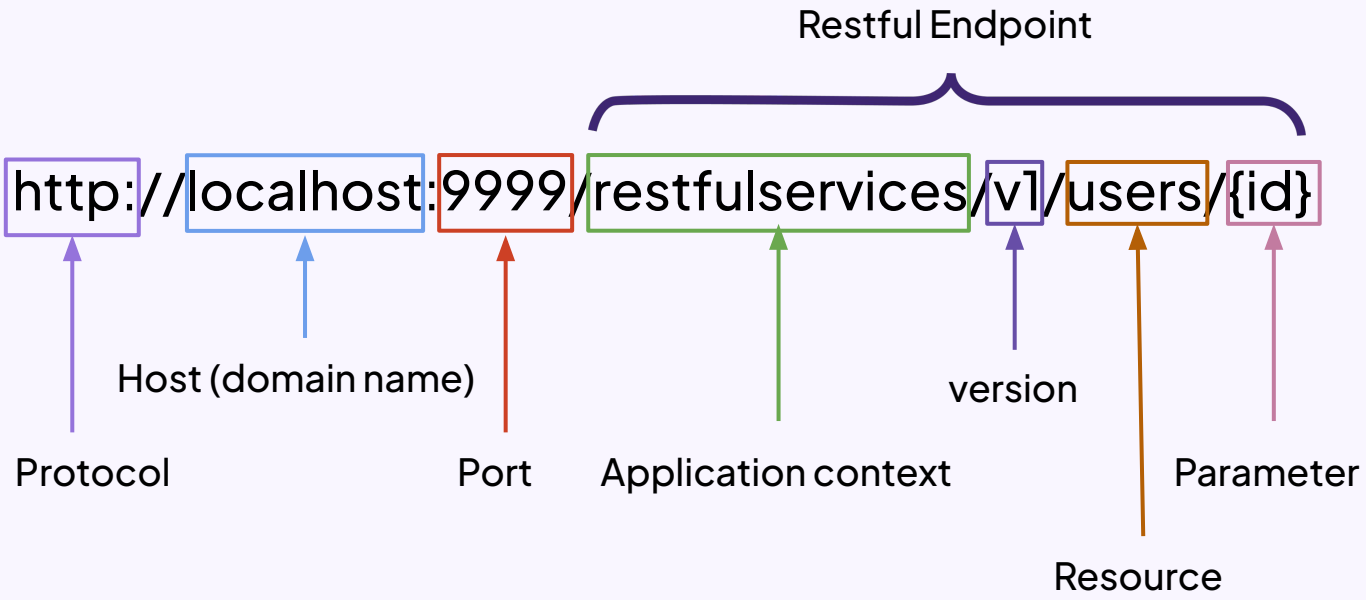
HTTP Status Codes

Level 200 (Success)
OK
Created
203 : Non-Authoritative Information
204 : No Content

Level 400
400: Bad Request
401: Unauthorized
403: Forbidden
404: Not Found
409: Conflict

Level 500
500: Internal Servor Error
503: Service Unavailable
501: Not Implemented
504: Gateway Timeout
599: Network timeout
502: Bad Gateway

Restful endpoint structure



Airtribe

Frameworks for java web applications

Airtribe

What is a framework ?

- **Framework**: A framework in Java is a pre-defined structure or platform that provides a foundation and standard set of functionalities to build applications. It includes libraries, tools, and best practices.

- **Characteristics:**

  - Reusable code
  - Inversion of control
  - Standardization
  - Extensible
  - Integration

Spring Framework

- What is spring framework
- Features of spring
- Lifecycle of a bean
- Dependency Injection (DI) and Inversion of control (IOC)
- Aspect Oriented Programming (AOP)

Why do we need spring ?

- **Simplification of Development**
  - **Boilerplate Code Reduction**: Minimizes repetitive code with features like Dependency Injection and AOP.
  - **Configuration Management**: Simplifies application configuration with annotations and XML.
- **Flexibility and Modularity**
  - **Modular Design**: Use only the needed components (e.g., Spring Core, Spring MVC, Spring Data).
  - **Integration**: Seamlessly integrates with other frameworks and libraries (e.g., Hibernate, JMS).

