**O.W.C.A**

Jacques Klooster - u23571561
Aidan McKenzie - u23545080
Reece Jordaan - u23547104
Richard Kruse - u23538172
Luke Gouws - u23668882

## Table of Contents

## 1. Introduction

Automated assessment systems have become indispensable in computer science education, streamlining the grading process and providing students with rapid feedback. At the University of Pretoria's Computer Science Department, our existing tool, FitchFork, has served this role admirably for several years. However, with the rapid evolution of both programming paradigms and academic integrity challenges, there's a clear need to rebuild and expand its capabilities. Our capstone project aims to not only replicate all of FitchFork's current features but also to integrate cutting-edge research from GATLAM, a new AI-powered engine for in-depth code evaluation and personalised feedback.

The motivation behind this rebuild is twofold. First, by working on "our nemesis," we get hands-on experience in software engineering, AI, and formal methods, all while directly contributing to an essential departmental tool. Second, by incorporating advanced features like plagiarism and AI-generation detection, we're ensuring that the system remains robust, fair, and scalable in the face of ever-more sophisticated student submissions.

## 2. User Stories / User Characteristics

Student User Stories

1) View Assignments
   As a student,
   I want to view and download the assignment instructions so that I know what I need to do.

   Acceptance Criteria:
   - The student can select an assignment from a list in a module
   - The system allows file downloads

2) Submit Assignments
   As a student,
   I want to submit my assignment for a module,
   so that my code can be automatically marked.

   Acceptance Criteria:
   - The student can select an assignment from a list in a module
   - The system allows file uploads
   - The submission is stored and queued for auto-marking

- The student is notified once marking is complete

3) View Feedback and Marks
   As a student,
   I want to see my marks and feedback after submitting my assignment.
   so that I can understand how I performed and improve.

   Acceptance Criteria:
   - The system displays marks out of a total score
   - Feedback includes test results, terminal output and comments
   - Students can view previous submissions and their respective feedback

Tutor User Stories

1) View Student Submissions
   As a tutor,
   I want to view student submissions for an assignment,
   so that I can assist students with their issues.

   Acceptance Criteria:
   - Tutors can access a list of student submissions for modules they are assigned to
   - Each submission shows the student's name, timestamp, and status
   - Tutors cannot modify the student's code or marks

2) View Terminal Output
   As a tutor,
   I want to see the terminal output of a student's submission,
   so that I can better understand what went wrong and provide help.

   Acceptance Criteria:
   - Tutors can view the console output (stdout/stderr) from a student's code execution
   - Output includes test results and any runtime errors
   - Tutors cannot rerun or modify the execution

Lecturer User Stores

1) Manage Assignments

As a lecturer,
I want to create an assignment in a module,
so that students can submit their work for marking.

Acceptance Criteria:

- The lecturer can choose the module and create a new assignment
- Students linked to the module can see the new assignment

2) Submit Memo Code
   As a lecturer,
   I want to upload my memo code solution,
   So that the system can automatically generate a testing framework for student marking.

   Acceptance Criteria:

   - The lecturer uploads one working solution file

3) Choose Testing Algorithm
   As a lecturer,
   I want to choose between RNG, Code Coverage or GATLAM algorithm,
   so that the appropriate testing framework is generated for student submissions.

   Acceptance Criteria:

   - The lecturer is presented with the three algorithm options when creating an assignment
   - The system applies the selected algorithm to generate the testing framework

4) Detect Plagiarism
   As a lecturer,
   I want to view plagiarism reports for assignments,
   so that I can identify and address cheating among students.

   Acceptance Criteria:

   - A plagiarism report is available for each assignment after student submissions
   - The report highlights matching submissions and a similarity score

Admin User Stories

1) Access All Tools
   As an admin,
   I want access to all features available to students, tutors and lecturers,
   so that I can manage and oversee the entire system.

   Acceptance Criteria:
   - Admins can view and manage all modules, users, assignments, and submissions

2) Manage User Permissions
   As an admin,
   I want to change the roles and permissions of any user,
   so that I can ensure correct access levels are assigned.

   Acceptance Criteria:
   - Admins can view a list of all users with their current roles
   - Admins can change a user's role

3) Manage Modules
   As an admin,
   I want to create modules,
   so that I can assign new lecturers to them

   Acceptance Criteria:
   - Admins can create modules and assign lecturers to them

## 3. Functional Requirements

**FR1: User Authentication and Roles**

- **FR1.1**: System shall support user authentication for Admin, Tutor, and Student roles.
- **FR1.2**: System shall restrict access to features based on user roles.

