Spring In Action

# Core - Internals

1. Springs core is a **container** – **Spring application context**.
2. The container does the magic of **dependency injection** – initializes dependencies and injects them into the components that need them.
3. Automatic configuration = Autowiring + Component Scanning.
4. @Autowired means automatic injection into the constructor. You either need to add the annotation or inject the dependency manually.
5. @SpringBootApplication = @SpringBootConfiguration + @EnableAutoConfiguration + @ComponentScan
6. Spring uses **Tomcat** by default as the web server.
7. Spring devTools provide automatic restart on code change. It has too class loaders that load to the JVM. One contains code. The other contains the dependencies.

# Data – JPA

1. Custom **JPA** queries can be parsed (according to some criteria) in a way that **JPA** understands it. (readByDate – verb + By + predicate)

# Configurations

1. The configurations of spring can be set through the following ways: **JVM properties, OS envs, CLI args and App property configuration files**.
2. It aggregates the configs into a context where the spring beans can be injected.
3. The connection pool uses **Tomcats JDBC**, if not it uses **Hikari**.
4. Setting **server.port** to 0 will generate random port numbers every time.
5. Adding @ConfigurationProperties(prefix="bazz")at the top of the class will make the class referenceable as bazz from outsize of the files and accessing the properties of that class can be done through bazz.property.
6. Configuring **profiles** can make it easier than **envs** or **CLI args** to work differently with **dev** or **prod** environments which can be created using with **Yamls**.

# RESTful

1. Using the @RestController is the same as using the @Controller. But for REST services you should either annotate all the methods with @ResponseBody or return **ResponseEntity.**
2. The @RequestMapping annotation has the produces attribute that refers to the data type returned. ({"application/json", "text/xml"})
3. Methods that apply to uploading methods like POST or PATCH can have a property consumes that refers to the content that is uploaded. ("application/json")
4. The annotation @ResponseStatus replaces the use of **ResponseEntity**’s status code.
5. **HATEOAS** basically means creating self-describing APIs that contain links to other related resources.
6. **HATEOAS** can be easily implemented using the Resource/’s class in order to wrap returned content.
7. The **Data REST** dependency addsREST endpoints according to the data repositories you have. The endpoints are also associated with **HATEOAS.**
8. spring.data.rest.base-path property sets the base path for **Data REST**.
9. **Data REST** pluralizes the name for the entity/repository. (/api/people)
10. @RestResource’s rel and path attributes let you set the entity name and path.
11. Using custom @RestController along with **Data REST** introduces two problems: No-alignment between the paths and non-awareness of the two controllers with each other’s entities.
12. Of the many ways of consuming Springs REST APIs internally, there are mainly three: **RestTemplate**, **Traverson** and **WebClient**.

# Security

1. Spring securities auto-configuration has the following properties: **Authentication prompt for all routes**, **no authorization roles** and **no login page**.

# Messaging

1. With Spring, there are some ways of async messaging: **JMS**, **AMQP**s (**RabbitMQ**) and **Kafka**.
2. Those tools need a **message broker**.
3. **JMS** uses **Apache** **ActiveMQ** or **Artemis**. Both run on **TCP**.
4. In message consumption, there are two ways of handling incoming messages: **Pull Model** and **Push Model**.
5. **Pull Model**: An invoked method that is blocked by a **thread** until a message arrives. (blocking IO)
6. **Push Model**: An **event listener** is registered for when the message arrives it fires. (Non-blocking IO)
7. **JMS** only uses the **Pull Model**.
8. The **Push** **Model** of **JMS** is implemented using the@JmsListener annotation.
9. In **RabbitMQ**, the **queue** is the only thing the consuming app needs to know.
10. **RabbitMQ** has both **Pull** and **Push Models**. In receiving methods, if you set a **timeout** the **Pull Model** is invoked. Else, the **Push Model** is invoked.
11. Annotating a method using@RabbitListener will make the method a **message listener**.
12. Apache Kafka run in clusters which makes it very scalable.
13. Apache kafka’s broker does not push the messages like RabbitMQ. It waits for the consumer to consume them.
14. The way Apache Kafka consumes messages is through listening to messages in a method that is annotated using @KafkaListener.

# 3rd Party Libraries – Lombok

1. **Lombok** is not a Spring library. It is a 3rd party.
2. **Lombok** does the internal initialization of the setters and getters behind.

# Spring Data – JdbcTemplate

1. There are two ways of accessing a database: **JDBC** and **JPA**.
2. All exceptions that might be thrown in accessing database with java cannot be caught with SQLException. Some of them will be thrown for **handling upstream**.
3. Using **Lombok** can save you creating setters and getters in entities.