1 > Spring Boot Starter => inside the starter it has all required dependencies

**Starter dependency** is nothing but a special dependency that aggregates commonly used dependencies for a particular feature. For example, suppose you are building a Spring based web application. For this, you will need to add dependencies for spring-core, spring-web, jackson, etc. You will also need to get the right versions of all these dependencies. On the other hand, if you use Spring Boot, it provides a single dependency called spring-boot-starter-web. You just need to add this. This will automatically add all the dependencies that are required for a building a web application via Spring.

* Spring-boot-starter-web : Used to build a spring based web application. Can also be used to build a REST application
* spring-boot-starter-data-jpa : Used to add Spring Data JPA support
* spring-boot-starter-test : Used to add testing support like JUnit Mockito, etc
* spring-boot-starter-thymeleaf : Used to add Thymeleaf support

2>Spring Boot auto-configuration attempts to automatically configure your Spring application based on the jar dependencies that you have added. For example, if HSQLDB is on your classpath, and you have not manually configured any database connection beans, then Spring Boot auto-configures an in-memory database.

You need to opt-in to auto-configuration by adding the @EnableAutoConfiguration or @SpringBootApplication annotations to one of your @Configuration classes.

Must Read from below URL :

<https://medium.com/@iamssrofficial/how-spring-boot-works-under-the-hood-1e49e4d5bfb3>

<https://medium.com/empathyco/how-spring-boot-autoconfiguration-works-6e09f911c5ce>

----------------------------------------------------------------------------------------------------------------------------

Spring Boot Autoconfiguration detects that Spring MVC is on the classpath as soon as we include Spring Boot Starter Web as a dependency in our project. It configures dispatcherServlet, a default error page, and webjars automatically.

When you add Spring Boot Data JPA Starter, you’ll see that Spring Boot Auto Configuration creates a datasource and an Entity Manager for you.

3>SpringApplication.run() =>

* Creates Application Context
* Register CommandLinePropertySource to expose command line arguments as Spring properties
* Refresh the Application Context , Loading all singleton beans.
* Trigger any CommandLineRunner beans
* Must Read :

<https://medium.com/javarevisited/what-happens-internally-when-you-start-a-spring-boot-application-part1-a683a64c1be8#:~:text=refreshContext(context)%20method%20in%20the,required%20to%20run%20the%20application>.

**Different ways to run spring app class:**

SpringApplication app = new SpringApplication(HelloworldApplication.class);

app.setBannerMode(Mode.OFF);

app.setWebApplicationType(WebApplicationType.NONE);

app.run(args);

4>Dependency management :

In Gradle Project , below 2 plugins are doing the magic of dependency injection

id 'org.springframework.boot' version '2.2.5.RELEASE'

id 'io.spring.dependency-management' version '1.0.9.RELEASE'

**What if we want to exclude some dependencies and our own versions**:

implementation ('org.springframework.boot:spring-boot-starter-web'){

exclude group:'org.springframework.boot', module:'spring-boot-starter-tomcat'

}

implementation ('org.springframework.boot:spring-boot-starter-tomcat:2.2.4.RELEASE')

https://docs.spring.io/dependency-management-plugin/docs/current/reference/html/

<https://www.javatpoint.com/spring-boot-dm>

The following spring-boot-starter-parent inherits automatically when we configure the project.

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.2.2.BUILD-SNAPSHOT</version> <!-- lookup parent from repository -->

<relativePath/>

</parent>

If we don't want to use spring-boot starter-parent dependency, but still want to take the advantage of the dependency management, we can use <scope> tag, as follows:

<dependencyManagement>

<dependencies>

<dependency><!-- Import dependency management from Spring Boot -->

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-dependencies</artifactId>

<version>2.2.2.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

6> Failure Analyzer :

We can use FailureAnalyzer & AbstractFailureAnalyzer to create our own Custom Failure Analyzer

