Disabling CSRF Token

Approaches to Configure Security

Your images show two approaches to configuring security and disabling CSRF:

1. **Lambda Way**:
    - Configuration uses functional programming with lambdas for concise and readable syntax.
2. **Imperative Way**:
    - Configuration uses traditional imperative-style method calls.

Both achieve the same result but differ in style.

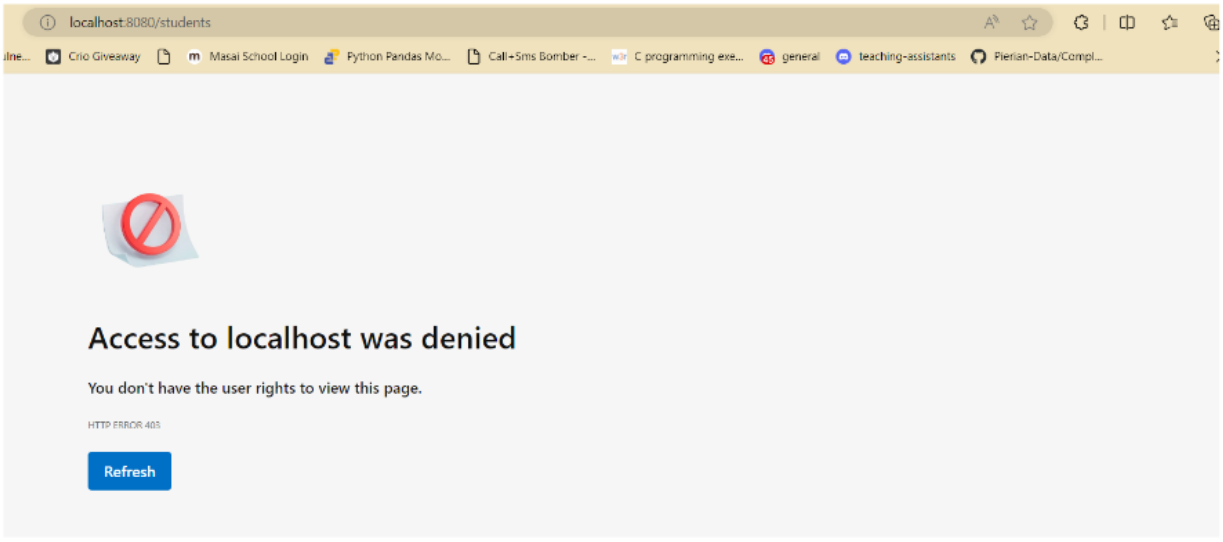
Lambda Way Configuration

Example:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

   @Bean
   public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
      http
         .csrf(csrf -> csrf.disable())  // Disabling CSRF
         .authorizeHttpRequests(request -> request.anyRequest().authenticated());
               //Allowing only authenticated requests

      return http.build();
   }
}
```



Behavior:

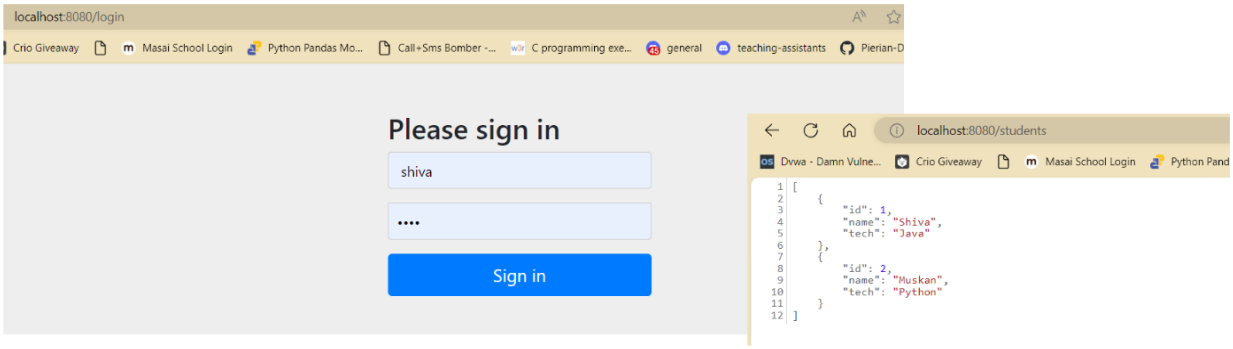
- **CSRF Disabled**:
  - No CSRF token is required for POST, PUT, or DELETE requests.
- **Authentication Required**:
  - All endpoints require authentication.
- **Output**:
  - If accessed without proper credentials, an "Access Denied" error is shown.

Example:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

	@Bean
	public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
		http.csrf(customizer->customizer.disable());
		http.authorizeHttpRequests(request->request.anyRequest().authenticated());
		http.formLogin(Customizer.withDefaults());

		return http.build();
	}
}
```



Behavior:

- **CSRF Disabled**:
  - Like the lambda approach, CSRF tokens are not needed.
- **Form-Based Authentication**:
  - Users are prompted with a login form for authentication.
- **Output**:
  - Upon accessing secured endpoints without credentials, users see a login page instead of "Access Denied."

Stateless Configuration (Sessionless)

Example:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

   @Bean
   public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
      http
         .csrf(csrf -> csrf.disable()) // Disable CSRF protection

         // Configure authentication
         .authorizeHttpRequests(request -> request.anyRequest().authenticated())

         // Add Basic Authentication
         .httpBasic()
```

```
        // Stateless sessions
        .and()
        .sessionManagement(session -> session
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS));

    return http.build();
  }
}
```

Behavior:

- **Stateless Sessions**:
  - The server does not maintain session information.
  - Common for REST APIs, where each request is authenticated independently.
- **Output**:
  - REST APIs become simpler and more compatible with token-based authentication systems like JWT.

Key Takeaways:

1. **CSRF Disabled**:
   - Removing CSRF makes APIs easier to use for stateless services.
   - Ensure other forms of authentication (e.g., token-based) are in place.
2. **Lambda vs Imperative**:
   - Both styles work; choose based on readability and team preferences.
3. **Stateless Configuration**:
   - Suitable for REST APIs.
   - Requires enabling SessionCreationPolicy.STATELESS to avoid session tracking.
4. **Form Login vs Basic Auth**:
   - **Form Login**: Preferred for user-facing web applications.
   - **Basic Auth**: Best for APIs or development/testing.