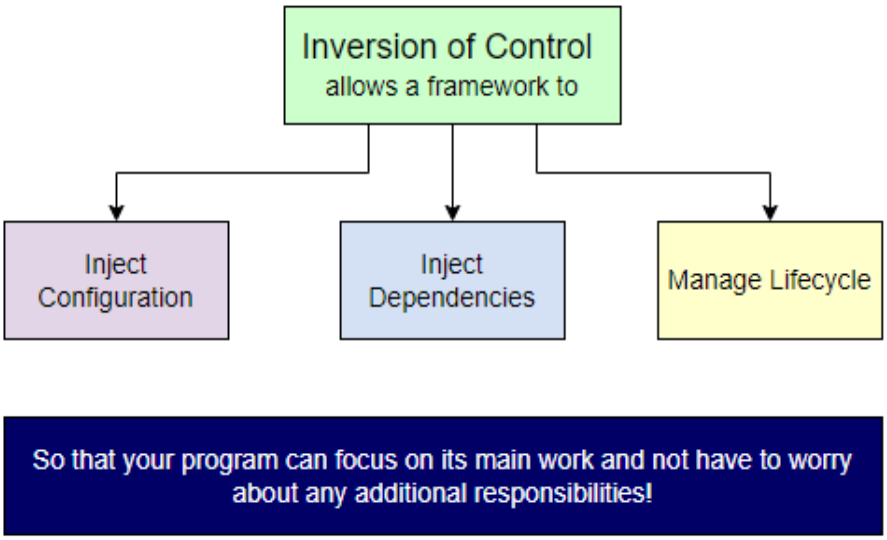
Homework 2- Mustafa YAŞAR

1- IoC and DI means?

a) Inversion Of Control (IoC)

One of the major principles of Software Engineering is that classes should have minimum interdependence, i.e., low coupling. Inversion of Control (IoC) is a design principle that allows classes to be loosely coupled and, therefore, easier to test and maintain. IoC refers to transferring the control of objects and their dependencies from the main program to a container or framework. IoC is a principle, not a design pattern – the implementation details depend on the developer. All IoC does is provide high-level guidelines.



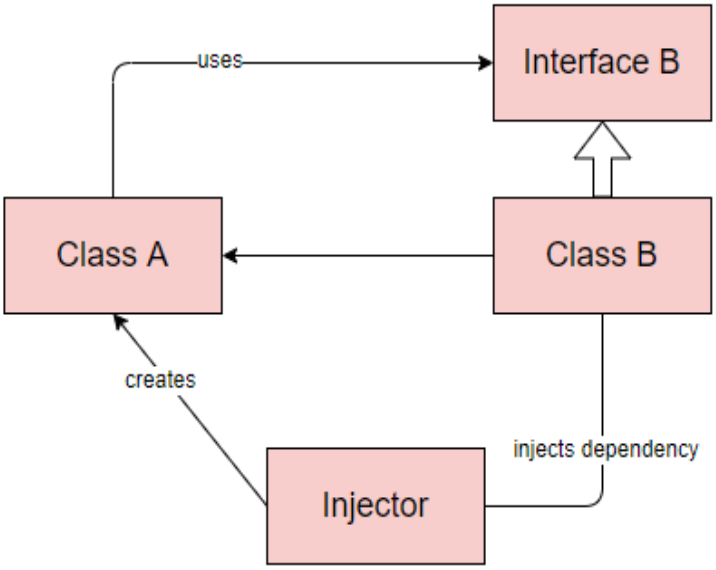
Inversion of Control and Dependency Injection are often used interchangeably. However, Dependency Injection is only one implementation of IoC. Other popular implementations of IoC are:

- Service Locator Pattern
- Event-driven programs

b) Dependencies Injection (DI)

Dependency injection is a technique that allows objects to be separated from the objects they depend upon. For example, suppose object A requires a method of object B to complete its functionality. Well, dependency injection suggests that instead of creating an instance of class B in class A using the new operator, the object of class B should be injected in class A using one of the following methods:

- Constructor injection
- Setter injection
- Interface injection



c) Benefits of IoC

- Reduces amount of application code
- Decreases coupling between classes
- Makes the application easier to test and maintain [1]

2- Spring Bean Scopes?