Refer FailureAnalyzeDemo of greelerner in 2023eclise code

7>server port:

java -jar springbootbanner-0.0.1-SNAPSHOT.jar -server.port=8080 **OR**

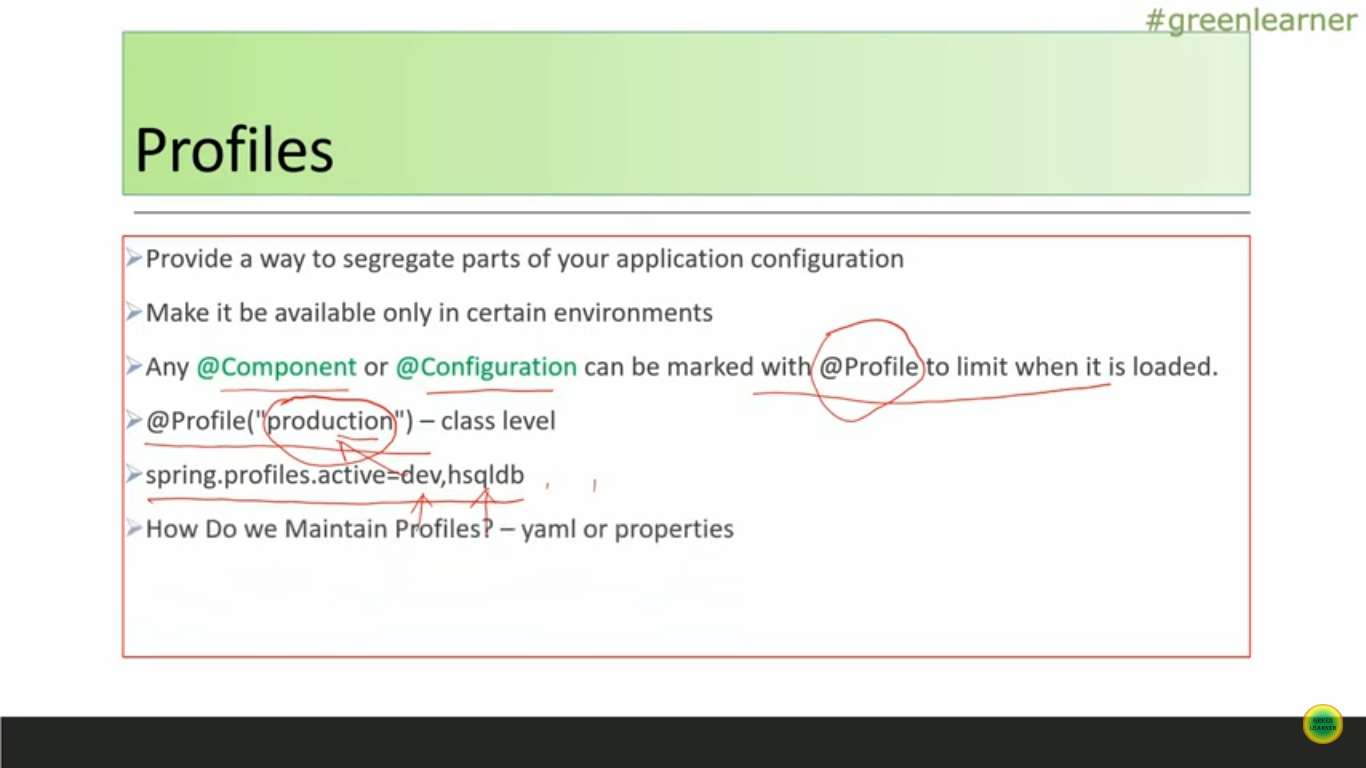
server.port in application.properties

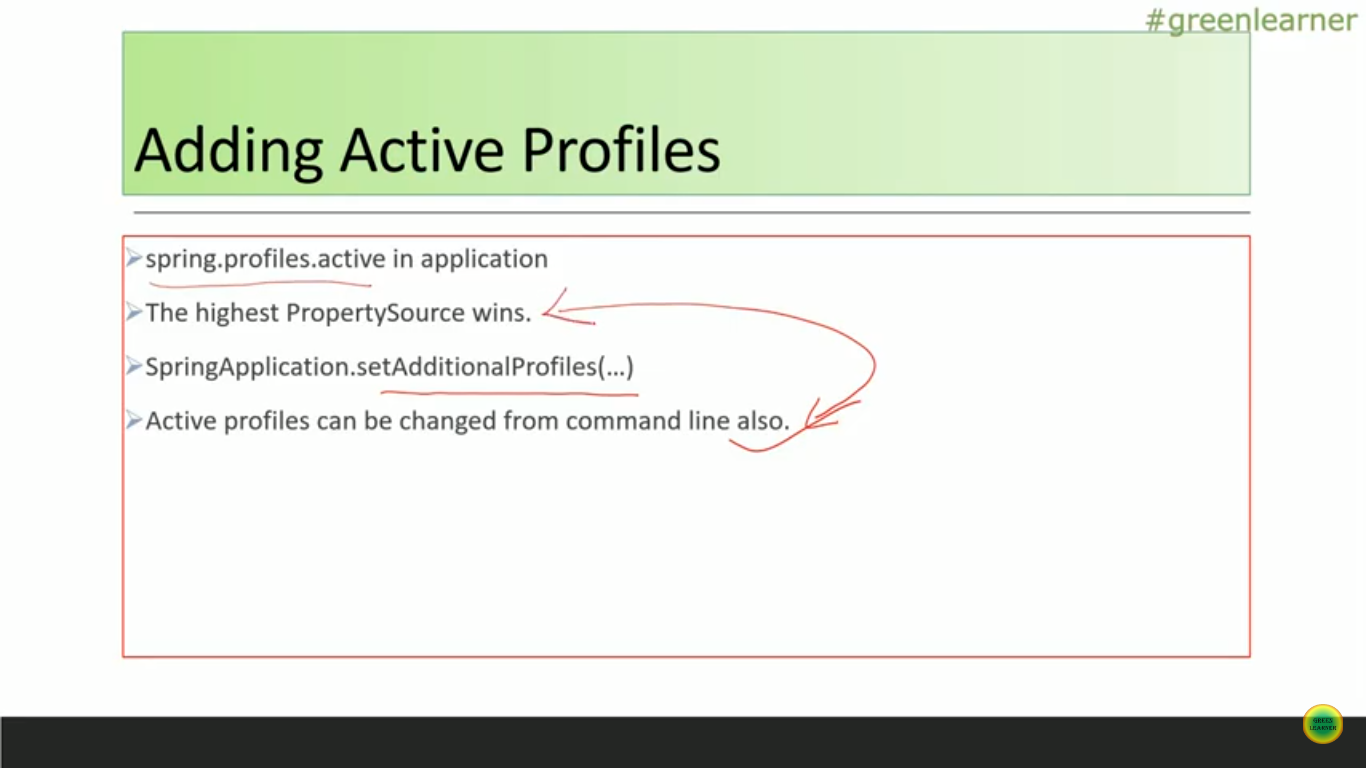
spring profiles -> dev & test

spring.profiles.active = test in application.properties file

then spring boot app will run on port mentioned in application-test.properties which is overriding

the port in application.properties.





Refer profilesDemo project by greenlearner (springBoot\_internal\_working workspace of 2023Eclipse)

We can have active profile in application.properties as well as we can choose it while running application using command Line also.

Command line will override the one mentioned in application.properties file.

