Technology Report- Spring Boot Security Layer

-What is Spring boot security?

Spring Boot security is an extension of Spring Security. This Spring Boot Security layer is a powerful authentication and authorization framework provided by the Spring Boot framework to provide security features for Java based web applications. This can be found in 'src/main/java/team/bham/security' path in the git repository. JDL generates most of the general required Java code for the securing our web app.

-Why will Spring Boot Security be useful to us?

Spring boot security is a vital component in our web application development, to ensure the projects are only accessed by authorised teams and persons. It provides a robust authentication and authorization mechanisms to secure the web app against various security threats. Spring Boot Security simplifies the process of securing our web application by offering configuration and customizable components and setup of security features such as authentication, authorization, and mechanisms like CSRF protection, session management. This reduces time spent for securing our Spring Boot web application with minimal configuration and boilerplate code. It also integrates seamlessly with other Spring Boot features, making it easy to implement security measures in our web app.

-Features and mechanisms

1.Supports various authentication mechanisms including form-based, HTTP Basic, OAuth, and JWT.

2.Authorization: Implements role-based access control (RBAC) and expression-based access control to enforce fine-grained authorization rules. The roles restrict access to tasks in the app.

3. CSRF Protection: Provides built-in protection against Cross-Site Request Forgery (CSRF) attacks.

4. Session Management: Offers session fixation protection, concurrent session control, and session timeout configuration.

5. Security Headers: Allows configuration of security headers to mitigate common web vulnerabilities such as XSS, CSRF, and clickjacking.

6. OAuth Integration: Simplifies OAuth 2.0 integration for enabling secure third-party authentication and authorization.

7. Spring Boot Security relies on annotations from the `org.springframework.security.config.annotation` package to configure security settings in your application. We can easily configure various aspects of security using annotations.eg. @Order, @AuthenticationPrinciple, @EnableWebSecurity, @Secured, @Configuration @PreAuthorize, @PostAuthorize etc.

-Steps to use Spring Boot Security for our web app

1.Dependency Configuration: Include the `spring-boot-starter-security` in the `pom.xml`.

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

2.Customize security configurations found in SecurityConfiguration.java, by extending `WebSecurityConfigurerAdapter` and override the configure (HttpSecurity http) method to define security rules to specify authentication and authorization rules.

3.User Authentication: Implements in memory authentication mechanisms such as form-based login, OAuth, or LDAP authentication. you can customize it in DomainUserDetailsService.java.

4.Role-Based Access Control: Define roles and permissions of persons and teams using annotations or bconfiguration to restrict access to specific resources.

5.Securing Endpoints: Secure RESTful endpoints or MVC controllers using method-level security annotations or HTTP security configurations. Use annotations @Secured or @PreAuthorize to secure individual methods in your controllers.

6.Customize Login and Logout process by providing your own login and logout pages and handlers in login.component.html.

-Potential Errors in Java Code and Fixes:

1. Misconfiguration of Security Dependencies can cause unexpected behaviour. Fix: Ensure that all Spring Security dependencies are compatible and properly configured in your `pom.xml` file.

2. Improper CSRF Configuration can cause security vulnerabilities. Fix: Enable CSRF protection by adding `.csrf().csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())` to configuration.

3. Certain endpoints that should be secured are accessible without authentication. Fix: add appropriate security configurations using `.authorizeRequests()` in the SecurityConfiguration class.

4. Incorrect Role-Based Access Control (RBAC): Error: Users might have incorrect roles assigned, allowing them access to unauthorized resources. Fix: Double-check role assignments and ensure that only authorized users have access to specific resources.

5. Weak password policies might allow users to set insecure passwords, leading to security vulnerabilities. Fix: Implement strong password policies using Spring Security's `PasswordEncoder` and consider integrating with libraries like OWASP's Java Encoder to enforce password requirements.

6. Exposed Sensitive Information: Error messages or stack traces might expose sensitive information. Fix: Customize error handling to provide generic error messages and prevent stack trace leakage.

