Spring Boot - Engineering Digest

..
In spring, instead of making/creating an object by ourselves, we route the call to SpringBoot **IOC [Inversion of Control].**

IOC contains all the objects in a Spring Boot Application or Project. It is like a container which contains all class. By **Component Scan. Keep only special class. @Component**

Application Context is nothing but a way to achieve IOC. These terms are used interchangeably.

Bean is nothing but an object. IOC classes contains/registers only those classes object which are called beans.

@Component is an Annotation that tells to **dump** the class into **IOC container. Class** is
Automatically registered as a **Spring Bean.**

@EnableAutoConfiguration helps in enabling auto configuring your SpringBoot project just by providing other details so as to what for you need to configure.

@Configuration annotation means the class is used to configure something. It is always used with @Bean config
@Bean annotation works or is **applied only on functions**,
not on class or etc
@Bean comes auto with **@RestController and @Component** annotations

@SpringBootApplication annotation does 3 works:
1. @Configuration
2. @EnableAutoConfiguration
3. @ComponentScan - *BasePackage [Add/include all Components class objects here only]*

@Autowired annotation does **Dependency Injection**. So the class annotated with Autowired is the class on which the other class depends inside which it is written.

Eg: **@RestController - It is used to serve all types HTTP requests - GET,PUT,PATCH,POST,DELETE**
public class Car{

```
        @Autowired
        private Dog dog;
// you can either use private Dog dog = new Dog();
        @GetMapping("/getinfo")
        public String ok(){
            return dog.fun();
        }
    }
```

- *Spring gives this Dog object which it contains in base package, so also called a bean.*
- We're using Autowired since, this would make the object once and just calls wherever it is used, otherwise we need to create instance every time. So, it keeps in IOC container as bean. DEPENDENCY INJECTION.

- Methods inside a Controller class should be public so that they can be accessed as well as invoked by Spring Framework and also the external HTTP requests.

@RequestMapping annotation is applied on complete class.
@RequestBody - It's like telling spring to take the data from the received request so that I can use it as a Java Object in my code.

```
@PostMapping("/abc")
public boolean createEntry(@RequestBody JournalEntry myEntry){}
```

@PathVariable annotation for dynamic paths.

```
3 usages
private Map<Long, JournalEntry> journalEntries = new HashMap<>();

no usages
@GetMapping
public List<JournalEntry> getAll() {
    return new ArrayList<>(journalEntries.values());
}

no usages
@PostMapping
public boolean createEntry(@RequestBody JournalEntry myEntry) {
    journalEntries.put(myEntry.getId(), myEntry);
    return true;
}

no usages
@GetMapping("id/{myId}")
public JournalEntry getJournalEntryById(@PathVariable Long myId){
    return journalEntries.get(myId);
}
```

MongoDB VS MySQL

Collections <-> Tables/Relations
Columns <-> Fields
Documents <-> Rows

Object-Relational Mapping (ORM)

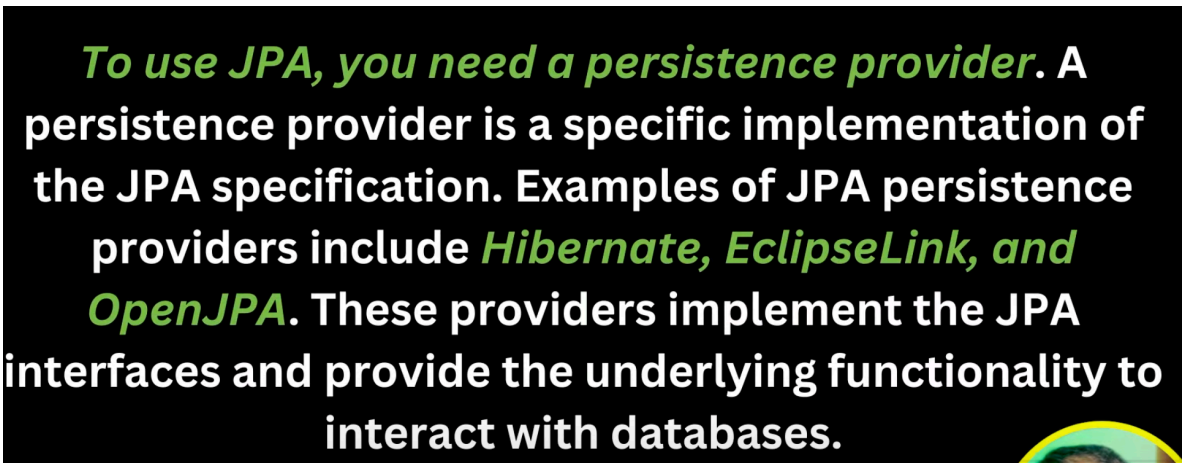
ORM is a technique used to map Java Obj to database tables.