Why do we need spring ?

- **Enterprise-Grade Features**
  - **Transaction Management**: Simplifies transaction management across various resources.
  - **Security**: Comprehensive security framework for authentication and authorization.
  - **Batch Processing**: Robust support for batch processing and task scheduling.

Why do we need spring ?

- **Community and Ecosystem**
  - **Large Community**: Extensive documentation, community support, and third-party extensions.
  - **Rich Ecosystem**: Includes Spring Boot, Spring Cloud, Spring Data, and more for various enterprise needs.
- **Rapid Development**
  - **Spring Boot**: Simplifies setup and accelerates development with auto-configuration, embedded servers, and opinionated defaults.

Features of Spring

- **Core Features**
  - **Dependency Injection (DI)**: Allows objects to define their dependencies and inject them at runtime, promoting loose coupling and easier testing.
  - **Inversion of Control (IoC)**: The control of object creation and management is transferred from the application to the Spring container.
- **Aspect-Oriented Programming (AOP)**
  - **Cross-Cutting Concerns**: Enables separation of concerns such as logging, security, and transaction management from the business logic.
  - **Declarative Support**: Easily define aspects using annotations or XML configuration.

Core components of Spring Framework

- **Spring Core** - Dependency Injection (DI) and Inversion of Control (IoC) containers.
- **Spring MVC**: Model-View-Controller framework for building web applications.
- **Spring Boot:** Simplifies the setup and development of new Spring applications with convention-over-configuration.

Core components of Spring Framework

- **Spring Data JPA:** Abstraction over JPA for simplifying database interactions.
- **Spring Security:** For securing web applications.
- **Spring Boot:** A project that simplifies the setup and development of new Spring applications with convention-over-configuration.
- **Spring Cloud:** For building microservices and cloud-native applications.

What is a bean ?

- **Definition**: A bean is an object that is instantiated, assembled, and managed by the Spring IoC container.
- **Lifecycle**: The container is responsible for managing the lifecycle of beans, including their creation, initialization, and destruction.

Bean configuration methods

- XML Configuration
  - Typically done using file applicationContext.xml or spring.xml.
- Java Configuration
  - Typically done using a class for config annotated with @Configuration.
- Annotation configuration
  - Done using annotations like @Component, @Service, @Repository, and @Controller to declare beans.

