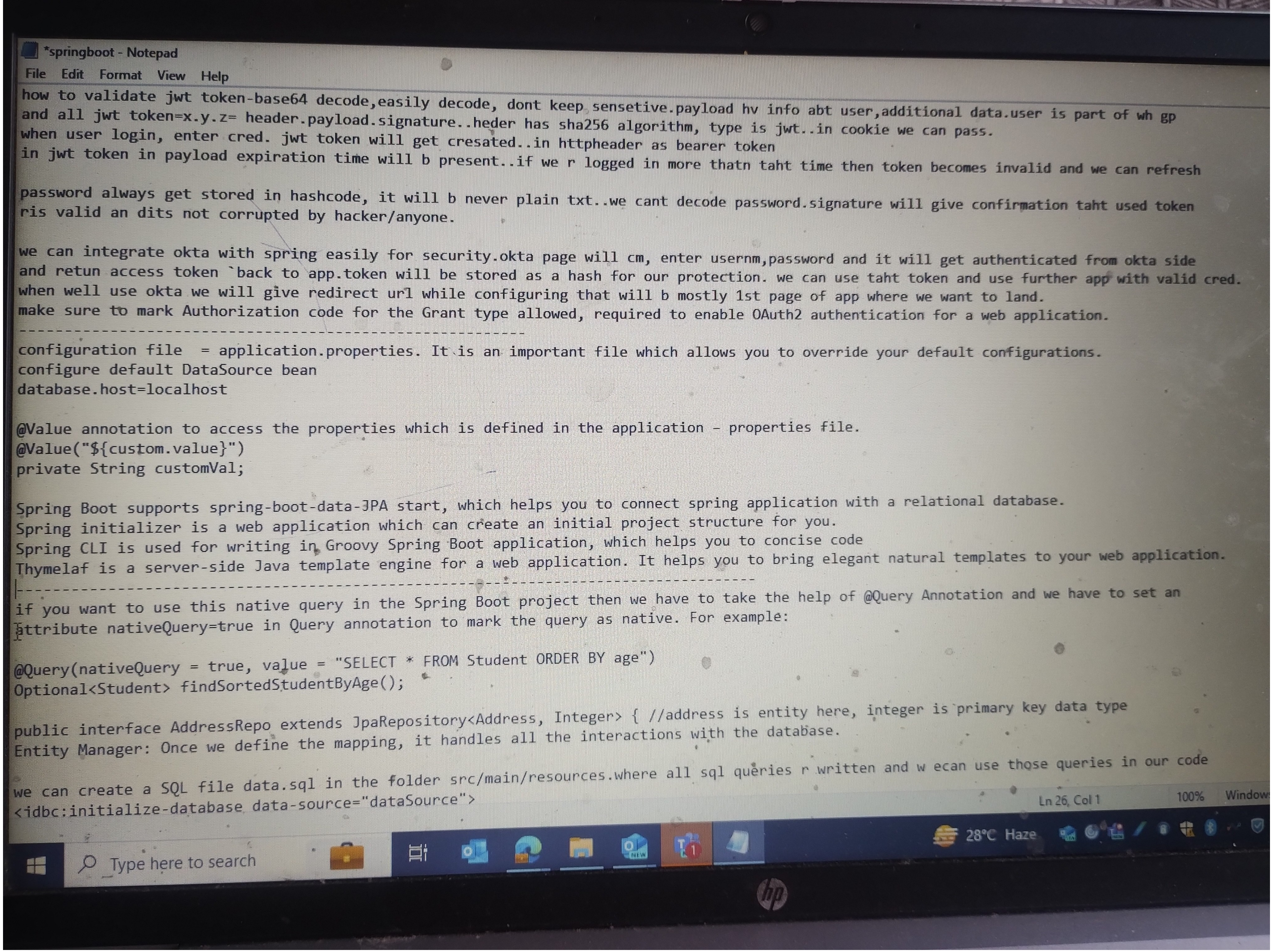
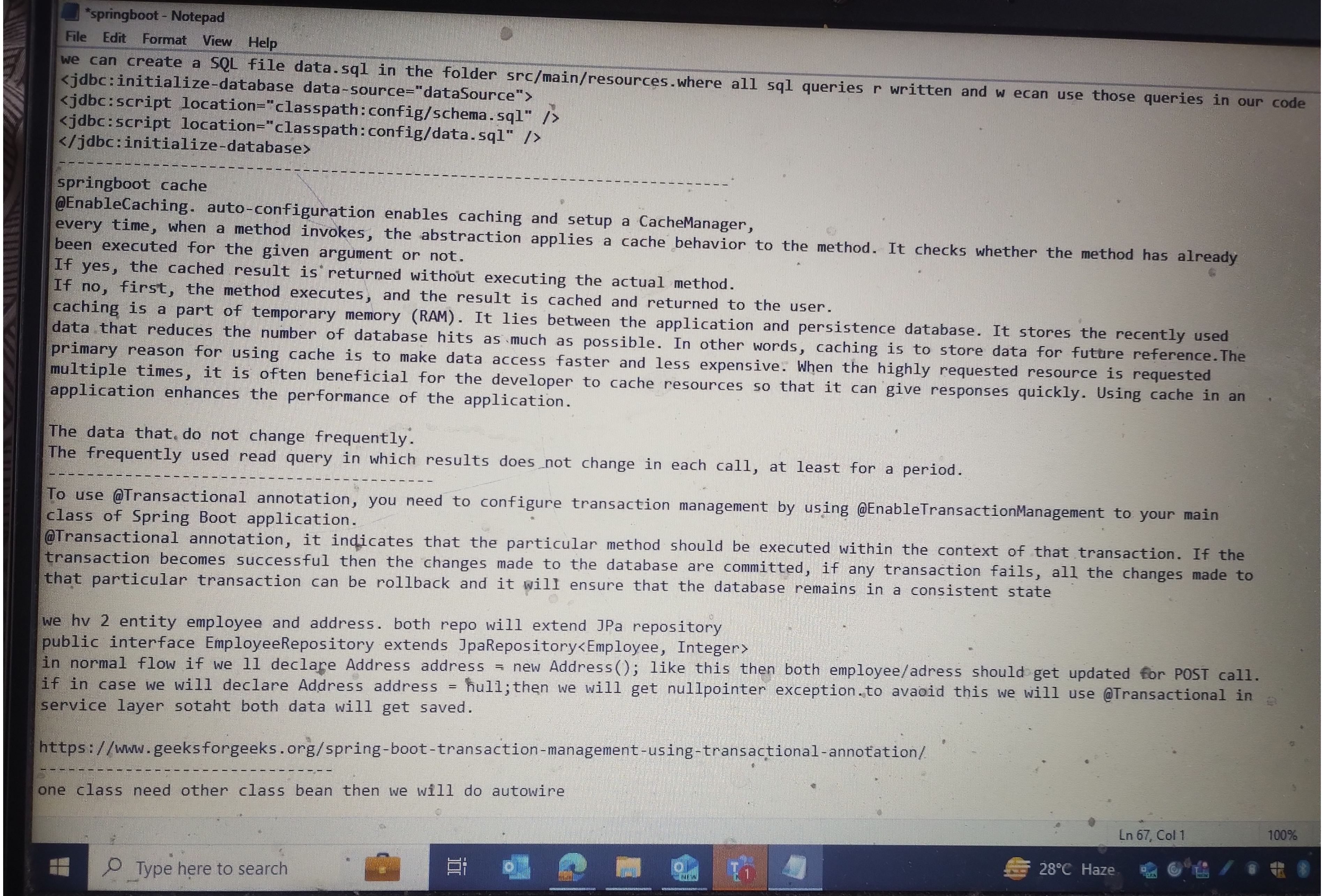


```
how to validate jwt token-base64 decode,easily decode, dont keep sensetive.payload hv info abt user,additional data.user is part of wh gp
and all jwt token=x.y.z= header.payload.signature..heder has sha256 algorithm, type is jwt..in cookie we can pass.
when user login, enter cred. jwt token will get cresated..in httpheader as bearer token
in jwt token in payload expiration time will b present..if we r logged in more thatn taht time then token becomes invalid and we can refresh

password always get stored in hashcode, it will b never plain txt..we cant decode password.signature will give confirmation taht used token
ris valid an dits not corrupted by hacker/anyone.

we can integrate okta with spring easily for security.okta page will cm, enter usernm,password and it will get authenticated from okta side
and retun access token `back to app.token will be stored as a hash for our protection. we can use taht token and use further app with valid cred.
when well use okta we will give redirect url while configuring that will b mostly 1st page of app where we want to land.
make sure to mark Authorization code for the Grant type allowed, required to enable OAuth2 authentication for a web application.
-------------------------------------------------------------
configuration file  = application.properties. It is an important file which allows you to override your default configurations.
configure default DataSource bean
database.host=localhost

@Value annotation to access the properties which is defined in the application - properties file.
@Value("${custom.value}")
private String customVal;

Spring Boot supports spring-boot-data-JPA start, which helps you to connect spring application with a relational database.
Spring initializer is a web application which can create an initial project structure for you.
Spring CLI is used for writing in Groovy Spring Boot application, which helps you to concise code
Thymelaf is a server-side Java template engine for a web application. It helps you to bring elegant natural templates to your web application.
-----------------------------------------------------------------------------------
if you want to use this native query in the Spring Boot project then we have to take the help of @Query Annotation and we have to set an
attribute nativeQuery=true in Query annotation to mark the query as native. For example:

@Query(nativeQuery = true, value = "SELECT * FROM Student ORDER BY age")
Optional<Student> findSortedStudentByAge();

public interface AddressRepo extends JpaRepository<Address, Integer> { //address is entity here, integer is primary key data type
Entity Manager: Once we define the mapping, it handles all the interactions with the database.

we can create a SQL file data.sql in the folder src/main/resources.where all sql queries r written and w ecan use those queries in our code
<idbc:initialize-database data-source="dataSource">
```



