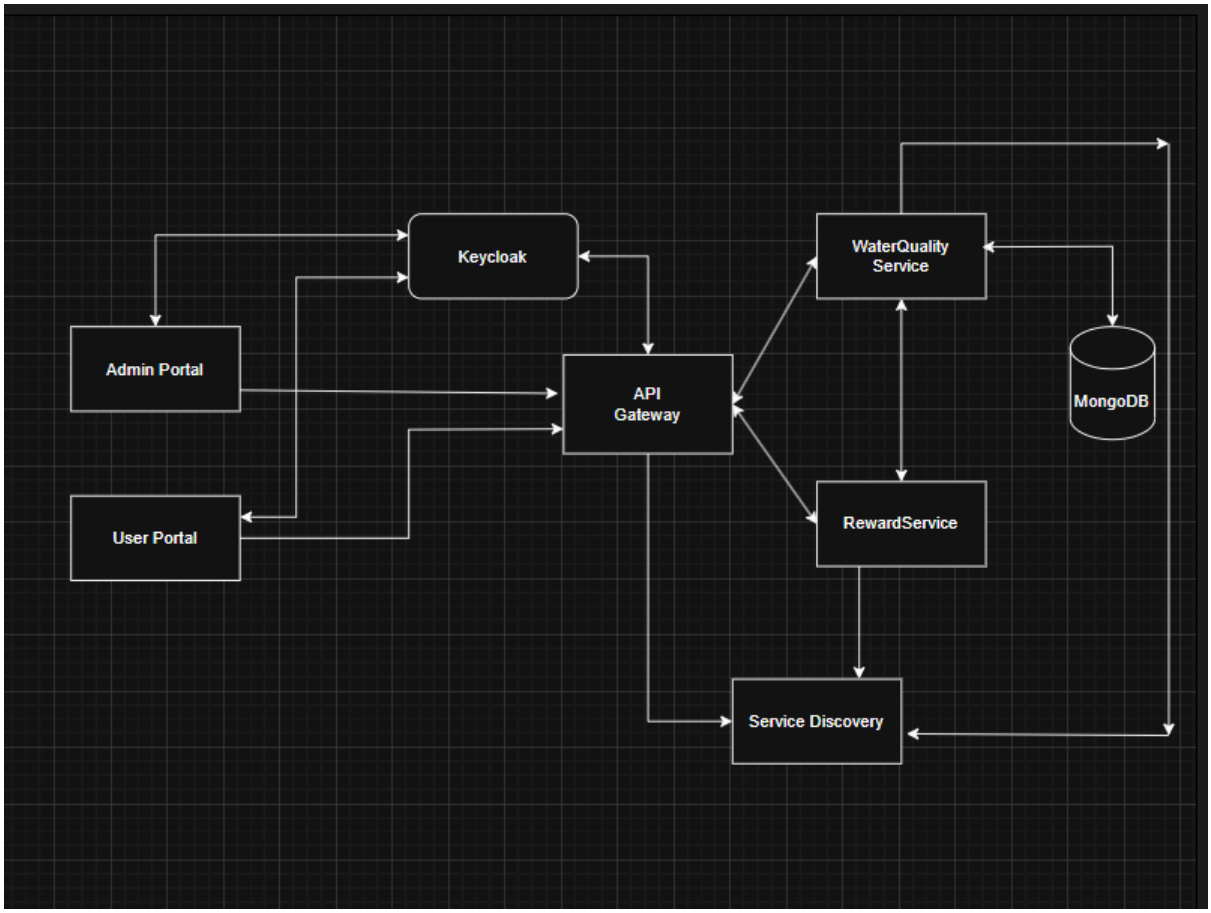
Technical Report

Architecture Diagram:



The developed solution follows a **microservices-based architecture** designed to ensure scalability, modularity, and clear separation of concerns. The system consists of multiple independent services communicating via RESTful APIs.