Dependency Injection and Bean scopes

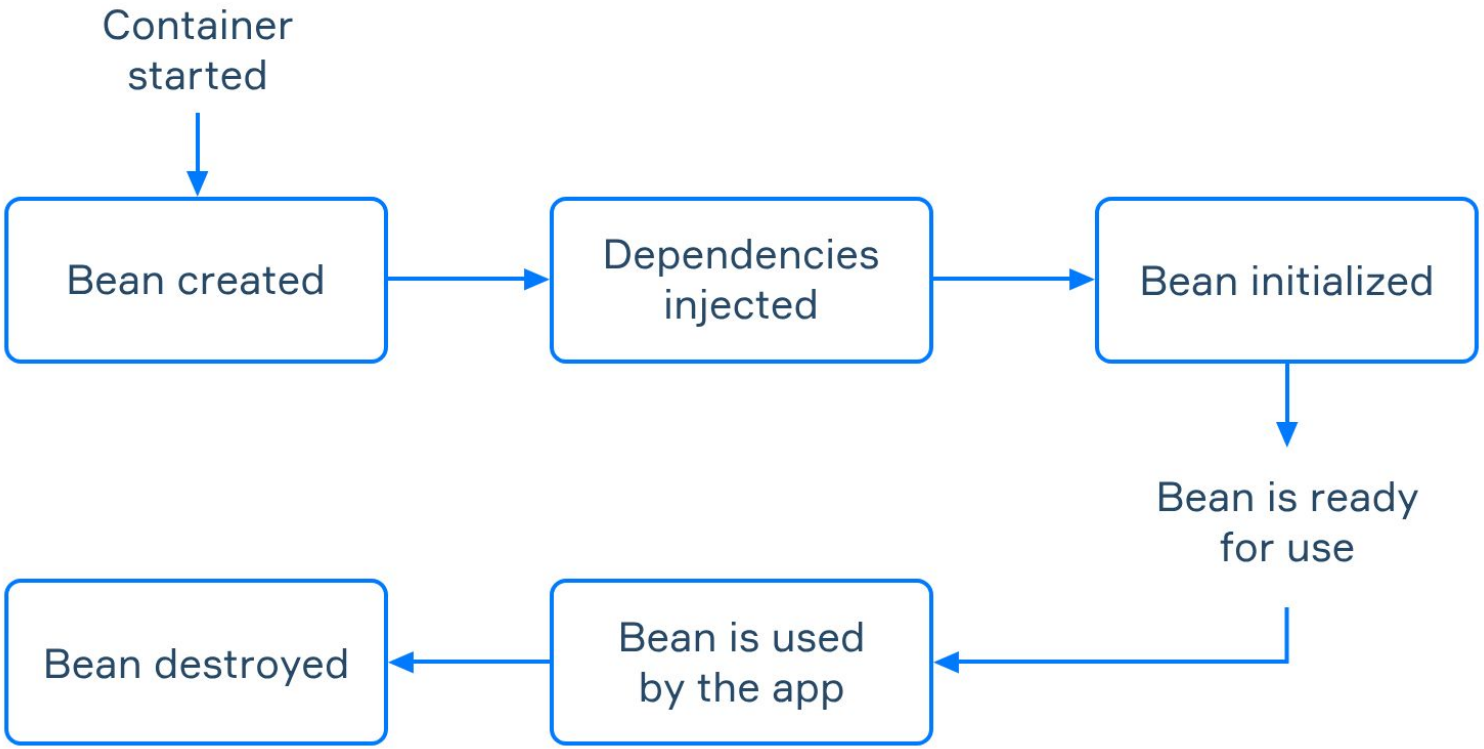
Dependency Injection

- Setter Injection
- Constructor Injection
- Field Injection

Bean Scopes

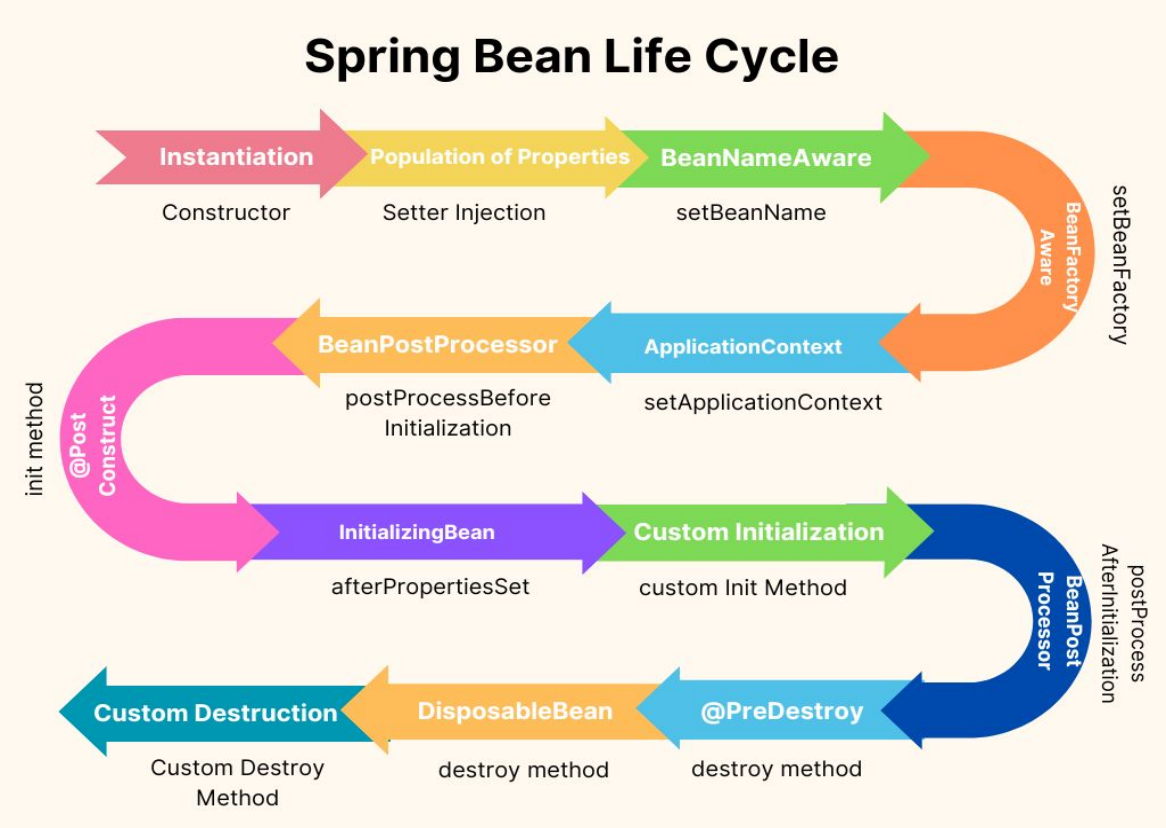
- **Singleton**: Single instance per Spring IoC container (default).
- **Prototype**: New instance every time requested.
- **Request**: Single instance per HTTP request (web applications).
- **Session**: Single instance per HTTP session (web applications).
- **Global Session**: Single instance per global HTTP session (web applications).