we can create a SQL file data.sql in the folder src/main/resources.where all sql queries r written and w ecan use those queries in our code

```
<jdbc:initialize-database data-source="dataSource">
<jdbc:script location="classpath:config/schema.sql" />
<jdbc:script location="classpath:config/data.sql" />
</jdbc:initialize-database>
```

---

springboot cache

@EnableCaching. auto-configuration enables caching and setup a CacheManager,

every time, when a method invokes, the abstraction applies a cache behavior to the method. It checks whether the method has already been executed for the given argument or not.

If yes, the cached result is returned without executing the actual method.

If no, first, the method executes, and the result is cached and returned to the user.

caching is a part of temporary memory (RAM). It lies between the application and persistence database. It stores the recently used data that reduces the number of database hits as much as possible. In other words, caching is to store data for future reference.The primary reason for using cache is to make data access faster and less expensive. When the highly requested resource is requested multiple times, it is often beneficial for the developer to cache resources so that it can give responses quickly. Using cache in an application enhances the performance of the application.

The data that do not change frequently.

The frequently used read query in which results does not change in each call, at least for a period.

---

To use @Transactional annotation, you need to configure transaction management by using @EnableTransactionManagement to your main class of Spring Boot application.

@Transactional annotation, it indicates that the particular method should be executed within the context of that transaction. If the transaction becomes successful then the changes made to the database are committed, if any transaction fails, all the changes made to that particular transaction can be rollback and it will ensure that the database remains in a consistent state

we hv 2 entity employee and address. both repo will extend JPa repository

```
public interface EmployeeRepository extends JpaRepository<Employee, Integer>
```