F:\Microservices\_GreenLearner\profilesDemo\build\libs>java -jar profilesDemo-0.0.1-SNAPSHOT.jar --spring.profiles.active=prod

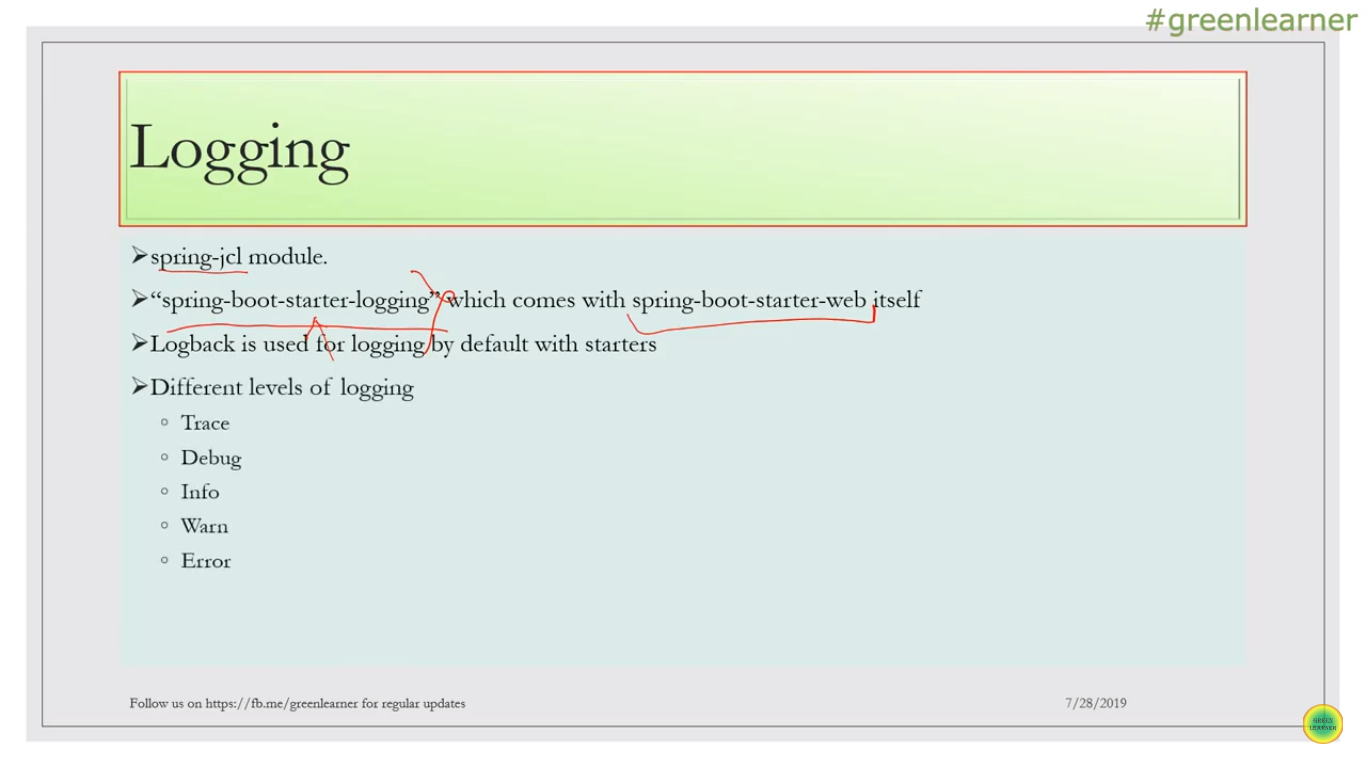
Using @Profile annotation we can configure Spring Beans specific to particular profiles