At the core of the architecture is the **API Gateway**, which acts as a single-entry point for all client requests. Both the **User Portal** and **Admin Portal**, implemented using **React**, interact exclusively with the API Gateway. This approach abstracts internal service details from the clients and centralises routing and security enforcement.

The **Crowdsourced Data Service** is responsible for receiving, validating, and persisting citizen-submitted water quality observations and user data. It exposes REST APIs to submit observations, retrieve historical reviews, and manage citizen profiles.

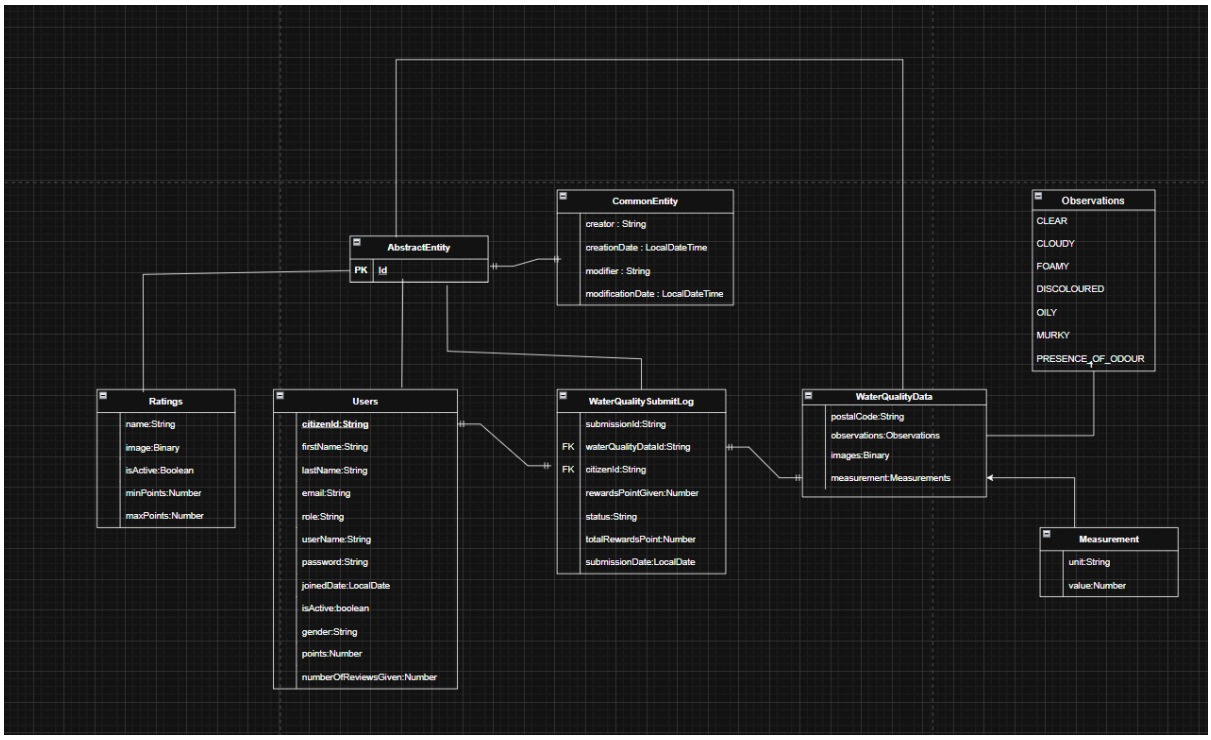
The **Reward Service** operates independently and processes validated submissions to calculate reward points and assign ratings or badges to citizens. This service consumes data from the Crowdsourced Data Service via API calls routed through the API Gateway.

A **Service Discovery** component enables dynamic registration and discovery of microservices, supporting loose coupling and scalability. **Keycloak** is integrated to provide authentication and role-

based authorisation, ensuring that sensitive endpoints (such as administrative operations) are protected.

Each microservice maintains its own responsibility boundary, improving fault isolation and enabling independent deployment.

Database Schema:



The database schema is designed to support permanent storage of crowdsourced water quality data while maintaining data integrity and traceability.