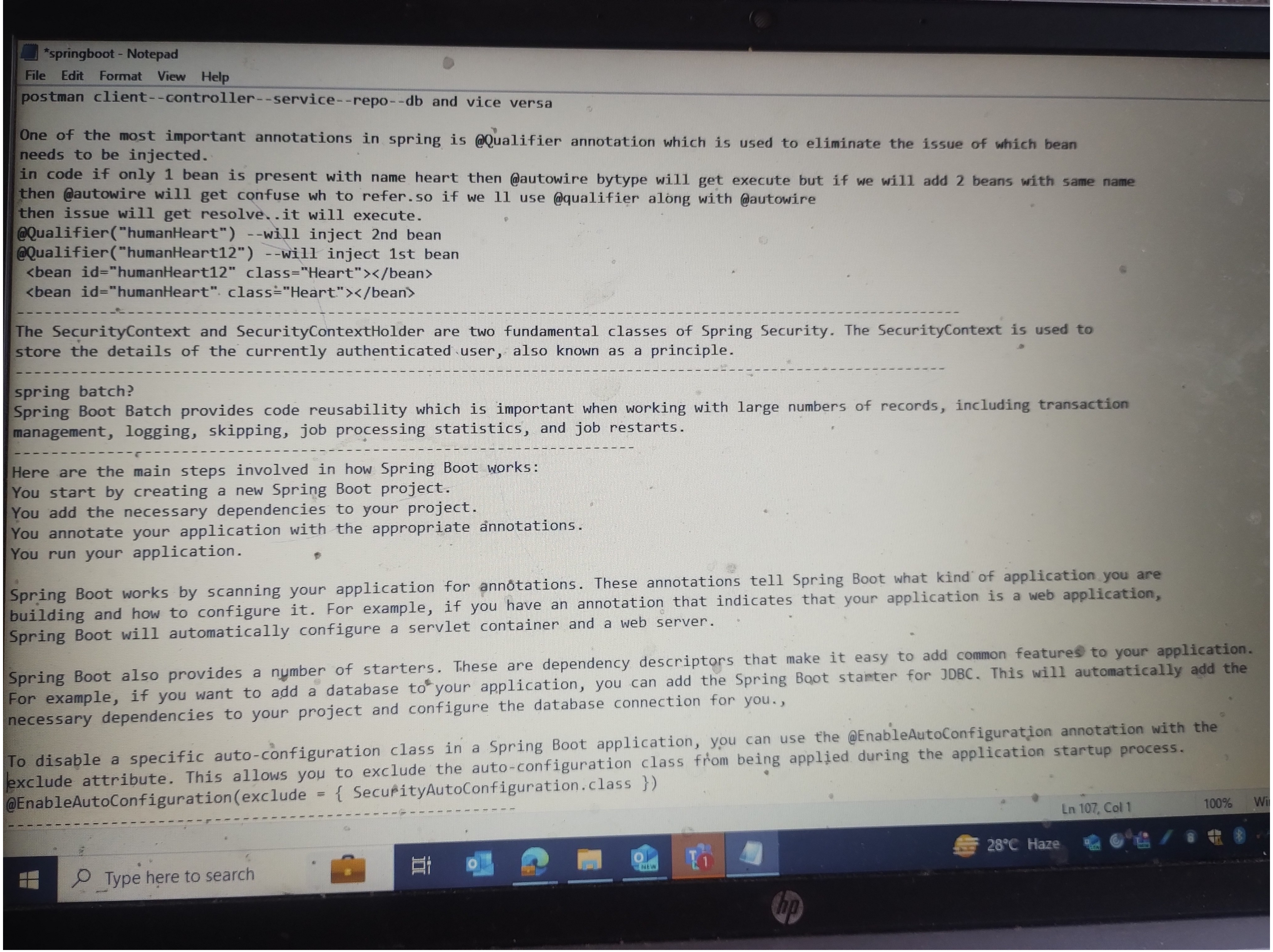
in normal flow if we ll declare Address address = new Address(); like this then both employee/adress should get updated for POST call.

if in case we will declare Address address = null;then we will get nullpointer exception.to avaoid this we will use @Transactional in service layer sotaht both data will get saved.

https://www.geeksforgeeks.org/spring-boot-transaction-management-using-transactional-annotation/

---

one class need other class bean then we will do autowire



postman client--controller--service--repo--db and vice versa

One of the most important annotations in spring is @Qualifier annotation which is used to eliminate the issue of which bean needs to be injected.
in code if only 1 bean is present with name heart then @autowire bytype will get execute but if we will add 2 beans with same name then @autowire will get confuse wh to refer.so if we ll use @qualifier along with @autowire then issue will get resolve..it will execute.

@Qualifier("humanHeart") --will inject 2nd bean
@Qualifier("humanHeart12") --will inject 1st bean

```
 <bean id="humanHeart12" class="Heart"></bean>
 <bean id="humanHeart" class="Heart"></bean>
```

---

The SecurityContext and SecurityContextHolder are two fundamental classes of Spring Security. The SecurityContext is used to store the details of the currently authenticated user, also known as a principle.

---

spring batch?
Spring Boot Batch provides code reusability which is important when working with large numbers of records, including transaction management, logging, skipping, job processing statistics, and job restarts.

---

Here are the main steps involved in how Spring Boot works:
You start by creating a new Spring Boot project.
You add the necessary dependencies to your project.
You annotate your application with the appropriate annotations.
You run your application.