7. Insecure Session Management: Error: Session fixation attacks or session hijacking vulnerabilities due to insecure session management. Fix: Configure secure session management settings such as setting `sessionFixation()` to `newSession()` and enabling secure cookies with `.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)`.

8. Missing HTTPS Configuration: Transmitting sensitive data over HTTP instead of HTTPS. Fix: Configure Spring Boot to run over HTTPS by providing appropriate SSL certificates and enabling HTTPS in your application properties or configuration.

9. Incorrect CORS Configuration: Missing or improper CORS configuration can lead to Cross-Origin Resource Sharing vulnerabilities. Fix: Configure CORS properly to restrict access only to trusted origins using `.cors()` in security configuration.

10. Failure to Handle Authentication Failures properly can lead to security issues. Fix: Implement proper authentication failure handling mechanisms, such as redirecting to a custom error page or returning appropriate error responses.

11. Ensure proper null checks are performed to prevent NullPointerExceptions.

12. Insecure Direct Object References (IDOR): Validate user input and enforce access controls to prevent unauthorized access to resources.

13. SQL Injection: Use parameterized queries or ORM frameworks like Hibernate to mitigate SQL injection vulnerabilities.

14. Session Management Issues: Configure secure session management settings to prevent session fixation and session hijacking attacks.

15. Always keep Spring Boot dependencies up-to-date, and test security configurations to ensure they are effective against potential threats by logging in and trying to access unauthorised tasks. Consider security testing tools and techniques such as penetration testing and code reviews to identify and address security vulnerabilities.

-Conclusion:

Spring Boot Security provides a robust framework for securing Java-based web applications against a wide range of security threats. By understanding its features, usage patterns, and potential pitfalls in Java code, developers can leverage Spring Boot Security to build secure and resilient applications.

-Sources

Spring Boot Reference Documentation: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#boot-features-security

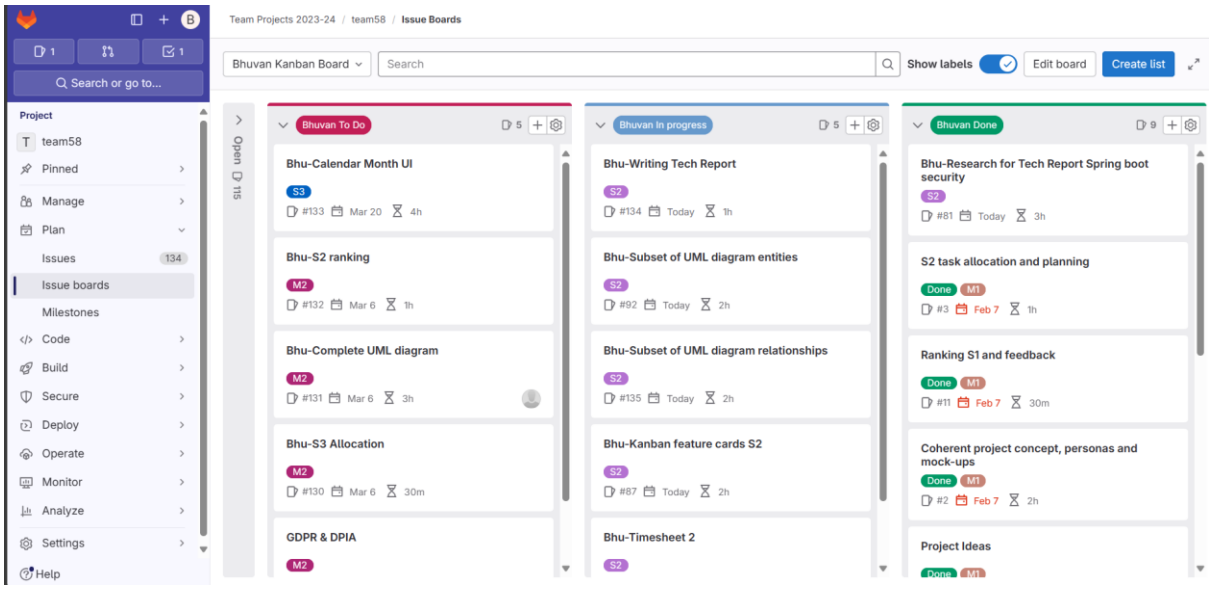
Spring Security Reference Documentation: https://docs.spring.io/spring-security/site/docs/current/reference/html5/