Spring Bean Scopes allows us to have more granular control of the bean instances creation. Sometimes we want to create bean instance as singleton but in some other cases we might want it to be created on every request or once in a session.

Spring Bean Scopes

Singleton – only one instance of the spring bean will be created for the spring container. This is the default spring bean scope. While using this scope, make sure bean doesn't have shared instance variables otherwise it might lead to data inconsistency issues.

Prototype – A new instance will be created every time the bean is requested from the spring container.

Request – This is same as prototype scope; however, it's meant to be used for web applications. A new instance of the bean will be created for each HTTP request.

Session – A new bean will be created for each HTTP session by the container.

Global-session – This is used to create global session beans for Portlet applications. [2]

3- What does @SpringBootApplication do?

Many Spring Boot developers like their apps to use auto-configuration, component scan and be able to define extra configuration on their "application class". A single @SpringBootApplication annotation can be used to enable those three features, that is:

- @EnableAutoConfiguration: enable Spring Boot's auto-configuration mechanism
- @ComponentScan: enable @Component scan on the package where the application is located (see the best practices)
- @Configuration: allow to register extra beans in the context or import additional configuration classes

The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration, and @ComponentScan with their default attributes, as shown in the following example: [3]

```java
package com.example.myapplication;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication // same as @Configuration @EnableAutoConfiguration @ComponentScan
public class Application {

        public static void main(String[] args) {
                SpringApplication.run(Application.class, args);
        }

}
```

4- What is Spring AOP? Where and how to use it?

The application is generally developed with multiple layers. A typical Java application has the following layers:

- Web Layer: It exposes the services using the REST or web application.
- Business Layer: It implements the business logic of an application.
- Data Layer: It implements the persistence logic of the application.

The responsibility of each layer is different, but there are a few common aspects that apply to all layers are Logging, Security, validation, caching, etc. These common aspects are called cross-cutting concerns.

If we implement these concerns in each layer separately, the code becomes more difficult to maintain. To overcome this problem, Aspect-Oriented Programming (AOP) provides a solution to implement cross-cutting concerns. [4]

- Implement the cross-cutting concern as an aspect.
- Define pointcuts to indicate where the aspect has to be applied.

It ensures that the cross-cutting concerns are defined in one cohesive code component.

a) AOP

AOP (Aspect-Oriented Programming) is a programming pattern that increases modularity by allowing the separation of the cross-cutting concern. These cross-cutting concerns are different from the main business logic. We can add additional behavior to existing code without modification of the code itself.

Spring's AOP framework helps us to implement these cross-cutting concerns.

Using AOP, we define common functionality in one place. We are free to define how and where this functionality is applied without modifying the class to which we are applying the new feature. The cross-cutting concern can now be modularized into special classes, called aspect.

There are two benefits of aspects:

First, the logic for each concern is now in one place instead of scattered all over the codebase.

Second, the business modules only contain code for their primary concern. The secondary concern has been moved to the aspect.

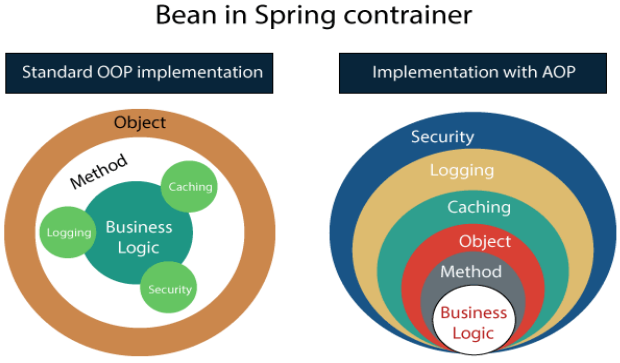
The aspects have the responsibility that is to be implemented, called advice. We can implement an aspect's functionality into a program at one or more join points.