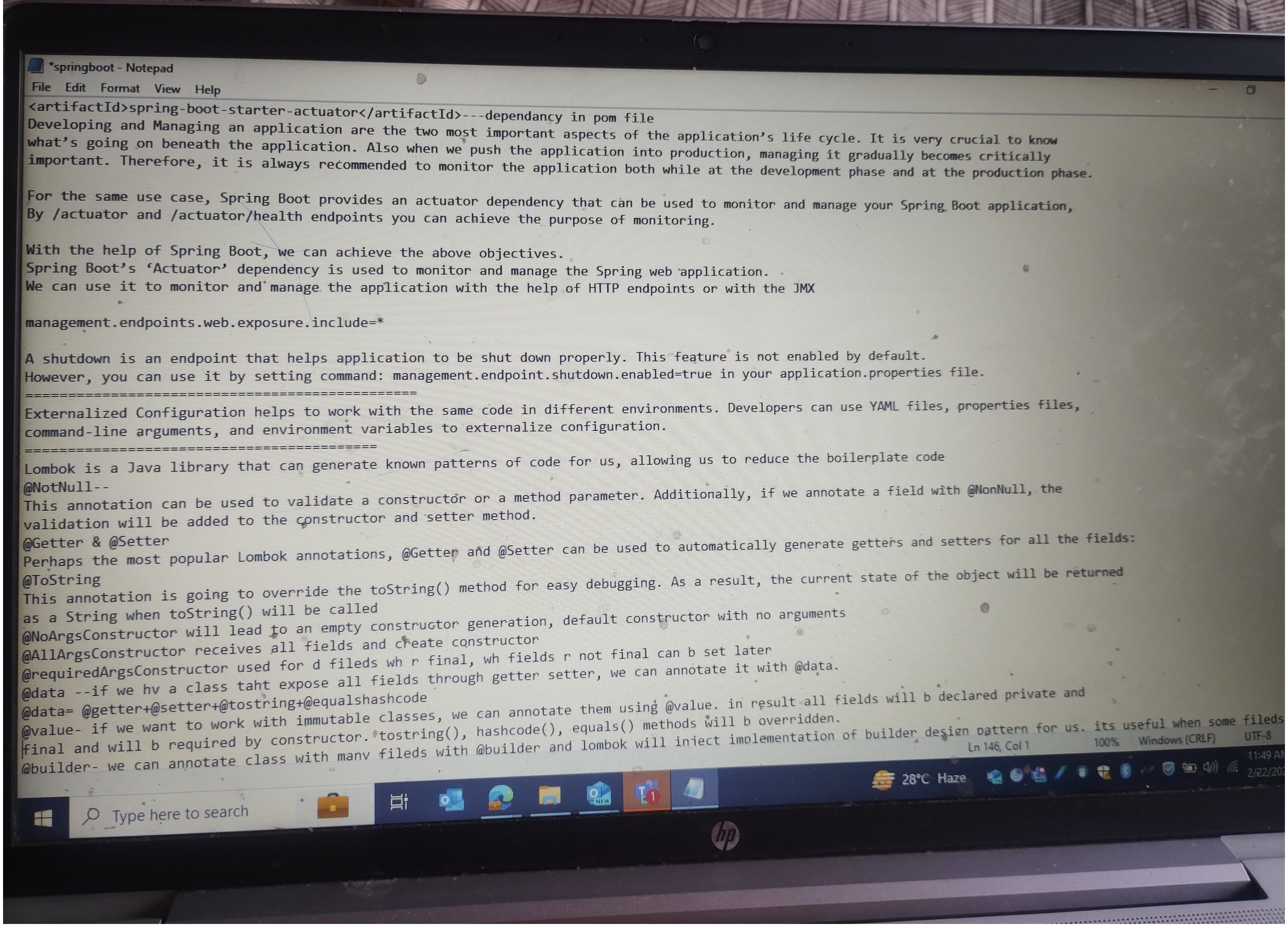
Spring Boot works by scanning your application for annotations. These annotations tell Spring Boot what kind of application you are building and how to configure it. For example, if you have an annotation that indicates that your application is a web application, Spring Boot will automatically configure a servlet container and a web server.