OWASP Top 10: https://owasp.org/www-project-top-ten/

Baeldung Spring Security Tutorials: https://www.baeldung.com/security-spring

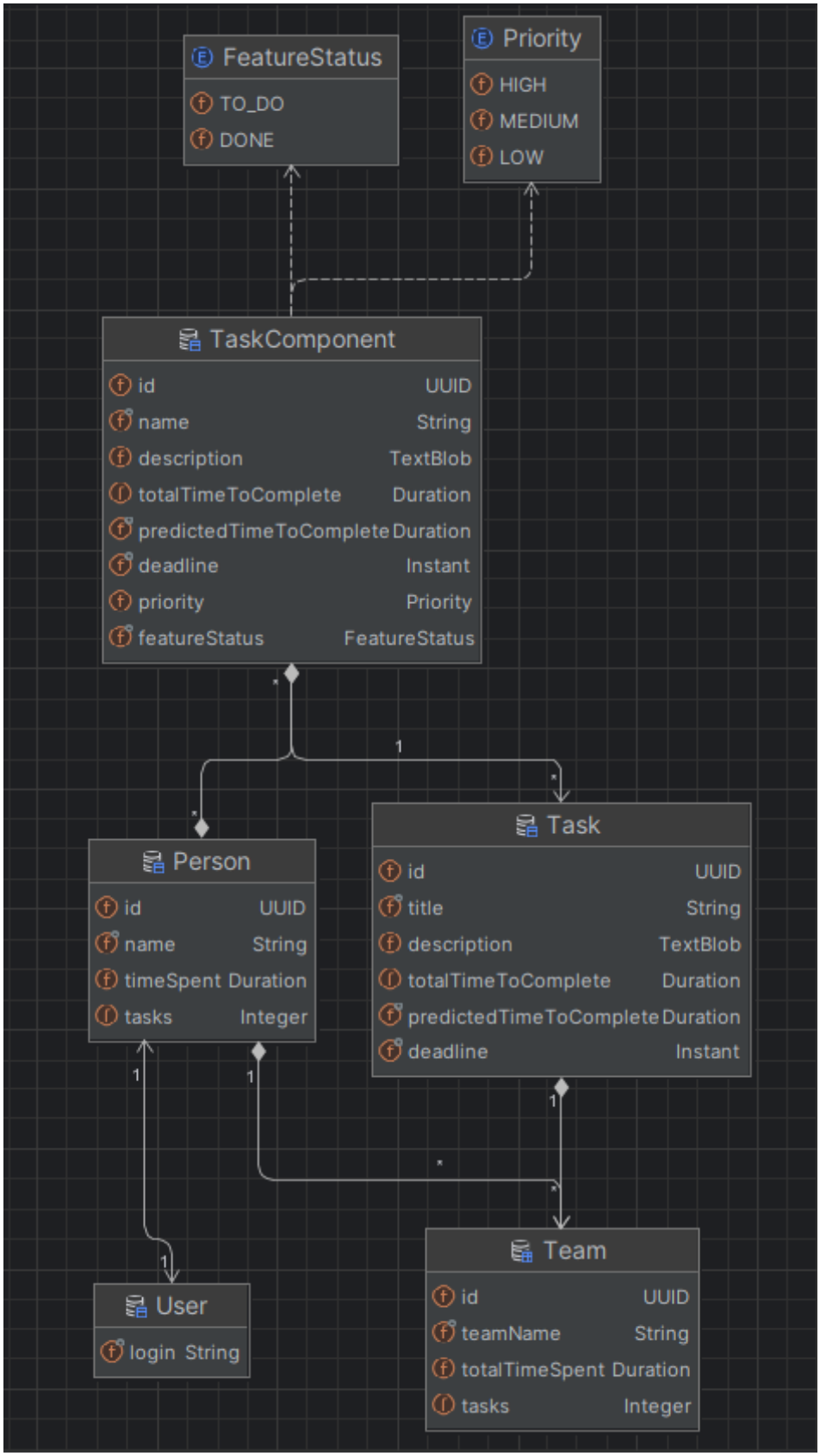
Java Code Errors and Fixes: https://www.oracle.com/java/technologies/javase/index.html