**FR2: Module Management**

- **FR2.1**: Admin/Tutor shall be able to create new modules.
- **FR2.2**: Admin/Tutor shall be able to edit module details.
- **FR2.3**: Admin/Tutor shall be able to delete modules.

**FR3: Assignment Management**

- **FR3.1**: Admin/Tutor shall be able to create assignments for a module.
- **FR3.2**: Admin/Tutor shall be able to edit assignment details.
- **FR3.3**: Admin/Tutor shall be able to delete assignments.

**FR4: Marking Script Management**

- **FR4.1**: Admin/Tutor shall be able to create marking scripts using:
    - **FR4.1.1**: GATLAM
    - **FR4.1.2**: Random Number Generator
    - **FR4.1.3**: Coverage-based algorithm
- **FR4.2**: Admin/Tutor shall be able to delete marking scripts.
- **FR4.3**: Admin/Tutor shall be able to edit marking scripts.
- **FR4.4**: Admin shall be able to upload custom marking logic (not use interpreter).
- **FR4.5**: Interpreter shall translate marking script into executable code.

**FR5: Grammar Input**

- **FR5.1**: Admin shall be able to define terminals.
- **FR5.2**: Admin shall be able to define nonterminals.

**FR6: Input Data Management**

- **FR6.1**: Admin/Tutor shall be able to create input data:
    - **FR6.1.1**: Manually
    - **FR6.1.2**: Automatically (supporting seeding)
- **FR6.2**: Admin/Tutor shall be able to delete input data.
- **FR6.3**: Admin/Tutor shall be able to edit input data.

**FR7: Code Submission**

- **FR7.1**: Students shall be able to upload their code files.
- **FR7.2**: System shall hash uploaded student files using md5sum or quantum-safe hash.

**FR8: Code Viewer and Runner**

- **FR8.1**: System shall allow viewing code without downloading.
- **FR8.2**: System shall support syntax highlighting for code.
- **FR8.3**: System shall allow running code without downloading.
- **FR8.4**: System shall show output and stacktrace of execution.

**FR9: Execution Environment**

- **FR9.1**: Student submissions shall be run in containerized environments.
- **FR9.2**: Student output shall be matched to marker output outside of the container.

**FR10: Plagiarism Detection**

- **FR10.1**: System shall support plagiarism detection per assignment.
- **FR10.2**: System shall allow modular swapping of plagiarism algorithms (e.g., MOSS).
- **FR10.3**: System shall compare ASTs before invoking MOSS.
- **FR10.4**: System shall match code using GitHub Search API.

**FR11: AI Assistance**

- **FR11.1**: System shall provide AI-generated summaries of exceptions.
- **FR11.2**: System shall provide AI-generated summaries of incorrect outputs.

**FR12: Gamification and Progression**

- **FR12.1**: System shall support achievements and other gamified elements.
- **FR12.2**: System shall support unlocking tasks by completing previous tasks.

**FR13: Grading System**

- **FR13.1**: System shall calculate grades per assignment.
- **FR13.2**: System shall allow different grade weights per task.
- **FR13.3**: System shall display grades to students.
- **FR13.4**: System shall support time and space complexity analysis.

**FR14: Submission Rules**

- **FR14.1**: Admin shall be able to configure:
    - **FR14.1.1**: Submission deadlines (date and time)
    - **FR14.1.2**: Late submission policy
    - **FR14.1.3**: Submission count limit (including infinite)

**FR15: Reporting and Statistics**

- **FR15.1**: System shall provide live statistics per assignment.

- **FR15.2**: Statistics shall be available as downloadable reports.
- **FR15.3**: Statistics shall be displayed in graph form.