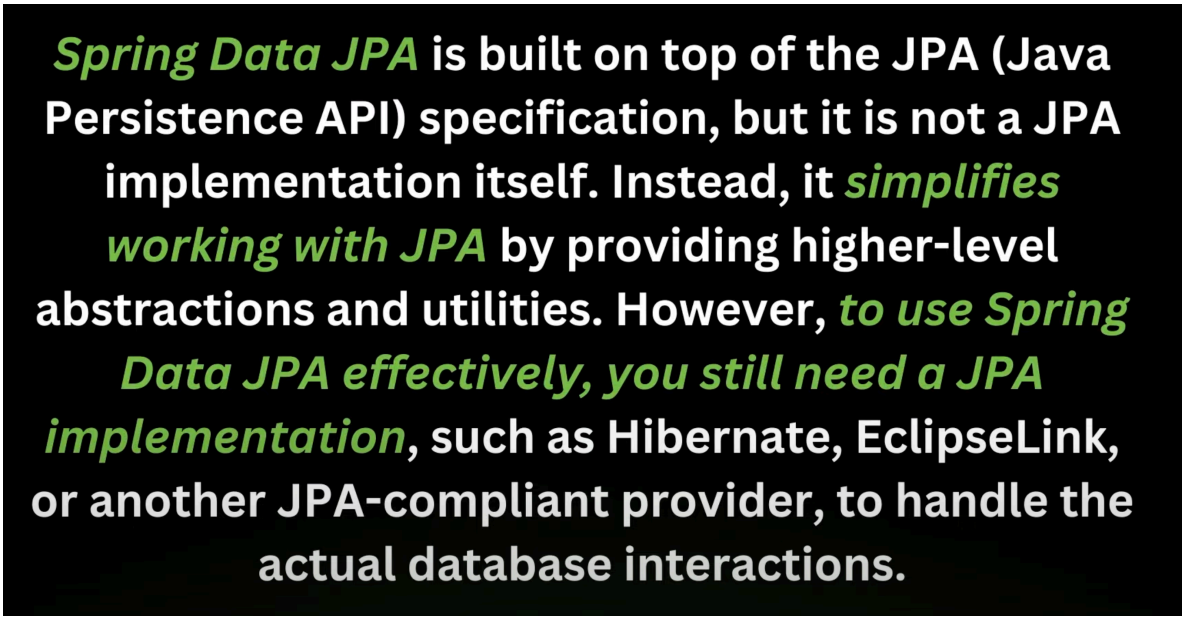
It allows developers to work with dbs using OOPs concepts, making it easier to interact with relational databases.

Java Persistance API (JPA)

Persistence - permanently store
API - set of rules

It is a way to achieve ORM, includes interfaces and annotations that you use in your Java classes, requires a persistence provider(ORM tools) for implementation.

