Spring security Basic Authentication and Authorization

1. Add **@EnableWebSecurity** annotation in the main application
2. Add some end points in the controller class
3. Add Some properties in the application.properties file  
    a. **spring.security.user.name =PARIR034**

**b. spring.security.user.password =password this is only for local testing purpose**

this can be used , if u want for specific url ,or specific method security is required  
this default mechanism can be used  
  
or else . remove the above two properties from the application.properties file  
and use **config** file  
  
@Configuration  
public class SpringSecurityConfig {  
  
 // ✅ Define two in-memory users  
 @Bean  
 public UserDetailsService userDetailsService(PasswordEncoder encoder) {  
 UserDetails admin = User.*builder*()  
 .username("PARIR034")  
 .password(encoder.encode("password"))  
 .roles("ADMIN")  
 .build();  
  
 UserDetails user = User.*builder*()  
 .username("RAHUL")  
 .password(encoder.encode("password1"))  
 .roles("USER")  
 .build();  
  
 return new InMemoryUserDetailsManager(admin, user);  
 }  
  
 // ✅ Password encoder  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 // ✅ Security configuration  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .csrf(csrf -> csrf.disable()) // disable CSRF for REST  
 .authorizeHttpRequests(auth -> auth  
 // Role-based restrictions  
 .requestMatchers("/rest/auth/\*\*").hasRole("ADMIN") // only admin  
 .requestMatchers("/rest/noauth/\*\*").hasRole("USER") // only user  
 .anyRequest().authenticated()  
 )  
 .httpBasic(basic -> {}); // enable Basic Auth  
  
 return http.build();  
 }  
}

@Configuration  
public class SpringSecurityConfig {  
}  
Marks this class as a **configuration** class — meaning Spring will pick it up automatically and apply the beans you define here.  
  
so when ever u run the main application which is using @SpringBootApplication annotation, then this contains three annotations those are  
@SpringBootConfiguration // marks this as a configuration class (like @Configuration)

@EnableAutoConfiguration // enables Spring Boot’s auto-config

@ComponentScan // scans for components in the same package and subpackages  
  
**What @ComponentScan does**

@ComponentScan tells Spring to **look for classes** in the main package and its subpackages that are annotated with any of these:

* @Component
* @Service
* @Repository
* @Controller
* @RestController
* @Configuration

When it finds such classes, Spring automatically **registers them as beans** inside the ApplicationContext.

So, if it finds a class annotated with @Configuration,  
it assumes that the class **contains @Bean methods**, and Spring will **run those methods** to create and register the beans inside the ApplicationContext