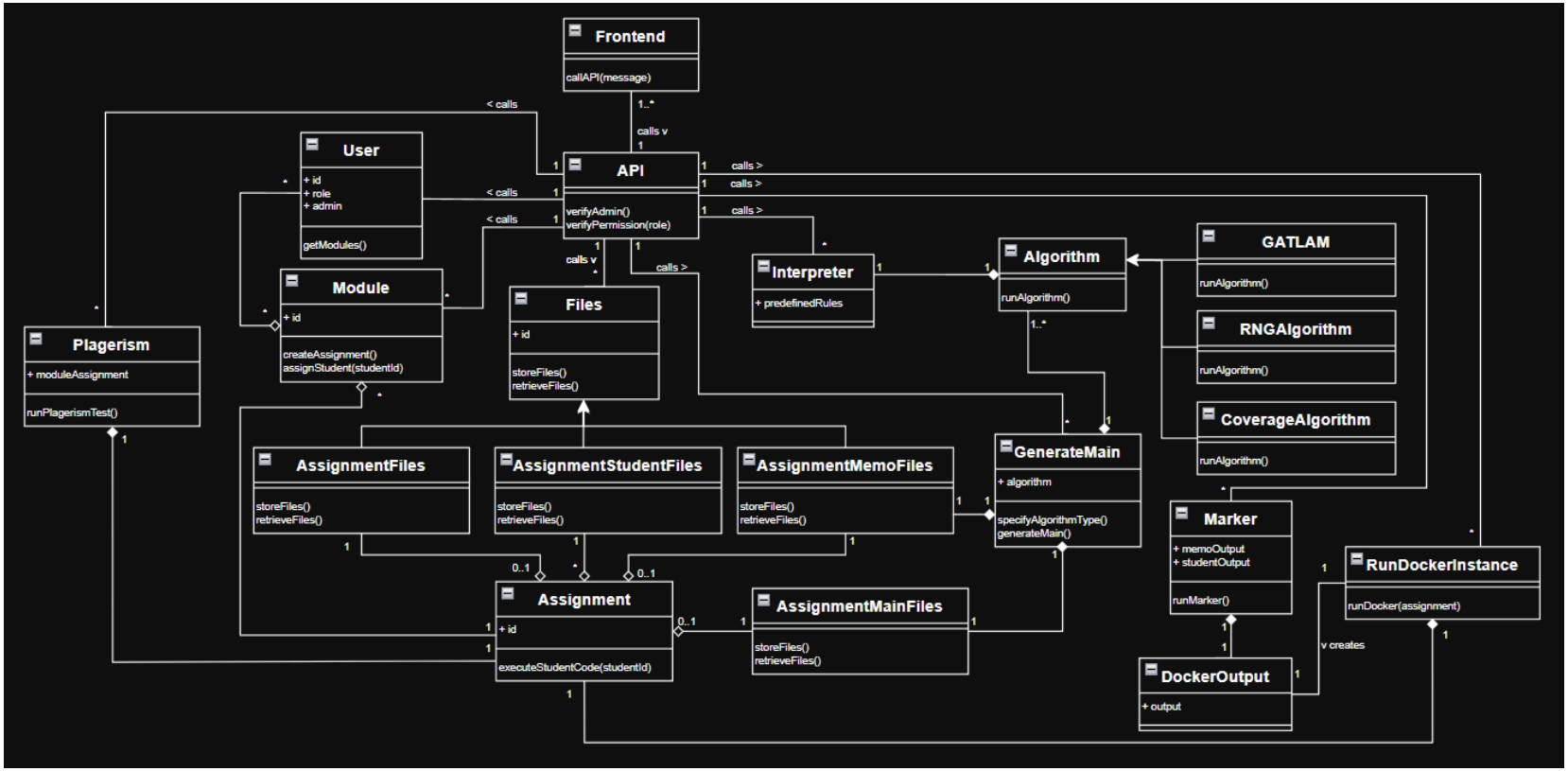
**FR16: Security**

- **FR16.1**: System shall restrict student access to memo content.
- **FR16.2**: System shall isolate containers to prevent memo leakage.

**FR17: Support System**

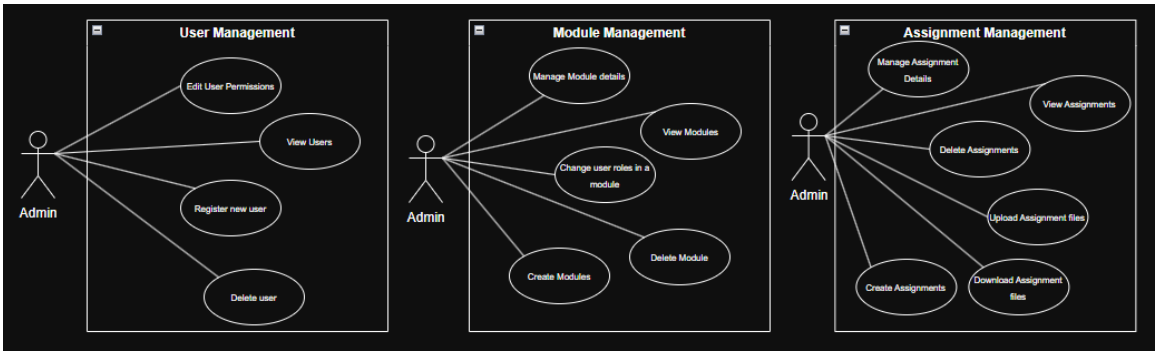
- **FR17.1**: System shall have a ticketing system (Feature Flag enabled).

