Sprint 6 Report

Code Review Report

This report summarizes the findings from a static code analysis conducted on our project using three tools: Checkstyle, SpotBugs, and SonarQube. The objective was to assess code quality, maintainability, and adherence to best practices. Below, we present key metrics, highlight significant issues, and provide suggestions for improvement.

CheckStyle Report

Configuration: Frontend module analyzed using Google Checks

- Errors: 0
- Warnings: 1,920

Note: issues are similar for backend so only frontend is presented here.

Common issues identified

- Documentation and formatting
  - Missing a Javadoc comment
  - Line is longer than 100 characters
  - Extra separation in import group

```
import utils.ImageUtil;

import java.io.File;
```

- Import structure
  - Wrong lexicographical order for import
  - Using the '.*' form of import should be avoided

```
import javafx.scene.control.*;
```

- Code style violations
  - Abbreviation in name must contain no more than '1' consecutive capital letters

```
@FXML private VBox monVBox, tueVBox, wedVBox, thuVBox, friVBox, satVBox, sunVBox;
```

  - WhitespaceAround: '{' is not preceded with whitespace

```
private void updateEventList(){  1 usage  teemvat
    cachedEvents = EventController.getAllEvents();
    updateListView(cachedEvents);
    updateCalendarView(cachedEvents);
}
```

- 'if' construct must use '{}'s

```
if (event == null) return;
```

- '}' should be alone on a line

```
private String getSelectedLocationId() {  2 usages  teemvat +1
    Location selectedLocation = locationComboBox.getSelectionModel().getSelectedItem();
    return (selectedLocation != null) ? selectedLocation.getId() : null;    }
```

- Empty blocks should have no spaces

```
private SessionManager() { }
```

- Statement and expression style
  - Each variable declaration must be in its own statement

```
@FXML private Label eventNameLabel, organizerLabel, dateLabel, startTimeLabel, endTime
```

  - '&&' and '+' should be on a new line

```
return (searchQuery.isEmpty() || event.getTitle().toLowerCase().contains(searchQuery.toLowerCase())) &&
        (date.isEmpty() || event.getDate().equals(date)) &&
        (eventType == null || Arrays.asList(event.getCategories()).contains(eventType)) &&
        (location == null || LocationController.getLocationById(event.getLocationId()).getName().equals(location)) &&
        (organizer == null || UserController.getUser(Integer.parseInt(event.getOrganizerId())).getUsername().equals(organizer));
```

SpotBugs Report

Metrics

1887 lines of code analyzed, in 48 classes, in 5 packages.

Metric	Total	Density*
High Priority Warnings	17	9.01
Medium Priority Warnings	27	14.31
Total Warnings	**44**	**23.32**

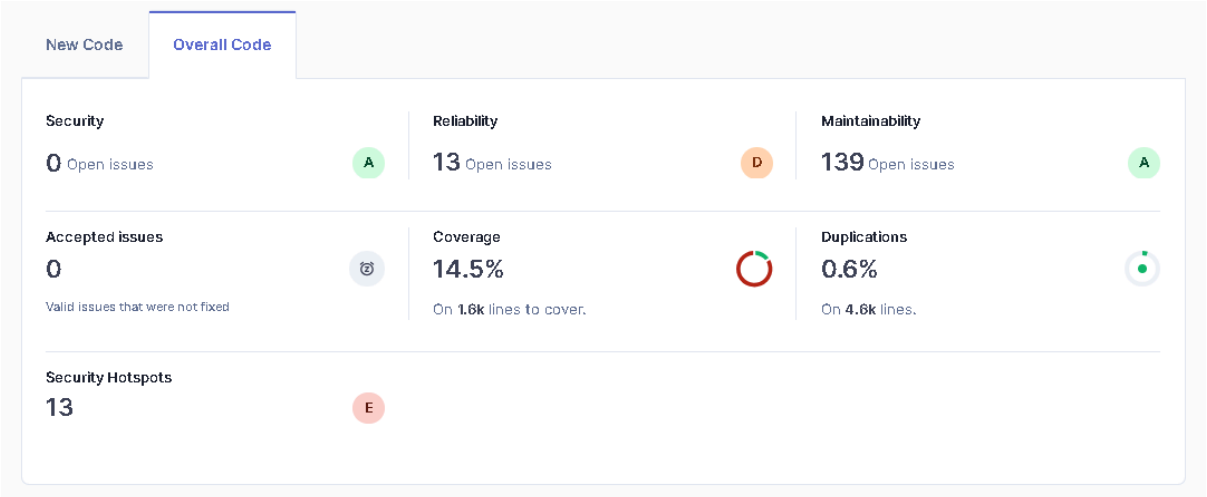
(Defects per Thousand lines of non-commenting source statements)*