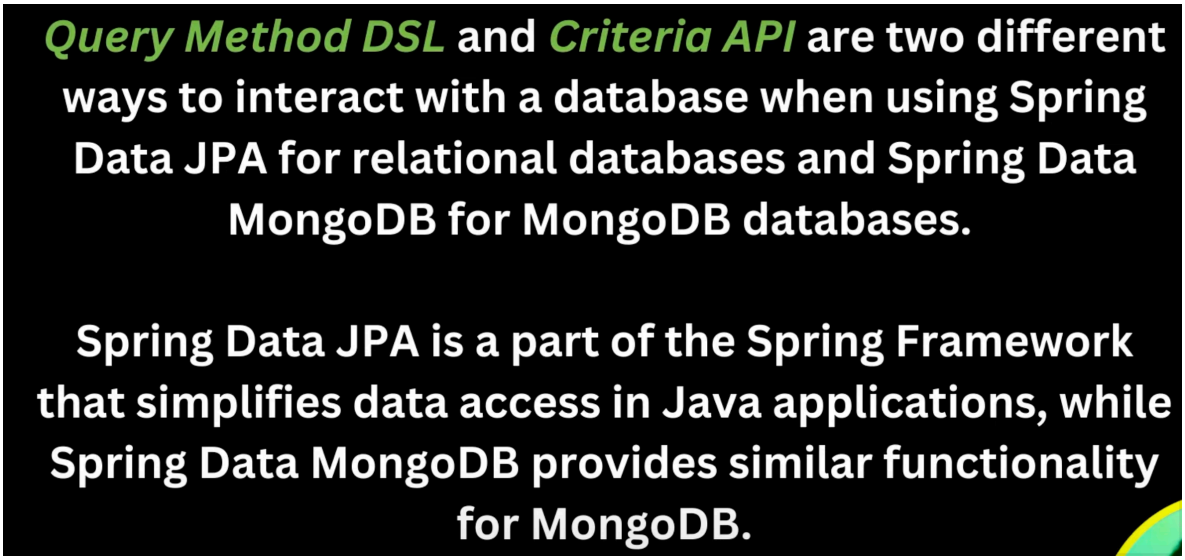




JPA is primarily designed to work with relational databases.

JPA is not used with MongoDB

To use MongoDB, **Spring Data MongoDB** is used.**[JPA]**



HTTP

Controller –> Service –> Repository

ResponseEntity class is part of Spring framework and is commonly used in Spring Boot applications to customise HTTP response.

It provides methods to set response status, headers and body. You can use it to return different types of data in ur controller methods, such as JSON, XML or even HTML. You can use generics too.

```
no usages
@GetMapping("id/{myId}")
public ResponseEntity<JournalEntry> getJournalEntryById(@PathVariable ObjectId myId) {
    Optional<JournalEntry> journalEntry = journalEntryService.findById(myId);
    if (journalEntry.isPresent()) {
        return new ResponseEntity<>(journalEntry.get(), HttpStatus.OK);
    }
    return new ResponseEntity<>(HttpStatus.NOT_FOUND);
}
```

```
@PostMapping
public ResponseEntity<JournalEntry> createEntry(@RequestBody JournalEntry myEntry) {
    try {
        myEntry.setDate(LocalDateTime.now());
        journalEntryService.saveEntry(myEntry);
        return new ResponseEntity<>(myEntry, HttpStatus.CREATED);
    } catch (Exception e) {
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
}
```

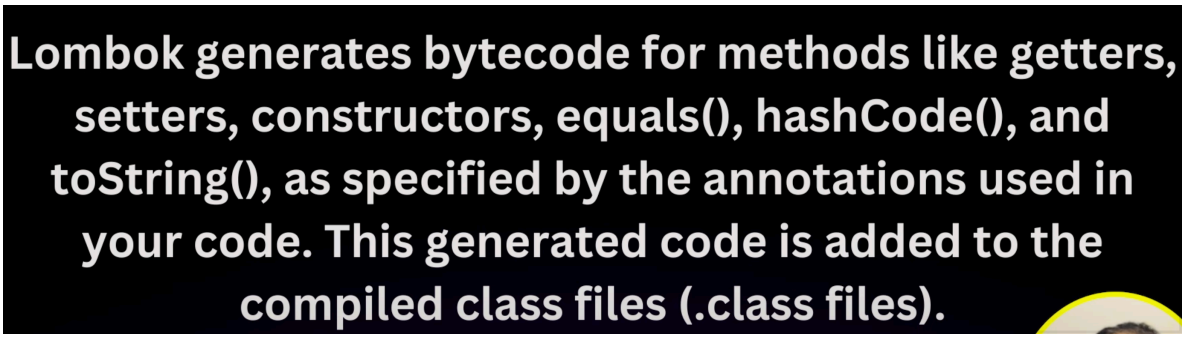
Lombok

Library in Java ecosystem, used in Spring Boot applications.