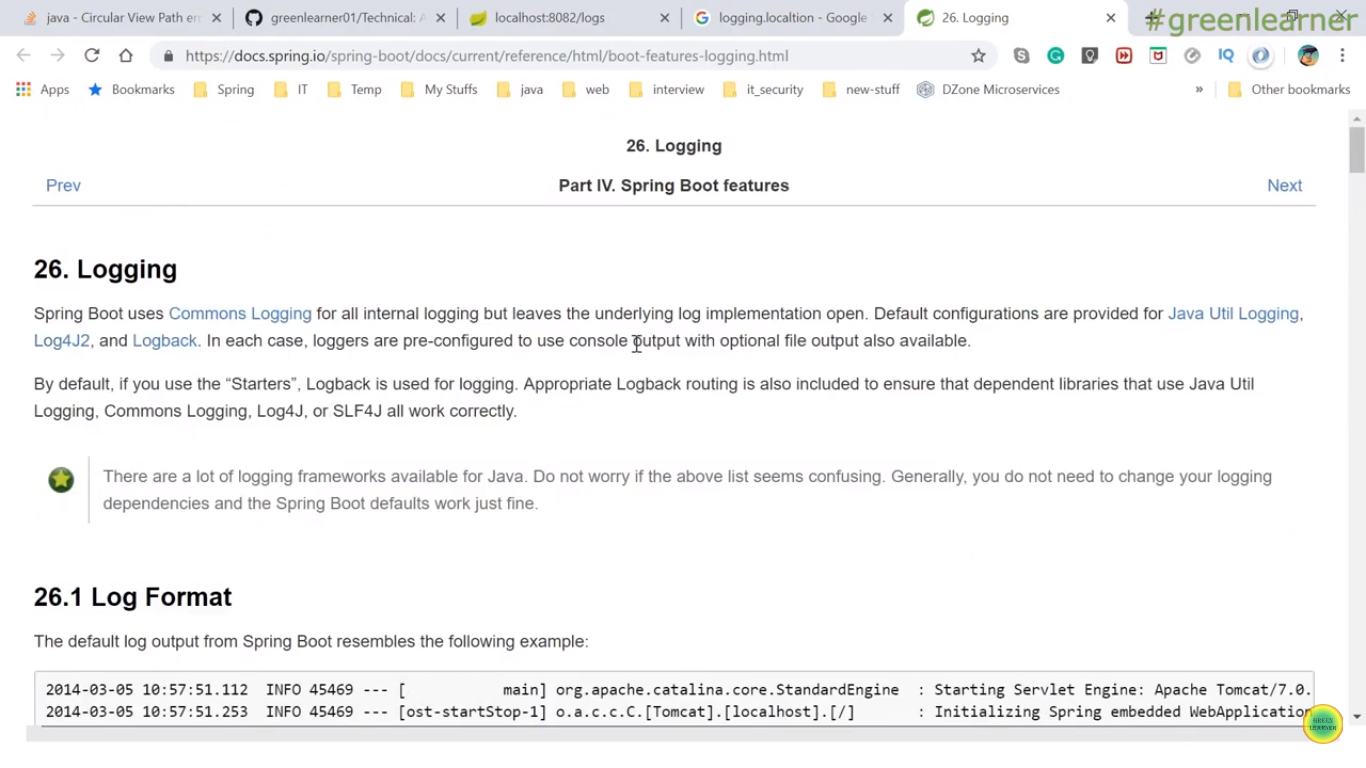
Using @Value annotation we can get the value of the properties mentioned in application.properties file