Summary

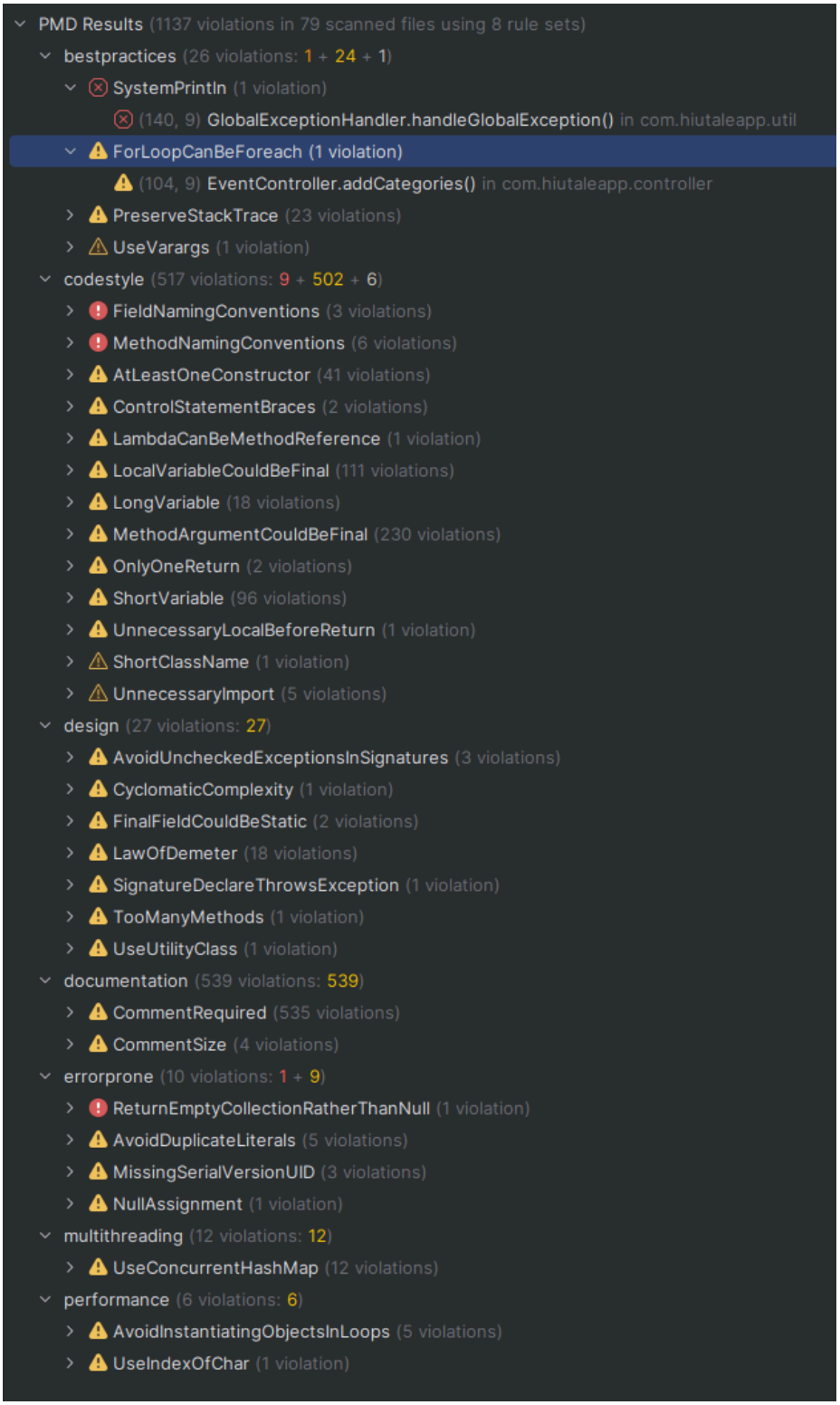
Warning Type	Number
Bad practice Warnings	4
Correctness Warnings	1
Experimental Warnings	1
Internationalization Warnings	17
Malicious code vulnerability Warnings	13
Performance Warnings	4
Dodgy code Warnings	4
Total	**44**