Lifecycle of the bean – 1000 ft view



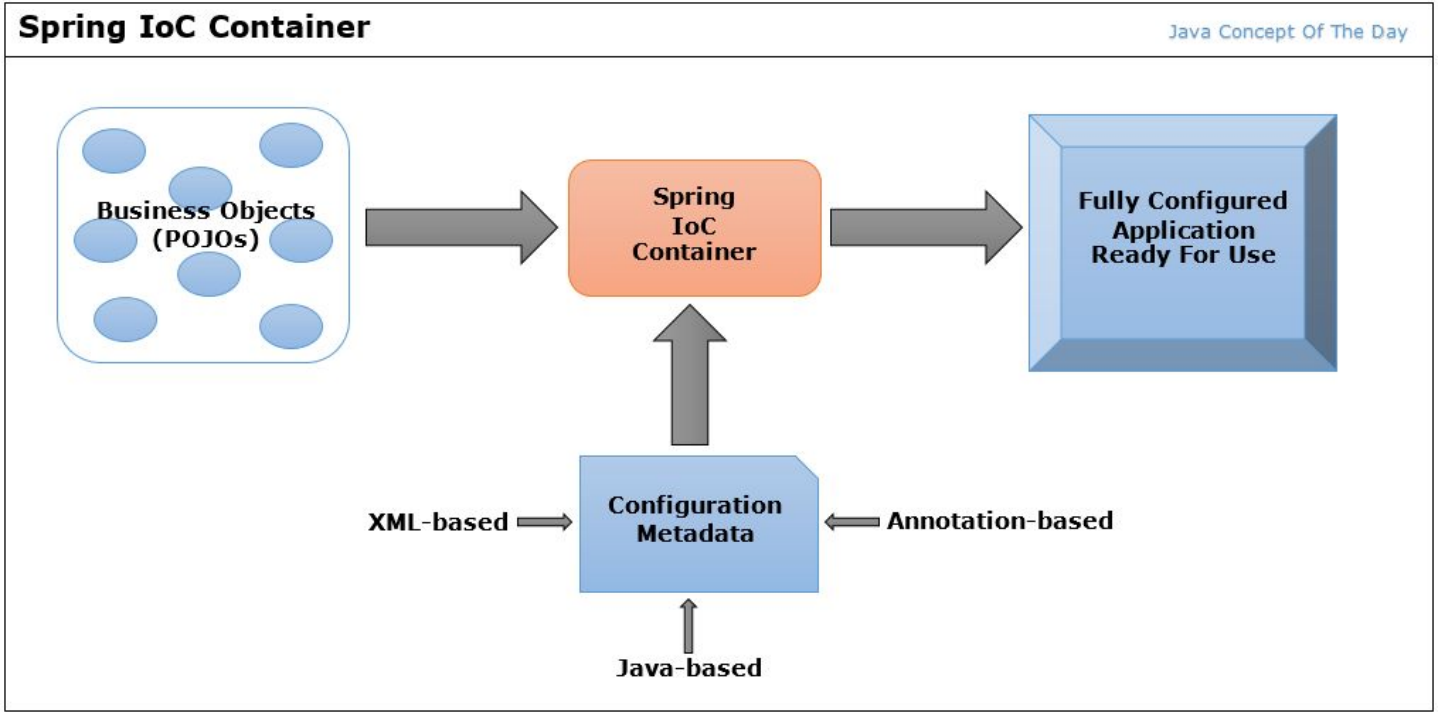
Airtribe

Life cycle of the bean



Airtribe

Spring IOC container



Airtribe

What is spring boot ?

Airtribe

What is spring boot ?

- **Spring Boot** is a project from the Spring team that simplifies the process of creating and running Spring applications by providing production-ready defaults and reducing the need for boilerplate code and configuration.
- **Configuration**: It offers auto-configuration, embedded servers, and a set of opinionated defaults to streamline application setup.

Features of spring boot

- Auto configuration
- Embedded servers
- Spring boot starters
- Production ready features
- Minimal configurations
- Spring boot cli
- Microservices support

JAR vs WAR Packaging

- **JAR (Java Archive)**: Used for standalone applications and libraries, includes the application code, dependencies, and an embedded server (for Spring Boot).

- **WAR (Web Application Archive)**: Used for web applications, includes web-specific configurations and resources, and is typically deployed to an external servlet container.

JAR vs WAR Packaging

- **Deployment**:
  - JAR: Self-contained, can be run directly (e.g., java -jar myapp.jar).
  - WAR: Deployed to a servlet container (e.g., Tomcat, Jetty).
- **Embedded Server**:
  - JAR: Includes an embedded server in Spring Boot applications.
  - WAR: Requires an external servlet container.

Creating a web application – Spring

- **Using Spring:**

1. **Configuration Setup:**
   - Manually configure DispatcherServlet in web.xml or use AbstractAnnotationConfigDispatcherServletInitializer for Java-based configuration.
2. **Application Context:**
   - Define a WebConfig class for component scanning and Spring MVC configuration.
3. **Dependency Management:**
   - Explicitly add dependencies for Spring MVC and other components in pom.xml or build.gradle.

Creating a web application – Spring Boot

- **Using Spring Boot:**

  - **Auto-Configuration:**
    - Spring Boot auto-configures DispatcherServlet and other necessary components, eliminating the need for manual setup.
  - **Simplified Configuration:**
    - Use @SpringBootApplication for automatic component scanning and configuration. application.properties or application.yml simplifies property management.
  - **Dependency Management:**
    - Spring Boot starters provide all required dependencies in a single line, streamlining dependency management.

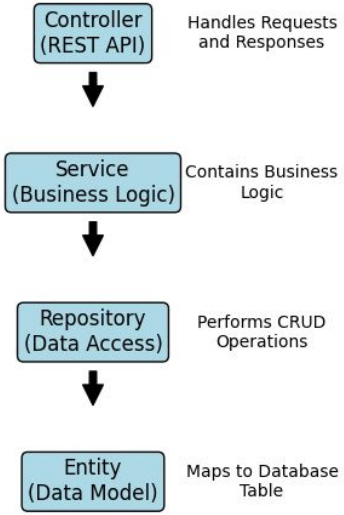
Spring vs Spring Boot

Aspect	Spring	Spring Boot
Setup	Requires manual setup of configuration.	Simplifies setup with auto-configuration and embedded server.
Configuration	Uses XML or Java-based configuration for setting up the application context and components.	Uses application.properties or application.yml for configuration. Auto-configures beans and settings.
Initialization	Requires explicit configuration of DispatcherServlet and other components in web.xml or Java config.	Automatically configures DispatcherServlet and other necessary components. No need for web.xml.
Dependencies	Must manually add dependencies in pom.xml or build.gradle.	Spring Boot starters provide a set of dependencies, reducing manual management.