b) Benefits of AOP

- It is implemented in pure Java.
- There is no requirement for a special compilation process.
- It supports only method execution Join points.
- Only run time weaving is available.
- Two types of AOP proxy is available: JDK dynamic proxy and CGLIB proxy.[4]

c) AOP vs OOP

AOP	**OOP**
Aspect: A code unit that encapsulates pointcuts, advices, and attributes.	**Class:** A code unit that encapsulates methods and attributes.
Pointcut: It defines the set of entry points in which advice is executed.	**Method signature:** It defines the entry points for the execution of method bodies.
Advice: It is an implementation of cross-cutting concerns.	**Method bodies:** It is an implementation of the business logic concerns.
Waver: It constructs code (source or object) with advice.	**Compiler: It converts source code to object code.**



d) Spring AOP vs. AspectJ

Spring AOP	AspectJ
There is a need for a separate compilation process.	It requires the AspectJ compiler.
It supports only method execution pointcuts.	It supports all pointcuts.
It can be implemented on beans managed by Spring Container.	It can be implemented on all domain objects.
It supports only method level weaving.	It can wave fields, methods, constructors, static initializers, final class, etc.

5- What is Singleton and where to use it?

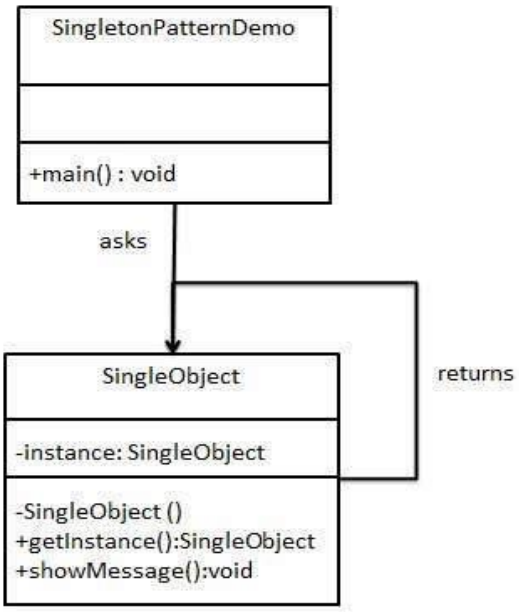
Singleton pattern is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

Implementation

We're going to create a SingleObject class. SingleObject class have its constructor as private and have a static instance of itself.

SingleObject class provides a static method to get its static instance to outside world. SingletonPatternDemo, our demo class will use SingleObject class to get a SingleObject object. [5]



6- What is Spring Boot Actuator and Where to use it?

Spring Boot Actuator Endpoints lets us monitor and interact with our application. Spring Actuator is a Spring Boot sub-module and provides built-in endpoints that we can enable and disable for our application.

Spring Boot Actuator Endpoints

Spring Boot Actuator Endpoints are exposed over JMX and HTTP, most of the times we use HTTP based Actuator endpoints because they are easy to access over the browser, CURL command, shell scripts etc.

Some of the useful actuator endpoints are:

a) beans: this endpoint returns the list of all the beans configured in our application.
b) env: provides information about the Spring Environment properties.
c) health: Shows application health
d) info: Displays application information, we can configure this in Spring environment properties.
e) mappings: Displays the list of all @RequestMapping paths.
f) shutdown: allows us to gracefully shutdown the application.
g) threaddump: provides the thread dump of the application. [6]

7- What is the primary difference between Spring and Spring Boot?

Spring

Spring is an open-source lightweight framework that allows Java EE 7 developers to build simple, reliable, and scalable enterprise applications. This framework mainly focuses on providing various ways to help you manage your business objects. It made the development of Web applications much easier than compared to classic Java frameworks and Application Programming Interfaces (APIs), such as Java database connectivity (JDBC), Java Server Pages (JSP), and Java Servlet. This framework uses various new techniques such as Aspect-Oriented Programming (AOP), Plain Old Java Object (POJO), and dependency injection (DI), to develop enterprise applications.

The Spring framework can be considered as a collection of sub-frameworks, also called layers, such as Spring AOP. Spring Object-Relational Mapping (Spring ORM). Spring Web Flow, and Spring Web MVC. You can use any of these modules separately while constructing a Web application. The modules may also be grouped together to provide better functionalities in a Web application.

