6.Custom Login

Introduction to Custom Login

In Spring Security, custom login allows developers to define their logic for user authentication, bypassing the default Spring Security login page. This involves implementing an API that authenticates users using their credentials against the application's security configuration.

UserController Implementation

The UserController defines an endpoint for login (/login) that accepts user credentials and authenticates them using the AuthenticationManager.

Example:

```
@PostMapping("/login")
public String login(@RequestBody User user) {
    Authentication authentication = authenticationManager
            .authenticate(new UsernamePasswordAuthenticationToken(
                user.getUsername(), user.getPassword()));

    if (authentication.isAuthenticated()) {
        return "Success";
    } else {
        return "Login Failed";
    }
}
```

Explanation:

1. **@PostMapping("/login"):**
    - Defines a POST endpoint at /login for handling login requests.
2. **@RequestBody User user:**

    - Maps the incoming JSON request body to a User object containing username and password.
3. **AuthenticationManager:**
    - A central interface in Spring Security used to handle authentication requests.
    - It is responsible for delegating authentication logic to configured AuthenticationProviders.
4. **UsernamePasswordAuthenticationToken:**
    - A token implementation used to represent a user's authentication request containing a username and password.
    - Passed to the authenticate() method of AuthenticationManager.
5. **authentication.isAuthenticated():**
    - A method to check if the user has been successfully authenticated.
    - If true, the API returns "Success"; otherwise, "Login Failed".

SecurityConfig: Defining the AuthenticationManager Bean

To use the AuthenticationManager in the controller, it must be explicitly defined as a bean in the SecurityConfig class.

Code:

```
@Bean
public AuthenticationManager authenticationManager(AuthenticationConfiguration config)
        throws Exception {
    return config.getAuthenticationManager();
}
```

Explanation:

1. **@Bean:**
    - Marks the authenticationManager method as a Spring-managed bean.
2. **AuthenticationConfiguration:**
    - Provides access to the AuthenticationManager configured by Spring Security.
    - The getAuthenticationManager() method retrieves the default AuthenticationManager instance.
3. **Purpose:**
    - This configuration ensures the AuthenticationManager is available for injection into other components, such as the UserController.

Postman Test

The /login endpoint can be tested using Postman with the following JSON payload:

Request:

- **Endpoint:** POST http://localhost:8081/login
- **Headers:** Content-Type: application/json
- **Body:**

```
{
   "username": "navin",
   "password": "n@123"
}
```

Response:

- **Status:** 200 OK
- **Body:**

```
"Success"
```

Conclusion

This implementation demonstrates how to create a custom login API using AuthenticationManager. Key points include:

- Defining an authentication bean in the SecurityConfig.
- Using UsernamePasswordAuthenticationToken for handling authentication credentials.
- Testing the API using Postman to ensure it works as expected.