Spring vs Spring Boot

Aspect	Spring	Spring Boot
Application Class	No specific entry point; configuration is spread across various files.	Requires a main application class annotated with @SpringBootApplication to launch the application.
Development Speed	More configuration and setup required; slower setup process.	Rapid development with minimal configuration; faster setup process.
Auto-Configuration	Requires manual configuration of components and beans.	Automatically configures application components based on dependencies and properties.
Management Endpoints	Requires additional setup for management endpoints (e.g., Actuator).	Provides built-in management endpoints (e.g., /actuator) for monitoring and management.
Learning Curve	Steeper learning curve due to manual configuration and setup.	Easier to learn with reduced configuration and built-in features.
Embedded Server	Requires separate configuration for embedded servers (e.g., Tomcat).	Includes an embedded server (e.g., Tomcat, Jetty) by default.

Visualizing the web application



Airtribe

Creating Simple controller

Airtribe

Creating simple controller

Let's take look at the code

Airtribe

Enabling actuator

Spring Boot Actuator provides several built-in endpoints that can be used to monitor and manage your application. some of the commonly used actuator endpoints are present in the next slide

Application.properties

```
management.endpoints.web.exposure.include=*
management.endpoint.health.show-details=always
```

Actuator

Endpoint	Description
/actuator/health	Health status of the application.
/actuator/info	Arbitrary application info.
/actuator/metrics	Application metrics.
/actuator/env	Environment properties.
/actuator/configprops	Configuration properties.
/actuator/beans	List of all Spring beans.
/actuator/mappings	All @RequestMapping paths.
/actuator/loggers	Logger configurations.
/actuator/httptrace	HTTP trace information.
/actuator/threaddump	JVM thread dump.
/actuator/heapdump	JVM heap dump.
/actuator/scheduledtasks	Scheduled tasks info.
/actuator/caches	Cache information.

Airtribe

Best Practices for Modelling Restful endpoints

Airtribe

Use clear and meaningful resource names

- Choose descriptive and intuitive names for your resources that accurately represent the entities they represent in your system.

Use nouns but no verbs in the URI/path

Purpose	Method	Incorrect	Correct
Retrieves a list of users	GET	/getAllCars	/users
Create a new user	POST	/createUser	/users
Delete a user	DELETE	/deleteUser	/users/10
Get balance of user	GET	/getUserBalance	/users/11/balance

Get method should not alter the state

- Use PUT, POST and DELETE methods instead of GET method to alter the state

- Do not use GET method or query parameters for state changes:

  GET /users/711?activate or

  GET /users/711/activate

Use plural nouns

- Do not mix up singular and plural nouns. Keep it simple and use only plural nouns for all the resources
- /cars insteads of /car

  /users instead of /user

Handle errors with http status codes

- Always return correct http status codes associated with the error handling

- Returning 200 with an error message is an anti-pattern

  Similarly returning 500 with response json is an anti-pattern

Use sub-resources for relations

- If a resource is related to another resource use subresources
- GET /cars/711/drivers (Returns the list of drivers for the car 711)

  GET /cars/711/drivers/4 (Returns driver #4 for car 711)

Provide filtering and sorting operations

- Allow clients to filter and sort resource collections by providing query parameters
- For example, **/products?category=electronics&sort=price** to fetch electronics products sorted by price.

Implement Pagination for large resource collections

- When dealing with resource collections that can become large, provide pagination options to retrieve a subset of the collection at a time. Include query parameters such as page and limit to control the pagination.
- Along with the paginated results, include metadata in the response payload that indicates the total number of items, current page, and links to navigate to the next and previous pages.

Don’t use unnecessary query parameters

- Avoid excessive query parameters that don't add significant value to the resource representation or retrieval. Keep the query parameters relevant and focused on filtering, sorting, or pagination.

Got Questions?

Airtribe

Thank you

Airtribe