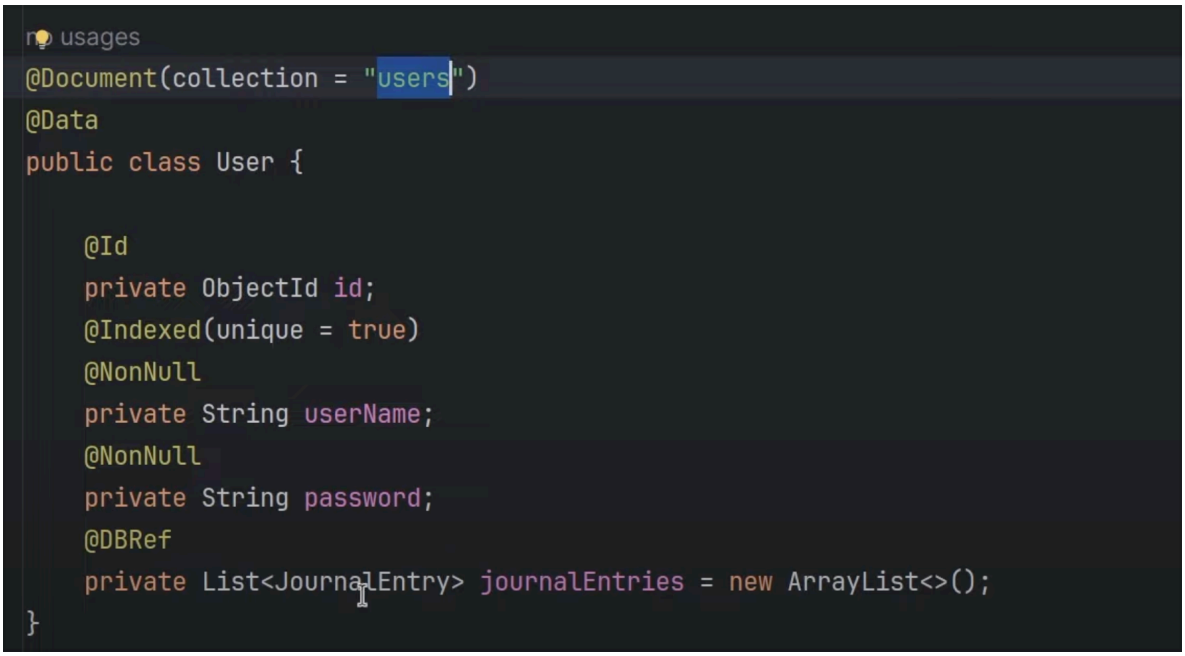
It aims to reduce boilerplate code that developers have to write such as getters, setters, constructors and more.

Lombok achieves the geeters and setters, etc without writing, since it generates code automatically during compilation based on annotations you add to your Java classes.

@Data annotation contains - **@Getter @Setter @RequiredArgsConstructor @ToString @EqualsAndHashCode**