SpotBugs Report

SonarQube Report



PMD report (backend)



Backend PMD report

Common issues identified:

- Issues such as not preserving the stack trace, using PascalCase to describe constants instead of all caps and some issues regarding function names not using camelCase properly.
- PMD complains about things such as Entities not having constructor and not using final variables (they do not need one in Spring Boot).
- Complaints about function having multiple exit points, however I'd argue that having things such as guard clauses can make code cleaner instead of having to use large if clauses.
- Complains about some functions having great cyclomatic complexity as result of having to conform to Spring Boot entity relationships.
- Creating objects in a loop
- Missing Javadoc documentation
- Unable to find duplicate code and does not complain of too long methods

Summary

Most problems are unavoidable with Spring Boot as repository is written to conform to Spring Boot style.

```java
@PostMapping("/create")  Tom
public EventDTO createEvent(@RequestBody EventForm eventForm) {
    try {
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();

        Event event = new Event();
        User user = new User();
        Location location = new Location();

        user.setUserId(Long.parseLong(auth.getName()));
        location.setLocationId(eventForm.getLocationId());

        event.setTitle(eventForm.getTitle());
        event.setDescription(eventForm.getDescription());
        event.setDescription(eventForm.getDescription());
        event.setCapacity(eventForm.getCapacity());
        event.setStart(eventForm.getStart());
        event.setEnd(eventForm.getEnd());
        event.setStatus(eventForm.getStatus());
        event.setPrice(eventForm.getPrice());

        // If there's time left fix this
        List<EventCategory> c = addCategories(eventForm.getCategories(), event);

        event.setEventCategories(c);
        event.setOrganizer(user);
        event.setLocation(location);
        return eventService.createEvent(event);
    } catch (CannotCreateTransactionException e) {
        throw new DatabaseConnectionException("Could not connect to the database");
    } catch (DataIntegrityViolationException e) {
        throw new DataViolationException("Could not create event due to foreign key error");
    }
}
```

```java
private List<EventCategory> addCategories(List<Long> categories, Event event) {  1 usage  Tom
    if (categories == null) return null;
    List<EventCategory> c = new ArrayList<>();
    for (int i = 0; i < categories.size(); i++) {
        Long categoryNumber = categories.get(i);
        EventCategory eventCategory = new EventCategory();

        Category category = new Category();
        CategoryFA categoryfa = new CategoryFA();
        CategoryFI categoryfi = new CategoryFI();
        CategoryJA categoryja = new CategoryJA();
        category.setCategoryId(categoryNumber);
        categoryfa.setCategoryId(categoryNumber);
        categoryfi.setCategoryId(categoryNumber);
        categoryja.setCategoryId(categoryNumber);

        eventCategory.setCategory(category);
        eventCategory.setCategoryfi(categoryfi);
        eventCategory.setCategoryfa(categoryfa);
        eventCategory.setCategoryja(categoryja);
        eventCategory.setEvent(event);
        c.add(eventCategory);
    }
    return c;
}
```

Examples of high cyclomatic complexity. Spring Boot necessitates complex nesting of different entities.

SonarQube Analysis

SonarQube provided deeper insights into code quality, particularly in complexity and duplication.

Key metrics:

Metric	**Value**
Cyclomatic complexity	18
Cognitive complexity	19
Duplicated code density	6.3%
Duplicated lines	302
Duplicated blocks	50
Duplicated files	8

Acceptance test plan

Test ID	Test name	Purpose	Steps	Acceptance
01	Registration	A new user can create an account.	1. Get registration page 2. Enter email, username and password 3. Click register	A new account is created
02	Log in	A registered user can log in to their account	1. Get login page 2. Enter email and password 3. Click login	Correct credentials log the user in
03	Filter events	User can filter events	1. Log in 2. Select filtering criteria 3. View events that match the select criteria	Filter shows correct events in UI
04	Event details	User can view details of selected event	1. Log in 2. Click on an event 3. Get event details	Correct info is shown
05	Register for an event	User can register to an event	1. Log in 2. Open chosen event 3. Click “Attend”	Attendance is saved
06	Mark event as favorite	User can mark an event as favorite	1. Log in 2. Open chosen event 3. Click “Favorite”	Favorite is saved
07	Event creation	User can create events to the system	1. Log in 2. Get event creation menu 3. Fill event info 4. Submit	Event appears in the event view