## 4. Domain Model

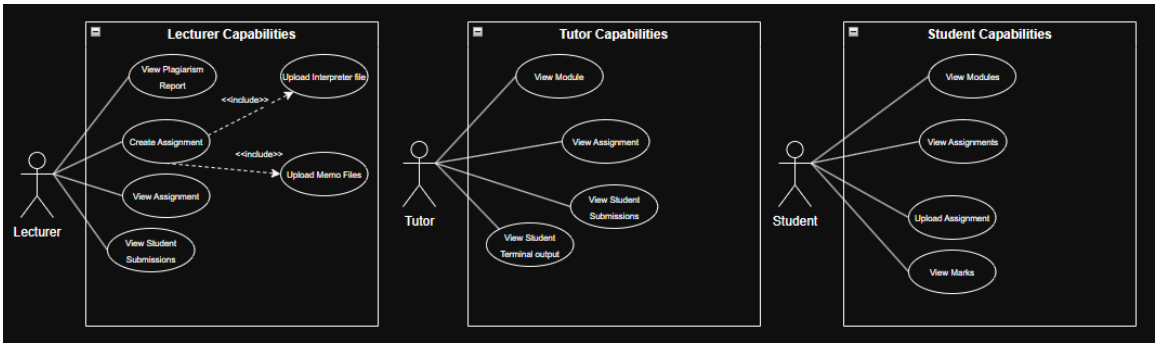


## 5. Use Cases

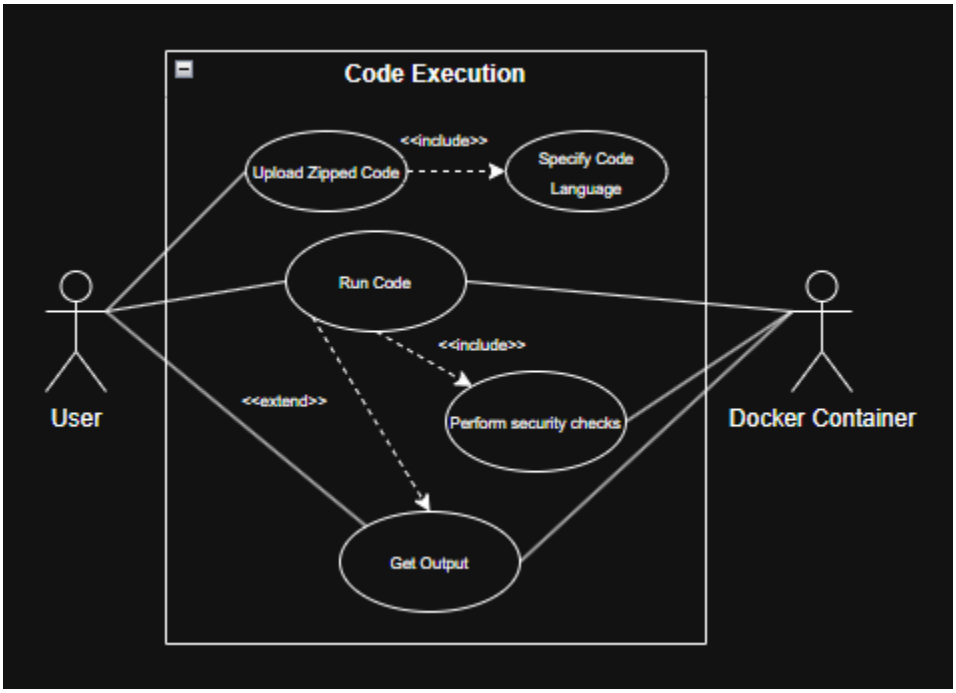
### Admins




### Lecturer/Tutor/Student



**Code Execution**




## 6. Architectural Requirements

### 6.1 Quality Requirements

**Performance**

- QR-PERF-1 — Average code submissions ≤ 5 s
- QR-PERF-2 — Plagiarism check ≤ 60 s for ≤ 100 submissions

