A. By validating csrf token on postman
1. Create a getMapping request in the controller.
2. By providing authorization details on postman, send the request
3. You'll get a Token and that token is used to validate your requests
4. Add a header named "X-CSRF-TOKEN" as name and "token-value" in value

B. By using the Security Configuration
1. Create a class SecurityConfiguration
2. Use annotations @EnableWebSecurity and @Configuration
3. Create a @Bean of SecurityFilterChain and return its object
4. Firstly we have to disable CSRF token
5. By using authorizeHttpRequests(), we make every request to be authenticated
6. By using httpBasic(Customizer.withDefaults()), we configure the http basic request
7. If we want that everytime we reload the page the session also change to maintain more security we use function sessionCreationPolicy(SessionCreationPolicy.STATELESS);
8. at last by using build(), we return the SecurityFilterChain object
9. Now we can login by using two ways: static & dynamic

```
@Bean
   public SecurityFilterChain securityFilterChain(HttpSecurity http) throws  Exception{
      return http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(request -> request.anyRequest().authenticated())
            .httpBasic(Customizer.withDefaults())
            .sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .build();
   }
```

a. STATIC
1. Create a @Bean of UserDetailsService
2. We have to return InMemoryUserDetailsManager (Imp of UserDetailsService, which uses static username and password)
3. Also for this you can set passwordEncoder() which encodes your password
4. Set role and build() the object of it.
5. Return the new InMemoryUserDetails()

```
   @Bean
   public UserDetailsService userDetailsService(){
      UserDetails userDetails =
User.withDefaultPasswordEncoder().username("nittan").password("nittan").roles("user").build();
      return new InMemoryUserDetailsManager(userDetails);
   }
```

b. DYNAMIC (username and password fetched from the database and password stored in database is encoded password)
1. Comment out @Bean of UserDetailsService
2. Create a @Bean of AuthenticationProvider which used to authenticate our user and also we can fetch our users from the database
3. DaoAuthenticationProvider provides the methods -> setUserDetails() and setPasswordEncoder() so we return its object by creating the object of it by setting values in it.
4. @Autowire Properly your UserDetailsService and Create your own implementation of UserDetailsService
5. Create MyUserDetailsService which implements UserDetails and implement its method loadUserByUsername()
6. By using Jpa Dsl query find user by its name

7. If user found, then continue else return a exception that user is not found
8. Then return the object of new class (also known as Principalclass or implementation class of UserDetails)
9. Create a AdminPrincipal class or UserDetailsImplementation class which implements UserDetails
10. Implement its methods and create a object of your User and then set values according to it by creating a constructor of it.
11. Aslo by using grantedAuthority we can provide roles to our user like USER,ADMIN.
12. Create object of PasswordEncoder and pass it to the function where we setPasswordEncoder for it.
13. Give request on postman and now you are able to acheive authorisation and authentication.

A. SECURITY CONFIGURATION

```
@EnableWebSecurity
@Configuration
public class SecurityConfiguration {
    @Autowired
    private UserDetailsService userDetailsService;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws  Exception{
        return http
                .csrf(csrf -> csrf.disable())
                .authorizeHttpRequests(request -> request.anyRequest().authenticated())
                .httpBasic(Customizer.withDefaults())
                .sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .build();
    }

    // used when we provide users in our code
//    @Bean
//    public UserDetailsService userDetailsService(){
//        UserDetails userDetails =
User.withDefaultPasswordEncoder().username("nittan").password("nittan").roles("user").build();
//        return new InMemoryUserDetailsManager(userDetails);
//    }

    // used when we fetch users from the database
    @Bean
    public AuthenticationProvider authenticationProvider(){
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(userDetailsService);
        provider.setPasswordEncoder(new BCryptPasswordEncoder(12));
        return provider;
    }
}
```

B. MYUSERDETAILSSERVICE OR Implementation of USERDETAILSSERVICE

```
@Service
public class MyUserDetailsService implements UserDetailsService {
    @Autowired
    private AdminRepository adminRepository;
```

```
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        System.out.println("Entered in loadbyusername");
        Admin admin  = adminRepository.findByaName(username);
        System.out.println(admin.toString());
        if(admin == null){
            System.out.println("admin not found");
            throw new UsernameNotFoundException(" Admin 404");
    }
        System.out.println("Found");
        return new AdminPrincipal(admin);
    }
}
```

C. ADMINPRINCIPAL or Implementation of USERDETAILS

```
public class AdminPrincipal implements UserDetails {

    private Admin admin;

    public AdminPrincipal(Admin admin){
        this.admin = admin;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return Collections.singleton(new SimpleGrantedAuthority("ADMIN"));
    }

    @Override
    public String getPassword() {
        return admin.getAPassword();
    }

    @Override
    public String getUsername() {
        return admin.getAName();
    }

//implement other methods and use accordingly, if you dont want to use them just return true.
}
```

D. ADMINSERVICE

```
@Service
public class AdminService {
    @Autowired
    private AdminRepository adminRepository;

    private BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(12);

    public Admin register(AdminDto adminDto){
        Admin admin = new Admin();
        System.out.println("Before encoded: " + adminDto.getAdminName() + " " +
```

```
adminDto.getAdminPassword());
        admin.setAPassword(encoder.encode(adminDto.getAdminPassword()));
        System.out.println("after: " + admin.getAPassword());
        admin.setAName(adminDto.getAdminName());
        System.out.println(admin.getAPassword());
        return adminRepository.save(admin);
    }
```

E. REPOSITORY

```
@Repository
public interface AdminRepository extends JpaRepository<Admin,Integer> {

//     @Query("SELECT a FROM Admin a WHERE LOWER(a.aName) = LOWER(?1)") // Case-insensitive
search
//     Admin findByUsernameIgnoreCase(String username);

// Note write this function correctly, case-sensitive is very important
     Admin findByaName(String username);
}
```

F. CONTROLLER

```
@RestController
@RequestMapping("/admin")
public class AdminController {
    @Autowired
    private AdminService adminService;

    @PostMapping("/register")
    public Admin registerAdmin(@RequestBody AdminDto adminDto ){
        System.out.println("Entered Controller");
        return adminService.register(adminDto);
    }
```