Spring Boot also provides a number of starters. These are dependency descriptors that make it easy to add common features to your application. For example, if you want to add a database to your application, you can add the Spring Boot starter for JDBC. This will automatically add the necessary dependencies to your project and configure the database connection for you.,

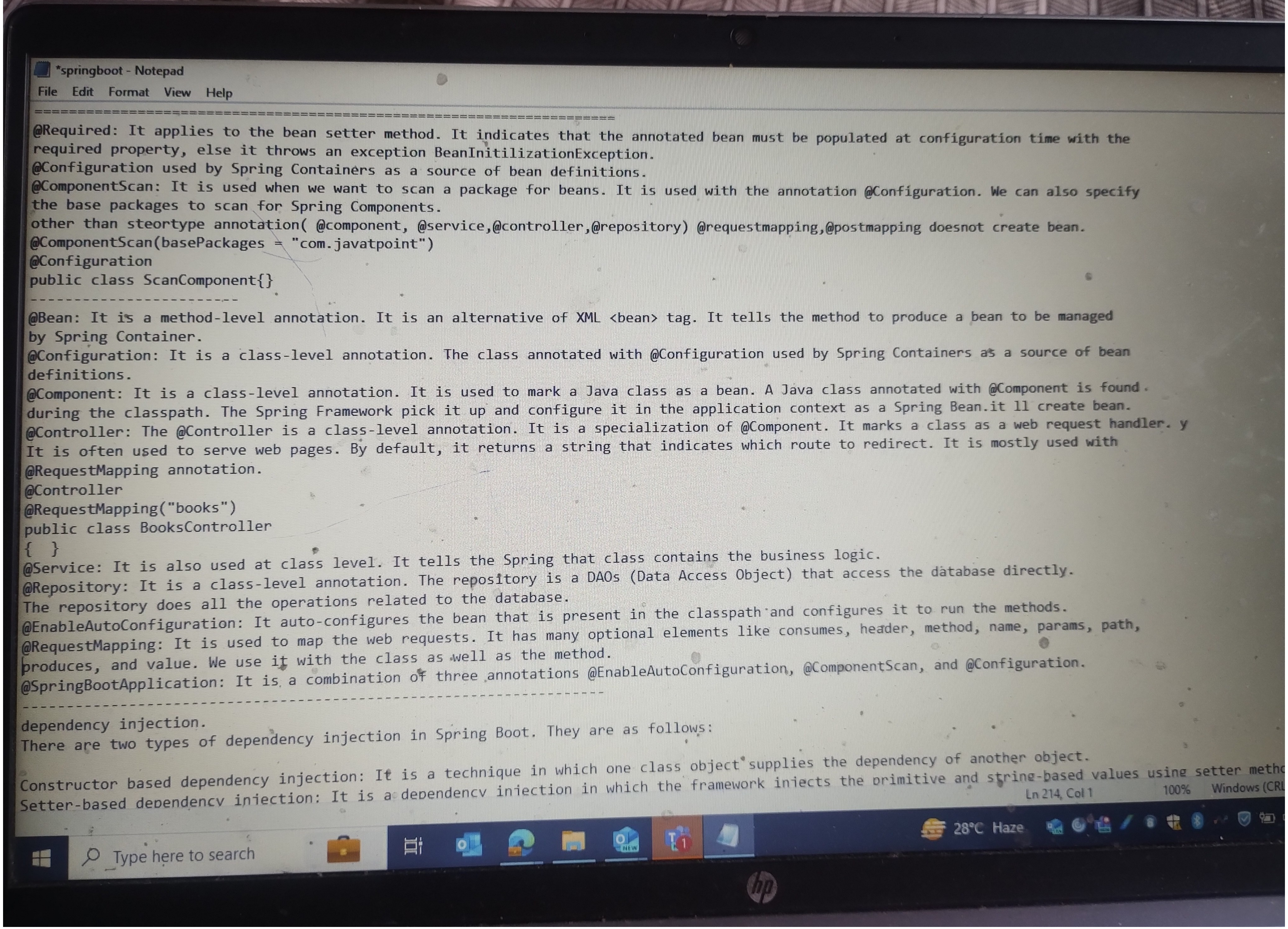
To disable a specific auto-configuration class in a Spring Boot application, you can use the @EnableAutoConfiguration annotation with the exclude attribute. This allows you to exclude the auto-configuration class from being applied during the application startup process.

```
@EnableAutoConfiguration(exclude = { SecurityAutoConfiguration.class })
```

