"Spring security with mysql database"

→ Create a new Project :→ Day2SpringSecurity2

Dependencies :→
- Spring security
- mysql connector
- lombok
- spring web
- spring data jpa
- spring devtools

→ Go to Application.properties :→

```
spring.datasource.url = jdbc:mysql://localhost:3306/security   (database)
spring.datasource.username = root
spring.datasource.password = root
spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto = update
```

→ → Create a one entity class :→ Student

```
@Entity
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Student
{
    @Id
    private int id;
    private String name;
    private String password;
    private String role;
}
```

→ Create an Repository interface :→ StudentRepository

```
interface StudentRepository extends JpaRepository<Student, Integer>
{
    Student findByName (String username);
}
```

→ Create a one service class :→ StudentService

```
@Service
class StudentService
{
    @Autowired
    private StudentRepository stdRepo;

    @Autowired          ↗ इसका आगे Bean बना दिया StudentSecurity class में।
    private PasswordEncoder passwordEncoder;

    public Student save (Student std)
    {
        String pass = passwordEncoder.encode (std.getPassword());
                      ↗ Password Encode करके Database में set कर रहे हैं।
        std.setPassword (pass);
        stdRepo.save (std);
        return std;
    }

    public List<Student> getAll()
    {
        return stdRepo.findAll();
    }
}
```

→ Now Create a Controller class :- StudentController

```
@RestController
class StudentController
{
    @Autowired
    private StudentService stdService;

    @PostMapping("/save")
    public Student saveStudent(@RequestBody Student std)
    {
        return stdservice.save(std);
    }

    @GetMapping("/get")
    public List<Student> get()
    {
        return stdservice.getAll();
    }
}
```

→ Now Create a Security class :- StudentUserDtls

```
@Service
class StudentUserDtls implements UserDetailsService   // interface
{
    @Autowired
    private StudentRepository studentRepository;

    public UserDetails loadByUsername(String username) throws Exception
    // interface, return type, override
    {
```

→

```
Student std = studentRepository.findByName(username);
if(std == null)
{
    throw new NoSuchElementException("user not found");
}
```

[security ka pre-defined class]

collection which contain only one element

```
return new User(std.getName(), std.getPassword(), Collections.singleton(
        new SimpleGrantedAuthority(std.getRole())));
}
}
```

(Parser 3 Parameter in constructor)

↓ It Create authority object that represents student Role

→ Create a class :-> StudentSecurity

```
@Configuration
@EnableWebSecurity
class StudentSecurity
{
    @Autowired
    private StudentUserDtls stdUserDtls;

    @Bean
```

interface — Apply filters

```
public SecurityFilterChain func1(HttpSecurity http) throws Exception
{
    return http.csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(req -> req.requestMatchers("/save").permitAll())
            .anyRequest().authenticated()
            .httpBasic(Customizer.withDefaults())
            .build();
```

→ only for postman

```
}
```

→

```
@Bean
public AuthenticationProvider func2()
{
    DaoAuthenticationProvider d = new DaoAuthenticationProvider();
    d.setPasswordEncoder( func3() );
    d.setUserDetailsService( stdUserDtls );
    return d;
}

@Bean
public PasswordEncoder func3()
{
    return new BcryptPasswordEncoder();
}
}
```

interface

Implementing class

interface

implementing class

JWT

→ JWT stands for "JSON Web Token"

→ JWT contain 3 Parts :→

① Header :→ Header contain type of token and signing algorithm.

② Payload :→ contains the claims or the actual data. This can include user information, roles etc

③ signature :→ Created by combining the encoded header, encoded payload, and a secret key using the specified algorithm.

" All these 3 Parts are seperated by using dot (.).