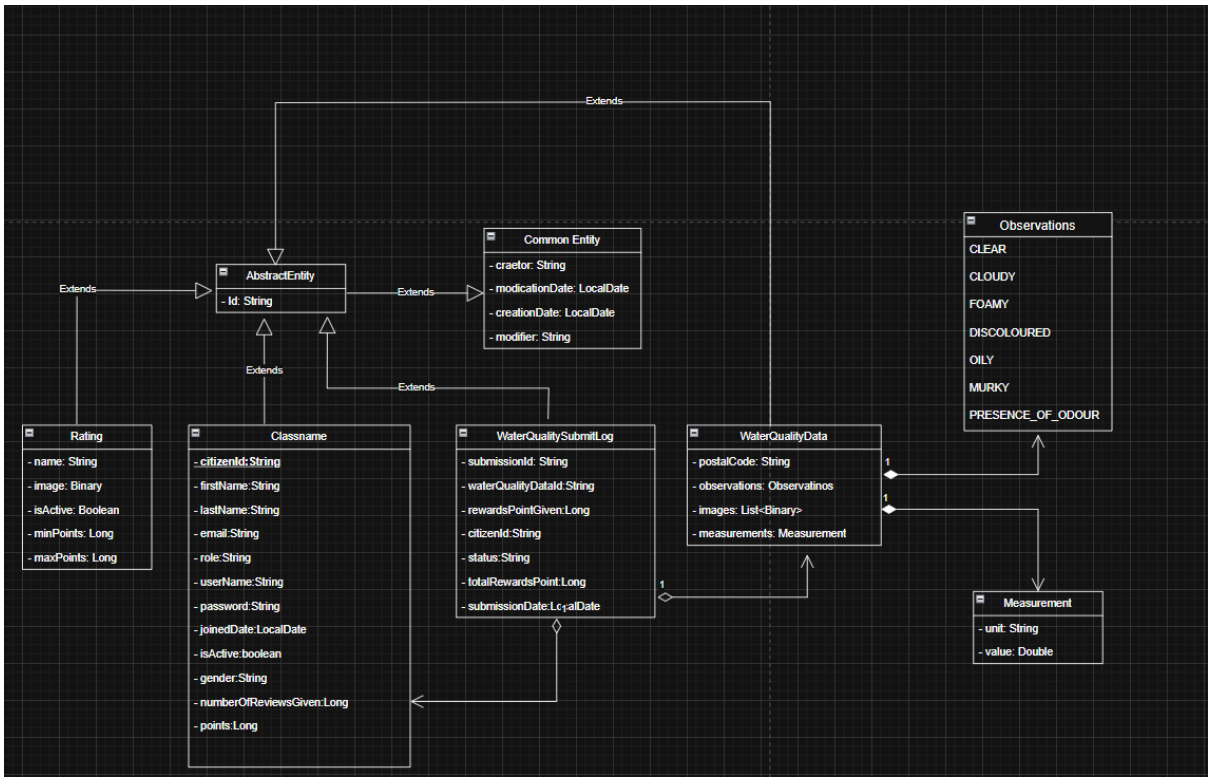
The **User** entity stores citizen-related information such as citizen ID, username, activation status, and metadata required for submission tracking. Each user is uniquely identified by a citizen ID.

The **WaterQualityData** entity stores individual submissions, including a generated UUID, submission timestamp, postal code, measurement units and values, observations, and optional image references. A one-to-many relationship exists between **User** and **WaterQualityData**, allowing a single citizen to submit multiple observations.

Supporting entities store structured representations of measurements and observations, ensuring flexibility and extensibility. Relationships are defined using foreign keys to maintain referential integrity.

This schema enables efficient querying of historical submissions, user activity, and aggregation of data for visualisation and reward calculation.

Class Diagram:



The class design follows **object-oriented principles**, promoting modularity and maintainability. Controller classes such as WaterQualityController, UserController, RewardController, and RatingController expose REST endpoints and delegate business logic to corresponding service classes.