---



```
<artifactId>spring-boot-starter-actuator</artifactId>---dependancy in pom file
Developing and Managing an application are the two most important aspects of the application’s life cycle. It is very crucial to know
what’s going on beneath the application. Also when we push the application into production, managing it gradually becomes critically
important. Therefore, it is always recommended to monitor the application both while at the development phase and at the production phase.

For the same use case, Spring Boot provides an actuator dependency that can be used to monitor and manage your Spring Boot application,
By /actuator and /actuator/health endpoints you can achieve the purpose of monitoring.

With the help of Spring Boot, we can achieve the above objectives.
Spring Boot’s ‘Actuator’ dependency is used to monitor and manage the Spring web application.
We can use it to monitor and manage the application with the help of HTTP endpoints or with the JMX

management.endpoints.web.exposure.include=*

A shutdown is an endpoint that helps application to be shut down properly. This feature is not enabled by default.
However, you can use it by setting command: management.endpoint.shutdown.enabled=true in your application.properties file.
=====================================================
Externalized Configuration helps to work with the same code in different environments. Developers can use YAML files, properties files,
command-line arguments, and environment variables to externalize configuration.
================================================
Lombok is a Java library that can generate known patterns of code for us, allowing us to reduce the boilerplate code
@NotNull--
This annotation can be used to validate a constructor or a method parameter. Additionally, if we annotate a field with @NonNull, the
validation will be added to the constructor and setter method.
@Getter & @Setter
Perhaps the most popular Lombok annotations, @Getter and @Setter can be used to automatically generate getters and setters for all the fields:
@ToString
This annotation is going to override the toString() method for easy debugging. As a result, the current state of the object will be returned
as a String when toString() will be called
@NoArgsConstructor will lead to an empty constructor generation, default constructor with no arguments
@AllArgsConstructor receives all fields and create constructor
@requiredArgsConstructor used for d fileds wh r final, wh fields r not final can b set later
@data --if we hv a class taht expose all fields through getter setter, we can annotate it with @data.
@data= @getter+@setter+@tostring+@equalshashcode
@value- if we want to work with immutable classes, we can annotate them using @value. in result all fields will b declared private and
final and will b required by constructor. tostring(), hashcode(), equals() methods will b overridden.
@builder- we can annotate class with many fileds with @builder and lombok will inject implementation of builder design pattern for us. its useful when some fileds
```



```
=======================================================================
@Required: It applies to the bean setter method. It indicates that the annotated bean must be populated at configuration time with the
required property, else it throws an exception BeanInitilizationException.
@Configuration used by Spring Containers as a source of bean definitions.
@ComponentScan: It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify
the base packages to scan for Spring Components.
other than steortype annotation( @component, @service,@controller,@repository) @requestmapping,@postmapping doesnot create bean.
@ComponentScan(basePackages = "com.javatpoint")
@Configuration
public class ScanComponent{}
------------------------
@Bean: It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed
by Spring Container.
@Configuration: It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean
definitions.
@Component: It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with @Component is found
during the classpath. The Spring Framework pick it up and configure it in the application context as a Spring Bean.it ll create bean.
@Controller: The @Controller is a class-level annotation. It is a specialization of @Component. It marks a class as a web request handler. y
It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with
@RequestMapping annotation.
@Controller
@RequestMapping("books")
public class BooksController
{  }
@Service: It is also used at class level. It tells the Spring that class contains the business logic.
@Repository: It is a class-level annotation. The repository is a DAOs (Data Access Object) that access the database directly.
The repository does all the operations related to the database.
@EnableAutoConfiguration: It auto-configures the bean that is present in the classpath and configures it to run the methods.
@RequestMapping: It is used to map the web requests. It has many optional elements like consumes, header, method, name, params, path,
produces, and value. We use it with the class as well as the method.
@SpringBootApplication: It is a combination of three annotations @EnableAutoConfiguration, @ComponentScan, and @Configuration.
-----------------------------------------------------------------

dependency injection.
There are two types of dependency injection in Spring Boot. They are as follows:

Constructor based dependency injection: It is a technique in which one class object supplies the dependency of another object.
Setter-based dependency injection: It is a dependency injection in which the framework injects the primitive and string-based values using setter metho
```