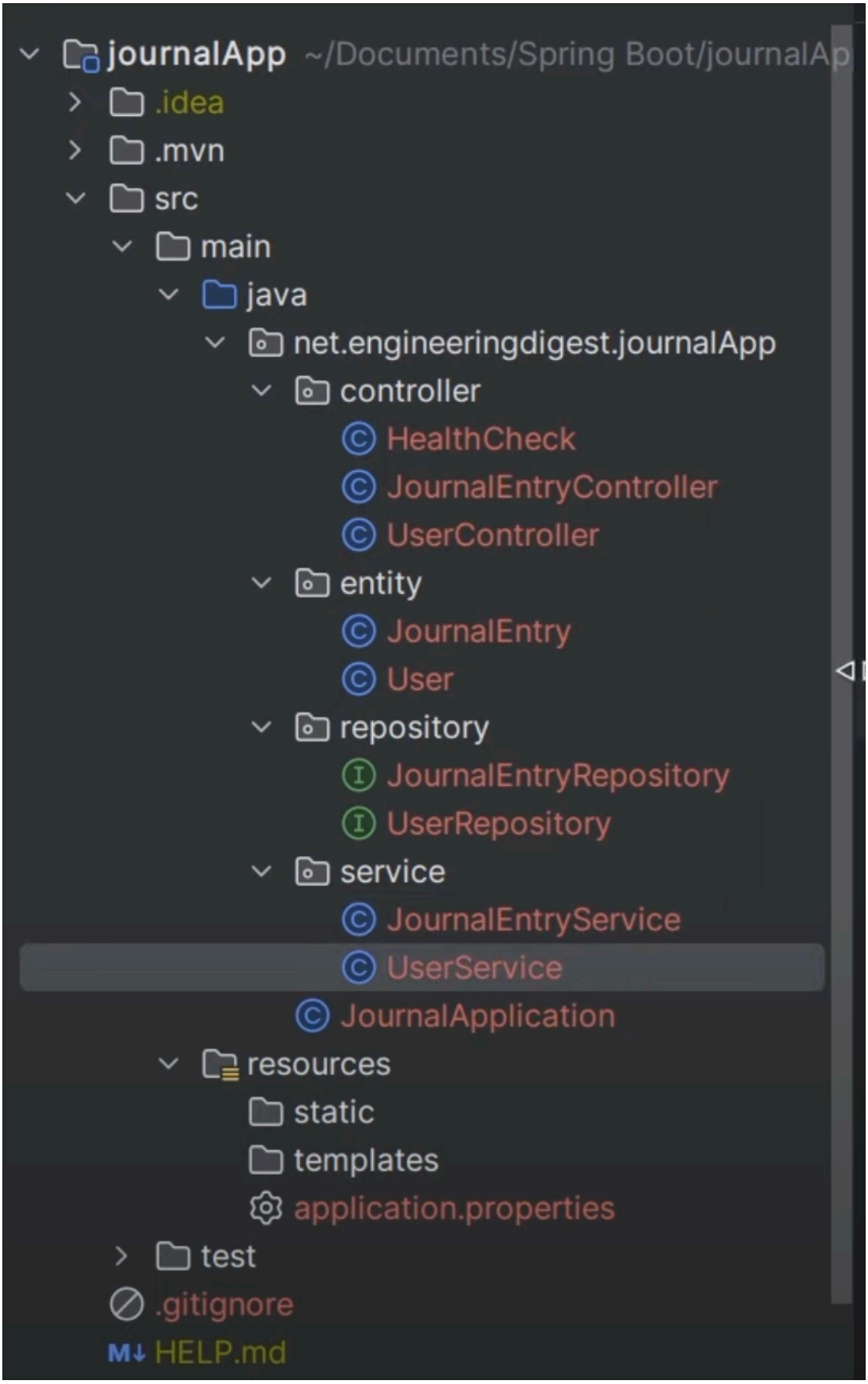
@DBRef



Now, the users and JournalEntry are connected by JournalEntry ObjectID.

f

Project Structure - [Spring Boot]

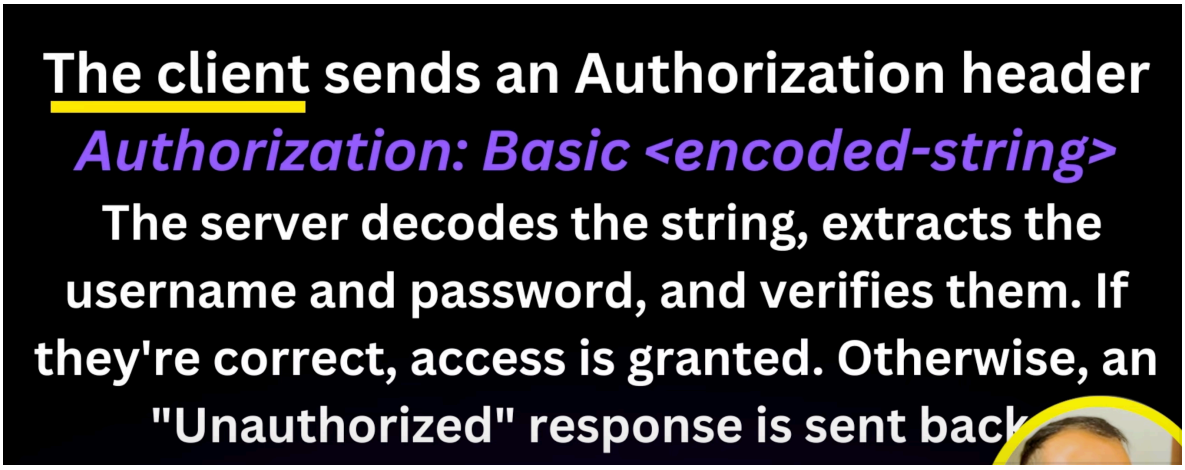


Spring Security

It is a powerful and highly customisable framework used in Spring Boot applications to handle authentication and authorisation.