Spring Boot

Spring Boot is built on top of the conventional spring framework. So, it provides all the features of spring and is yet easier to use than spring. Spring Boot is a microservice-based framework and making a production-ready application in very less time. In Spring Boot everything is auto-configured. We just need to use proper configuration for utilizing a particular functionality. Spring Boot is very useful if we want to develop REST API. [7]

Spring vs Spring Boot

Spring	Spring Boot
Spring is an open-source lightweight framework widely used to develop enterprise applications.	Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs.
The most important feature of the Spring Framework is dependency injection.	The most important feature of the Spring Boot is Autoconfiguration.
It helps to create a loosely coupled application.	It helps to create a stand-alone application.
To run the Spring application, we need to set the server explicitly.	Spring Boot provides embedded servers such as Tomcat and Jetty etc.
To run the Spring application, a deployment descriptor is required.	There is no requirement for a deployment descriptor.
To create a Spring application, the developers write lots of code.	It reduces the lines of code.
It doesn't provide support for the in-memory database.	It provides support for the in-memory database such as H2.

8- Why to use VCS?

Using version control software is a best practice for high performing software and DevOps teams. Version control also helps developers move faster and allows software teams to preserve efficiency and agility as the team scales to include more developers.

Version Control Systems (VCS) have seen great improvements over the past few decades and some are better than others. VCS are sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System). One of the most popular VCS tools in use today is called Git. Git is a Distributed VCS, a category known as DVCS, more on that later. Like many of the most popular VCS systems available today, Git is free and open source. Regardless of what they are called, or which system is used, the primary benefits you should expect from version control are as follows.

1. A complete long-term change history of every file. This means every change made by many individuals over the years. Changes include the creation and deletion of files as well as edits to their contents. Different VCS tools differ on how well they handle renaming and moving of files. This history should also include the author, date and written notes on the purpose of each change. Having the complete history enables going back to previous versions to help in root cause analysis for bugs and it is crucial when needing to fix problems in older versions of software. If the software is being actively worked on, almost everything can be considered an "older version" of the software.
2. Branching and merging. Having team members work concurrently is a no-brainer, but even individuals working on their own can benefit from the ability to work on independent streams of changes. Creating a "branch" in VCS tools keeps multiple streams of work independent from each other while also providing the facility to merge that work back together, enabling developers to verify that the changes on each branch do not conflict. Many software teams adopt a practice of branching for each feature or perhaps branching for each release, or both. There are many different workflows that teams can choose from when they decide how to make use of branching and merging facilities in VCS.

3. Traceability. Being able to trace each change made to the software and connect it to project management and bug tracking software such as Jira, and being able to annotate each change with a message describing the purpose and intent of the change can help not only with root cause analysis and other forensics. Having the annotated history of the code at your fingertips when you are reading the code, trying to understand what it is doing and why it is so designed can enable developers to make correct and harmonious changes that are in accord with the intended long-term design of the system. This can be especially important for working effectively with legacy code and is crucial in enabling developers to estimate future work with any accuracy.

While it is possible to develop software without using any version control, doing so subjects the project to a huge risk that no professional team would be advised to accept. So, the question is not whether to use version control but which version control system to use. [8]

9- What are SOLID Principles? Give sample usages in Java?

In object-oriented programming languages, the classes are the building blocks of any application. If these blocks are not strong, the building (i.e., the application) is going to face a tough time in the future.

Poorly designed applications can lead the team to very difficult situations when the application scope goes up, or the implementation faces certain design issues either in production or maintenance.

On the other hand, a set of well-designed and written classes can speed up the coding process, while reducing the tech debt and the number of bugs in comparison.

SOLID is the acronym for a set of practices that, when implemented together, makes the code more adaptive to change. Bob Martin and Micah Martin introduced these concepts in their book 'Agile Principles, Patterns, and Practices'.

The acronym was meant to help us remember these principles easily. These principles also form a vocabulary we can use while discussing with other team members or as a part of technical documentation shared in the community.

SOLID principles form the fundamental guidelines for building object-oriented applications that are robust, extensible, and maintainable.

a) Single Responsibility Principle

We may come across one of the principles of object-oriented design, Separation of Concerns (SoC), that conveys a similar idea. The name of the SRP says it all:

"One class should have one and only one responsibility"

In other words, we should write, change, and maintain a class only for one purpose. A class is like a container. We can add any amount of data, fields, and methods into it. However, if we try to achieve too much through a single class, soon that class will become bulky. If we follow SRP, the classes will become compact and neat where each class is responsible for a single problem, task, or concern.