**Scalability**

- QR-SCAL-1 — Handle 1 000 concurrent submissions
- QR-SCAL-2 — DB & containers must scale horizontally

**Availability**

- QR-AVAIL-1 — 99.9 % uptime during term

**Usability**

- QR-USAB-1 — WCAG 2.1 AA

- QR-USAB-2 — Tasks ≤ 3 clicks

**Security**

- QR-SEC-1 — Sandboxed, non-privileged containers
- QR-SEC-2 — TLS everywhere
- QR-SEC-3 — RBAC on files & data

**Maintainability**

- QR-MAINT-1 — The codebase must be modular so that plagiarism, and grading engines can be swapped in or out without touching the core API.

**Compatibility**

- QR-COMP-1—The web app must work on the latest two major releases of Chrome
- QR-COMP-2—Mobile layout is optional but must not break when the feature flag is turned on.

## 6.2 Architectural Patterns

### MVC Architecture

- **View**
    - **Implementation**: React/Next UI (client-side)
      **Rationale**: Handles presentation, routing, form validation, state-management; keeps browser concerns isolated.
- **Controller**
    - **Implementation**: Axum REST layer (/api/* endpoints)
      **Rationale**: Receives HTTP requests, performs authentication/authorisation, orchestrates calls to services, and returns serialised ApiResponse<T>.

- **Model**
    - **Implementation**: Domain + persistence (Rust services + SQLx repositories)
    - **Rationale**: Encapsulates business rules (e.g., assignment deadlines), integrates with Postgres, exposes domain objects (Assignment, Module, etc.).

## 6.3 Design Patterns

**Factory Method**

- **Where we use it**: We use the factory method for creating seeding data for our database and creating new entities in the system.
- **How it benefits us**: This helps us to easily seed the database with a bunch of random data to assist with development

**Proxy**

- **Where we use it**: Axum middleware layer sits in front of every route, verifies the JWT, and only forwards the request if it's valid. There are different types of checks for example you might need to be admin to access certain routes or you might need to be a lecturer for some module to access their crud routes.
- **How it benefits us**: Keeps auth separate from business code; you can stack more middleware (rate-limit, logging) without changing handlers.

**Singleton**

- **Where we use it**: A single, globally shared database pool.
- **How it benefits us**: Re-uses connections efficiently and prevents "too many open connections" errors. This makes a central access point for our database.

## 6.4 Constraints

**Business / Schedule**

- Fixed due date and demos
- No new cloud spend; must stay on existing university hardware

**Skills / Process**

- Team are Rust beginners
- Fully-automated CI/CD; rollbacks < 5 min

**Legal / POPIA**

- All PI data stored **in-country**
- **Encrypt at rest** (DB, backups, file store) & **in transit** (TLS 1.3)
  Logs retained 5 years; breach-response plan compliant with §22

**Technology Stack**

- Backend: **Rust + Axum**
- DB: **PostgreSQL 15** on-prem
- API: **HTTP/JSON REST**

- Use only LTS crates pinned in Cargo.lock

**Deployment / Ops**

- Run as OCI containers on **university Kubernetes ≥ 1.30**
- Per-pod limit: **1 vCPU / 1 GiB RAM**; cold-start < 2 s

**Security**

- All user code executes in **non-privileged, sandboxed containers**
- RBAC enforces file-level access
  **Scalability / Availability**
- Scale **horizontally only** (Kubernetes' HPA + DB replicas)
- Target **99.9 % uptime** during term

**Quality Floors**

- Execution feedback ≤ 5 s; plagiarism ≤ 60 s/100 files
- WCAG 2.1 AA compliance; supports latest 2 releases of Chrome and Firefox

## 7. Technology Requirements

**Hardware Requirements**

- 15 GB of total storage availability
- 16 GB of RAM

**Software Requirements**

- Operating System
  - Linux
  - MacOS
  - Windows
- Programming Languages
  - Rust
  - Any Programming Language that gets marked by FitchFork
- Frameworks
  - React
- Database
  - SQLite

**Network requirements**

- API connectivity