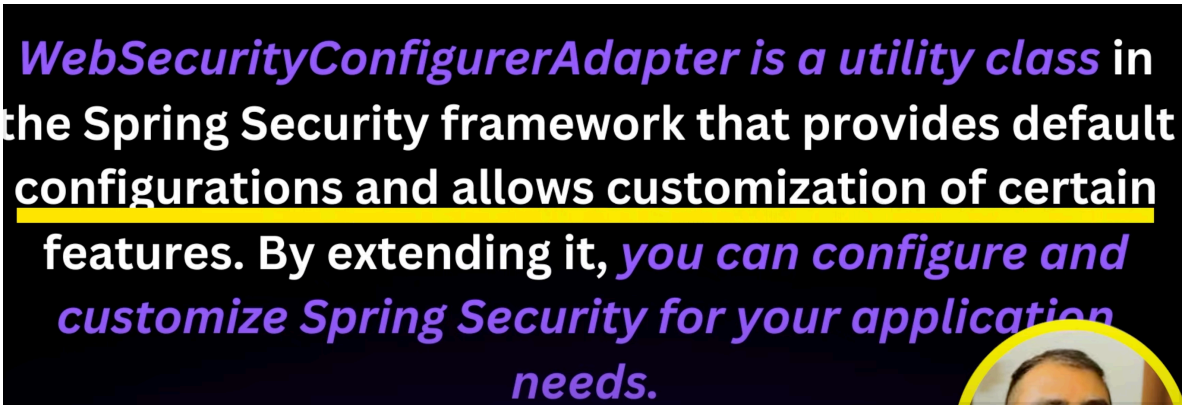
Authentication - The process of verifying a user's identity [eg. username, password]

Authorization - The process of granting or denying access to specified resource or actions based on user's roles and permissions.



Encoding Credentials are combined into string like **username:password** which is then encoded using **Base64**.

@EnableWebSecurity annotation signals Spring to enable its web security support. This is what makes your application secured. It's used in conjunction with **@Configuration.**



```java
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/hello").permitAll()
                .anyRequest().authenticated()
            .and()
            .formLogin();
    }
}
```

http.authorizeRequests(): This tells Spring Security to start authorizing the requests.

.antMatchers("/hello").permitAll(): This part specifies that HTTP requests matching the path /hello should be permitted (allowed) for all users, whether they are authenticated or not.

.anyRequest().authenticated(): This is a more general matcher that specifies any request (not already matched by previous matchers) should be authenticated, meaning users have to provide valid credentials to access these endpoints.

.and(): This is a method to join several configurations. It helps to continue the configuration from the root (HttpSecurity).

.formLogin(): This enables form-based authentication. By default, it will provide a form for the user to enter their username and password. If the user is not authenticated and they try to access a secured endpoint, they'll be redirected to the default login form.

When you log in with Spring Security, it manages your authentication across multiple requests, despite HTTP being stateless.

1. *Session Creation:* After successful authentication, an HTTP session is formed. Your authentication details are stored in this session.
2. *Session Cookie:* A JSESSIONID cookie is sent to your browser, which gets sent back with subsequent requests, helping the server recognize your session.
3. *SecurityContext:* Using the JSESSIONID, Spring Security fetches your authentication details for each request.
4. *Session Timeout:* Sessions have a limited life. If you're inactive past this limit, you're logged out.
5. *Logout:* When logging out, your session ends, and the related cookie is removed.
6. *Remember-Me:* Spring Security can remember you even after the session ends using a different persistent cookie (typically have a longer lifespan) .