My Personal Kanban Board



Subset of the UML/JDL diagram for the calendar feature

1st Screenshots show a JDL-studio UML diagram of entities and relationships, in line with the UML diagrams of my teammates. 2nd and 3rd screenshots are the .jdl file for the diagram.



```
enum Priority{HIGH, MEDIUM, LOW                    ⚠ 4
}
enum FeatureStatus{TO_DO, DONE
}
entity Task{
    id UUID unique,
    title String required,
    description TextBlob,
    totalTimeToComplete Duration,
    predictedTimeToComplete Duration required,
    deadline Instant required
}
entity TaskComponent{
    id UUID unique,
    name String required,
    description TextBlob,
    totalTimeToComplete Duration
    predictedTimeToComplete Duration required,
    deadline Instant required,
    priority  Priority,
    featureStatus FeatureStatus required,
}
entity Team{
    id UUID unique,
    teamName String required,
    totalTimeSpent Duration,
    tasks Integer
}
entity Person{
    id UUID unique,
    name String required,
    timeSpent Duration,
    tasks Integer
}
relationship OneToMany{
    Team to Person,
    Task to TaskComponent,
    Team to Task,
}
relationship OneToOne{
    User to Person
}
relationship ManyToMany{
    Person to TaskComponent
}
paginate TaskComponent with infinite-scroll
```

Timesheet 2

Running from after the S1 submission, to the submission of S2

Team sheet Number/ID: team58-bxp267 TS2

Team member name: Bhuvan Praveen Kumar

Team representative (secrata: Lilliana Etheridge **Date from:** 30/01/2024

Team meeting sign off date: 26/02/2024 **Date until:** 26/02/2024

Task	Date	Start time	End time	Total Hours
Tutor meeting-S1 queries	31/01/2024	1:00 PM	1:30 PM	0:30
Team meeting-M1-ranking S1, mockup coherence	05/02/2024	1:00 PM	3:00 PM	2:00
Improve mock up coherence, Personal kanban cards	05/02/2024	6:00 PM	8:00 PM	2:00
Tutor meeting-M1 queries	06/02/2024	2:00 PM	2:30 PM	0:30
Team meeting-allocating tech research topics and jdl planning	12/02/2024	1:30 PM	2:30 PM	1:00
Research for Tech report	12/02/2024	6:00 PM	9:00 PM	3:00
Tutor meeting-asking about tech reports and plan confirmatio	13/02/2024	10:00 AM	10:30 AM	0:30
Edit personal Kanban feature cards	14/02/2024	2:00 PM	3:00 PM	1:00
subset of UML diagram entities	15/02/2024	1:00 PM	3:00 PM	2:00
Team meeting-JDL start and git push	19/02/2024	1:00 PM	2:30 PM	1:30
Tutor meeting- Fix JDL and discuss S2 work done	21/02/2024	12:00 PM	1:00 PM	1:00
Writing tech report	22/02/2024	1:00 PM	2:00 PM	1:00
subset of UML diagram relationships	23/02/2024	7:00 PM	9:00 PM	2:00
Edit personal kanban cards	25/02/2024	1:00 PM	2:00 PM	1:00
Tutor meeting- S2 queries	26/02/2024	12:30 PM	1:00 PM	0:30
Team meeting-S2 finalising	26/02/2024	1:00 PM	2:30 PM	1:30
Completion of timesheet 2	26/02/2024	6:00 PM	9:00 PM	3:00

Total Hours **24:00**