For example, if a given class is a model class, then it should strictly represent only one actor/entity in the application. This kind of design decision will give us the flexibility to make changes in the class, in future without worrying the impacts of changes in other classes.

Similarly, if we are writing service/manager class then the class should contain only that part of methods and nothing else. The service class should not contain even utility global functions related to the module.

Better to separate the global functions in another globally accessible class. This will help in maintaining the class for that particular purpose, and we can decide the visibility of class to a specific module only.

b) Open Closed Principle

OCP is the second principle which we should keep in mind while designing our application. It states:

"Software components should be open for extension, but closed for modification"

It means that the application classes should be designed in such a way that whenever fellow developers want to change the flow of control in specific conditions in application, all they need to extend the class and override some functions and that's it.

If other developers are not able to write the desired behavior due to constraints put by the class, then we should reconsider refactoring the class.

I do not mean here that anybody can change the whole logic of the class, but one should be able to override the options provided by software in a non-harmful way permitted by the software.

c) Liskov's Substitution Principle

LSP is a variation of previously discussed open closed principle. It says:

"Derived types must be completely substitutable for their base types"

LSP means that the classes, fellow developers created by extending our class, should be able to fit in application without failure. This is important when we resort to polymorphic behavior through inheritance.

This requires the objects of the subclasses to behave in the same way as the objects of the superclass. This is mostly seen in places where we do runtime type identification and then cast it to appropriate reference type.

d) Interface Segregation Principle

This principle is my favorite one. ISP is applicable to interfaces as a single responsibility principle holds to classes. ISP says:

"Clients should not be forced to implement unnecessary methods which they will not use"

Take an example. Developer Alex created an interface Reportable and added two methods generateExcel() and generatedPdf(). Now client 'A' wants to use this interface but he intends to use reports only in PDF format and not in excel. Will he be able to use the functionality easily?

NO. He will have to implement both the methods, out of which one is an extra burden put on him by the designer of the software. Either he will implement another method or leave it blank. This is not a good design.

So, what is the solution? Solution is to create two interfaces by breaking the existing one. They should be like PdfReportable and ExcelReportable. This will give the flexibility to users to use only the required functionality only.

e) Dependency Inversion Principle

Most of us are already familiar with the words used in principle's name. DI principle says:

"Depend on abstractions, not on concretions"

In other words. we should design our software in such a way that various modules can be separated from each other using an abstract layer to bind them together.[9]

10-What is RAD model?

When you're building a skyscraper, you can't switch up the design halfway through. You need to follow the original path from start to finish.

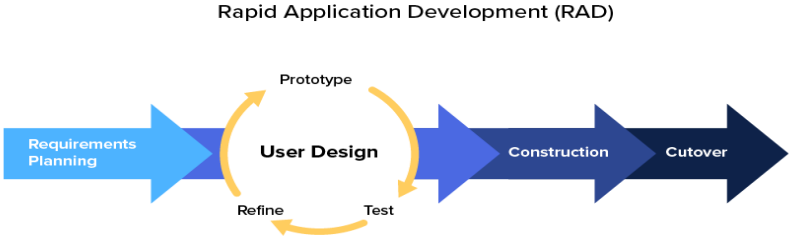
But when you are building software, that isn't the case. You can change the design, add functionality, remove things you don't want, all without harming the end product.

That's a major reason why software needs good development models to be efficient from design to launch. Rapid application development was conceived for this purpose–to develop prototypes rapidly for testing functions and features, without having to worry about how the end product will be affected.

RAD meaning: Rapid Application Development (RAD) is a development model prioritizes rapid prototyping and quick feedback over long-drawn-out development and testing cycles. With rapid application development, developers can make multiple iterations and updates to a software rapidly without needing to start a development schedule from scratch each time.

RAD is a development model that came into existence once developers realized the traditional waterfall model of development wasn't very effective.

A major flaw in the waterfall model is that once the program is in the testing phase, it becomes difficult to change the core functions and features of the software. This essentially leaves you with software that may or may not fit your evolving requirement.



Rapid Application Development (RAD) was conceived in the 1980s, so it's definitely not something new. But unlike the waterfall model, it's not singular. It's a continuous evolution of development philosophies according to the requirement at that particular time.