Check the profile demo project for the same

Run it from eclipse and also using command line

8>Logging





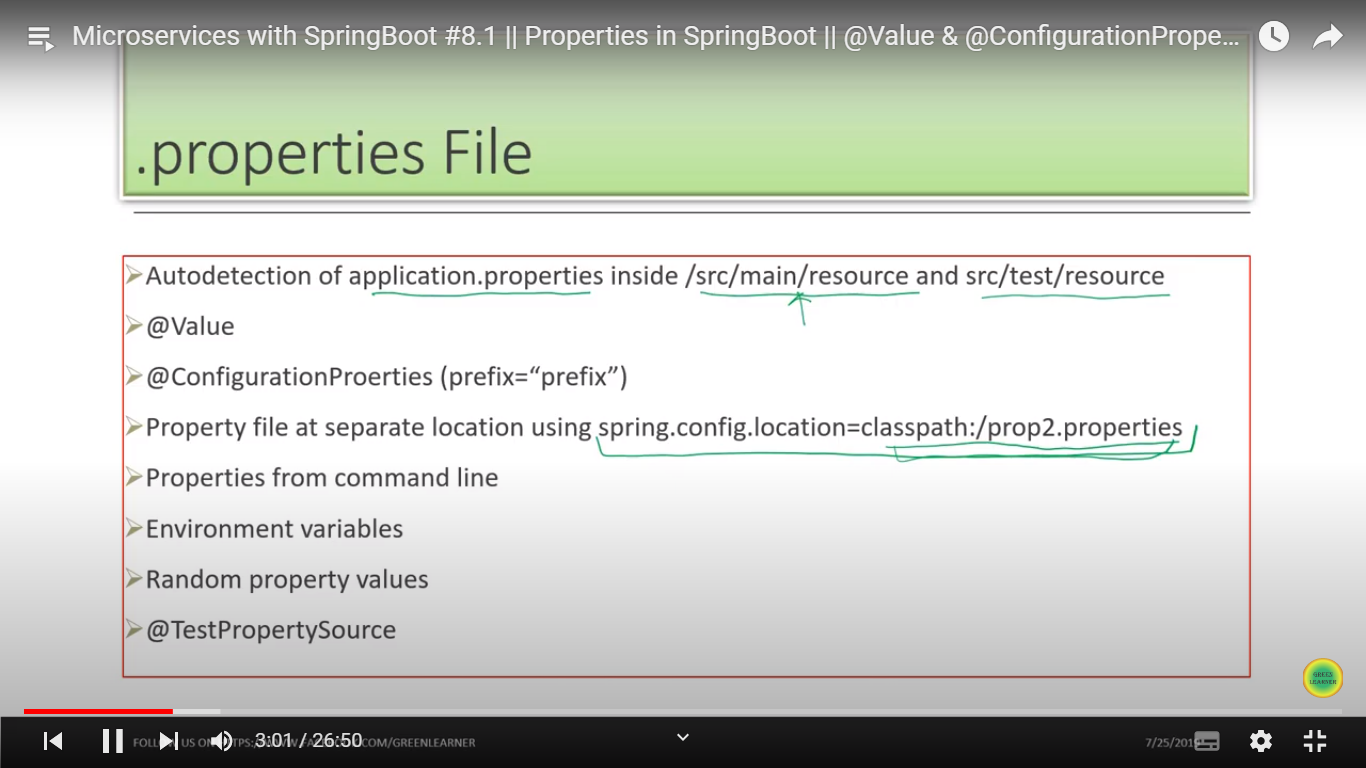
Add below lines in application.properties:

logging.level.root=trace

logging.file=userApp.log

Check User Management App for this

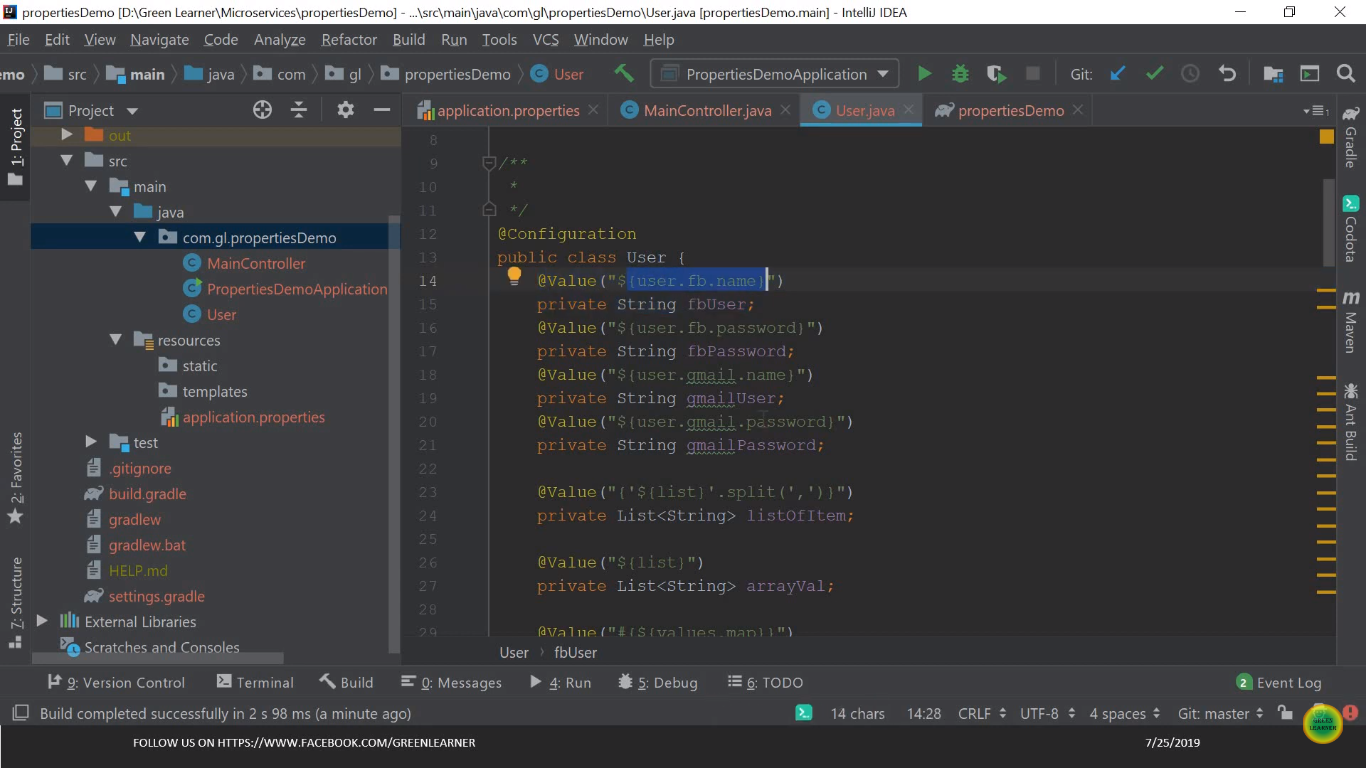
**9> Properties File:**



**Check PropertiesDemo Project by Greenlearner (2023 eclipse)**

@Configuration annotation is used on User class so that it will be registered as Spring Bean

**@Value** : Used to get the values from application.properties file.



How to get the value of List & Map from properties file ?

We need to use “ # “

values.map={key1: '1', key2: '2', key3: '3'}

list = {"a", "b", "1"}

@Value("#{${values.map}}")

private Map<String, Integer> mapValue;

@Value("#{systemProperties}")

private Map<String, String> systemProps;

@Value("#{'${list}'.split(',')}")

private List<String> listOfItem

**How to give default value to property ?**

@Value("${user.fb.name:defaultUser}")

private String fbUser;

**What is @ ConfigurationProperties annoartion ?**

@ConfigurationProperties(prefix="config")

config.port=${random.int}

config.userName=dummy

config.thirdParty.host=109.43.32.43

config.thirdParty.port=90

config.thirdParty.name=user2

Default value will be null for string

**How to change Properties From CommandLine :**

Create jar of project and execute below command :

Go to F:\Microservices\_GreenLearner\propertiesDemo\build\libs

java -jar -Dserver.port=8083 propertiesDemo-0.0.1-SNAPSHOT.jar

java -jar -Dconfig.host=10.10.10.27 propertiesDemo-0.0.1-SNAPSHOT.jar  
-----------------------------------------------------------------------------------------------------

How to give many properties value from command line ?

We can add those to 1 file and refer that at command line

java -jar propertiesDemo-0.0.1-SNAPSHOT.jar --spring.config.location=file:///F:/Microservices\_GreenLearner/propertiesDemo/build/libs/yogesh.properties

try to hit this and test : localhost:8080/userConfig as we have not mentioned nay port in external config fie .default port of 8080 is used

java -jar propertiesDemo-0.0.1-SNAPSHOT.jar --spring.config.location-file:d:/yogesh.properties => this did not work

java -jar propertiesDemo-0.0.1-SNAPSHOT.jar --spring.config.additional-location-file:d:/yogesh.properties

**Spring Boot annotations :**