Service classes encapsulate validation rules, reward calculation logic, and transactional operations. Repository classes abstract data access, ensuring separation between persistence logic and business logic.

Data Transfer Objects (DTOs) are used extensively to decouple internal domain models from external API contracts. Exception classes, such as CrowdDataSourceException, provide consistent error handling across services.

APIs Documentation

WaterQualityMonitoringSystem.postman_cc

Crowdsourced Data Service – Water Quality APIs

- **POST** /crowddata/waterquality/submit
  Consumes: multipart/form-data
  Request: postalCode, unit, value, observations, citizenId, userName, optional binaries
  Response: CrowdDataResponse<WaterQualityDataResponseDto>
  Errors: 400 (validation failure), 500 (internal error)
- **GET** /crowddata/waterquality/previousReviews
  Query Param: citizenId
  Response: CrowdDataResponse<List<ReviewsResponseDto>>

Crowdsourced Data Service – User APIs

- **POST** /crowddata/user/create
  Consumes: application/json
  Body: UserRequestDto
- **GET** /crowddata/user/get
  Query Param: citizenId
- **GET** /crowddata/user/allUsers *(Admin only)*
- **GET** /crowddata/user/count *(Admin only)*
- **POST** /crowddata/user/update *(Admin only)*
- **GET** /crowddata/user/toggleActivate *(Admin only)*
- **GET** /crowddata/user/rankings

Reward Service APIs

- **POST** /rewards/addReward
  Consumes: application/json
  Body: RewardRequestDto
- **POST** /rewards/rating/add
  Consumes: multipart/form-data
- **GET** /rewards/rating/get/{id}
- **GET** /rewards/rating/all
- **POST** /rewards/rating/update
- **GET** /rewards/rating/toggleRating
- **GET** /rewards/rating/count
- **GET** /rewards/rating/getUserRating

Critical Evaluation:

The development of the Citizen Science Water Quality Monitoring System demonstrates a strong application of microservices architecture principles and modern software engineering practices. One of the most successful aspects of the project is the clear separation of concerns achieved through independent microservices. By isolating the Crowdsourced Data Service and Reward Service, the system ensures that each component can evolve independently without tightly coupling functionality. This design choice enhances scalability, fault tolerance, and long-term maintainability.

The use of Spring Boot provided rapid development capabilities, robust dependency management, and strong support for RESTful APIs. Layered architecture, combined with DTO-based communication, improved code readability and testability. Additionally, integrating Keycloak for authentication and role-based authorisation ensured that administrative endpoints were properly secured, addressing critical security requirements early in the design phase.

Another strength lies in the structured validation of citizen submissions. Enforcing validation rules before data persistence improved data quality and reduced the likelihood of storing incomplete or meaningless records. The reward calculation logic, although not permanently persisted, successfully demonstrates real-time processing and separation of business logic from data storage responsibilities.

However, several areas could have been improved. While the Reward Service processes data correctly, the lack of persistent storage for reward data limits historical analysis and auditability. Persisting reward history could enhance transparency and allow more advanced analytics. Additionally, inter-service communication relies on synchronous REST calls, which may introduce latency and tight runtime dependencies under heavy load. Introducing asynchronous messaging (e.g., event-driven communication) could further improve system resilience.

Testing was implemented at the unit level, but broader integration testing between microservices could strengthen confidence in end-to-end workflows. Similarly, while the API Gateway centralises routing, additional concerns such as rate limiting and request logging could be enhanced to improve operational robustness.

From a frontend perspective, the React-based portals provide a modular and user-friendly interface. However, user experience could be improved through enhanced data visualisation and real-time updates. Incorporating charts and notification mechanisms would further support citizen engagement and administrative monitoring.

Overall, the project successfully meets its functional and technical objectives, demonstrating a professional understanding of microservices, RESTful API design, and secure application development. Future iterations could focus on improving resilience, persistence strategies, and observability to further align the solution with production-grade systems.