Initially, Barry Boehm, James Martin, and a number of others saw that software was not limited to traditional methods of engineering. It wasn't a singular resource that required a fixed structure. It was malleable to suit the needs of the user.

Initially, rapid application development took the shape of the Spiral model, where one or more development models were used to work on a particular project.

Over time, rapid application development changed. It molded itself to fit the requirements of the time while retaining some core development guidelines.

Steps in Rapid Application Development

Although RAD has changed over the years, these four basic steps provide some continuity over the years. [10]

- Define the requirements
- Prototype
- Receive Feedback
- Finalize Software

11-What is Spring Boot starter? How is it useful?

Spring Boot provides a number of starters that allow us to add jars in the classpath. Spring Boot built-in starters make development easier and rapid. Spring Boot Starters are the dependency descriptors.

In the Spring Boot Framework, all the starters follow a similar naming pattern: spring-boot-starter-*, where * denotes a particular type of application. For example, if we want to use Spring and JPA for database access, we need to include the spring-boot-starter-data-jpa dependency in our pom.xml file of the project.[11]

12-What is Caching? How can we archive caching in Spring Boot?

Caching is a common practice to improve your system in many ways. It helps to make your system resilient, scalable, fast, and even save some bucks depending on your use case.

If in your database some values are read often then it's a good idea to cache them. This could be the content of the email that you send to a newly registered user, a catchy banner text that changes every day or the number of hours you ban a user from your system for suspicious activity.

If these are accessed too frequently, you may store them in application properties or even in your code, but then you have to redeploy to make the change effective. These are the type of things which are most likely set by the business team, so if they decide to change something it is an overkill to deploy the application. So, you better give them a portal to update these and inside your application you both store them in database and cache such as Redis as well.[12]

13-What & How & Where & Why to logging?

Spring Boot uses Commons Logging for all internal logging but leaves the underlying log implementation open. Default configurations are provided for Java Util Logging, Log4J2, and Logback. In each case, loggers are pre-configured to use console output with optional file output also available.

By default, if you use the “Starters”, Logback is used for logging. Appropriate Logback routing is also included to ensure that dependent libraries that use Java Util Logging, Commons Logging, Log4J, or SLF4J all work correctly.[13]

14-What is Swagger? Have you implemented it using Spring Boot?

Swagger is a set of open-source tools for writing REST-based APIs. It simplifies the process of writing APIs by notches, specifying the standards & providing the tools required to write beautiful, safe, performant & scalable APIs.

In today’s software realm, there are no systems running online without exposing an API. We have moved from monolithic systems to microservices. And the whole design of microservices is laid on REST APIs.

Business can’t afford any loopholes or glitches in the functionality of the APIs after all the entire business rests on them. [14]

Swagger implementation

```
<dependency>
      <groupId>io.springfox</groupId>
      <artifactId>springfox-swagger2</artifactId>
      <version>2.9.2</version>
  </dependency>
```

The above code should be added to the pom.xml file. [15]

15-Reference

[1] https://www.educative.io/edpresso/what-is-inversion-of-control

[2] https://www.journaldev.com/21039/spring-bean-scopes

[3] https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-using-springbootapplication-annotation.html

[4] https://www.javatpoint.com/spring-boot-aop

[5] https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm

[6] https://www.journaldev.com/21597/spring-boot-actuator-endpoints

[7] https://www.geeksforgeeks.org/difference-between-spring-and-spring-boot/

[8] https://medium.com/@lanceharvieruntime/version-control-why-do-we-need-it-1681f4888cec

[9] https://howtodoinjava.com/best-practices/solid-principles/

[10] https://kissflow.com/low-code/rad/rapid-application-development/

[11] https://www.javatpoint.com/spring-boot-starters

[12] https://medium.com/javarevisited/caching-made-easy-in-spring-boot-27d1a545905c

[13] https://docs.spring.io/spring-boot/docs/2.1.18.RELEASE/reference/html/boot-features-logging.html

[14] https://www.scaleyourapp.com/what-is-swagger-and-why-do-you-need-it-for-your-project/

[15] https://medium.com/kodluyoruz/spring-boot-ile-swagger-kullan%C4%B1m%C4%B1-14ae99c6eca7