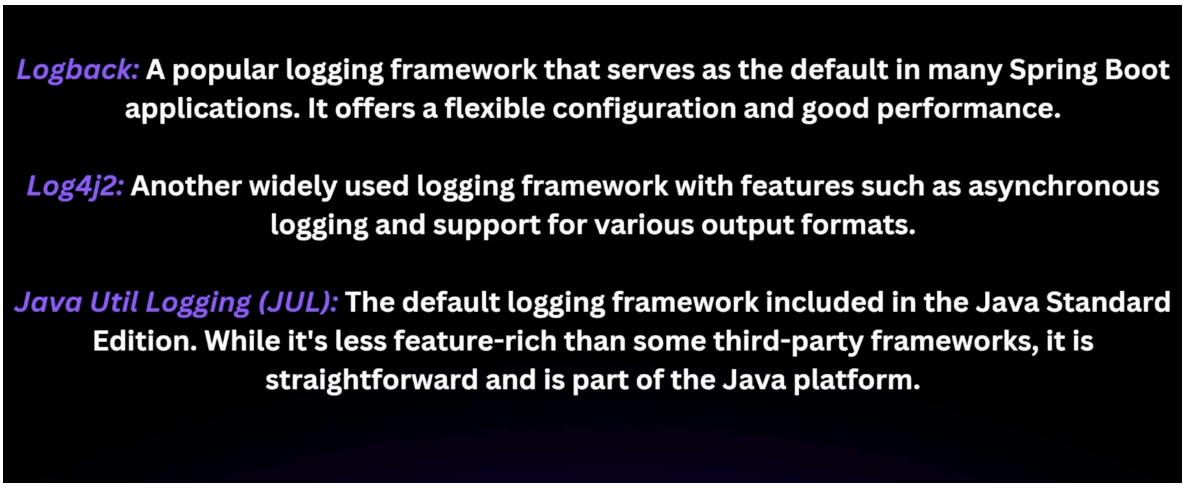
In essence, Spring Security leverages sessions and cookies, mainly JSESSIONID, to ensure you remai
authenticated across requests.

Priority

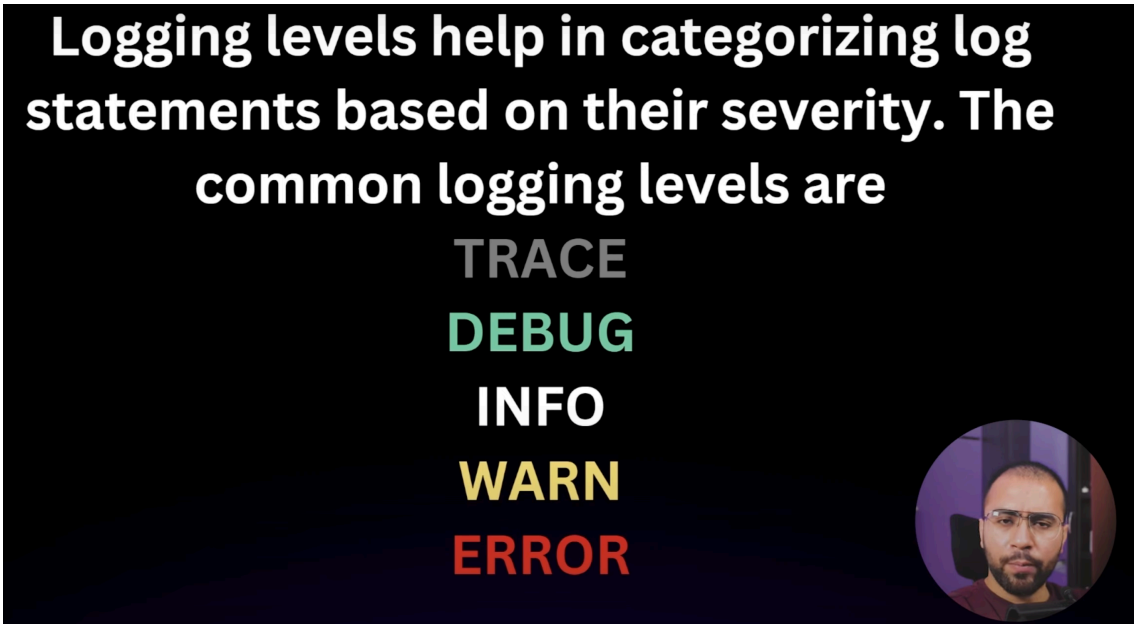
CMD > application.properties > application.yml

Logging

SpringBoot supports logging frameworks like Logback, Log4j2 and Java Util Logging(JUL).



SpringBoot uses **Logback** as default logging implementation.



SpringBoot provides annotations like **@Slf4j and @Log4j2**
That u can use to auto inject logger instances into ur classes.