ADVANCED OOP — R&D PROJECT PROPOSAL

# Interactive Quiz Engine

A Flask + MySQL platform for teachers to build multi-format quizzes and give students instant, auto-graded feedback.

**TEAM:** Hay Ravanmonyvan & Lim Sak
**COURSE:** Advanced OOP
**STACK:** Python · Flask · MySQL
**LECTURER:** SEK SOCHEAT

# Problem

Why this project matters

Python + Flask + MySQL

### Manual grading is slow

Teachers hand-grade multiple-choice, true/false, and short-answer questions on paper or in spreadsheets, delaying feedback for days.

### No reusable question bank

Quizzes are rebuilt from scratch each term instead of storing and reusing a shared bank of questions.

### No unified tracking

Student scores live across paper, Excel, or ad-hoc forms — there is no single place to see attempt history per class.

### Observed evidence

In our own classes, teachers currently rely on paper tests, Google Forms, or Word documents to run quizzes. None of these tools combine multiple question types, automatic grading, and per-student result tracking in one system — so feedback loops for students are slow and teacher workload is high.

### Who feels this problem

Teachers who need a faster way to author and grade quizzes, and students who want immediate, clear feedback on their understanding instead of waiting days for a graded paper.

# Objectives + Scope

Project goals and semester boundaries

Python + Flask + MySQL

## Measurable Objectives

1. Let a teacher build a quiz using 3 question types (MCQ, True/False, Short Answer) in under 10 minutes.
2. Auto-grade MCQ and True/False questions instantly on submission, with 0 manual steps.
3. Give students their score immediately after submitting a quiz.
4. Let a teacher review every student attempt for a quiz from one results screen.

## In Scope (This Semester)

- Quiz creation & editing (MCQ, True/False, Short Answer)
- Quiz taking flow for students
- Automatic grading for MCQ & True/False
- Manual review queue for short answers
- Role-based access: Admin, Teacher, Student
- Result history per student and per quiz

## Out of Scope (This Semester)

- AI-based grading of open-ended essays
- Plagiarism / cheating detection
- Native mobile app (web only, responsive)
- Live-proctoring or webcam monitoring
- Multi-school / multi-tenant deployment

# Users + Features

User personas and feature priority matrix

Python + Flask + MySQL

### Admin

Manage teacher & student accounts, assign roles

### Teacher

Create quizzes, build question bank, review results, grade short answers

### Student

Take assigned quizzes, view own scores and attempt history

## Feature Priority

| Priority | Feature | Detail |
|---|---|---|
| Must-have | **Quiz builder** | Create quizzes with MCQ, True/False, and Short Answer questions |
| Must-have | **Quiz taking** | Student flow to answer and submit a quiz |
| Must-have | **Auto-grading** | Instant scoring for MCQ & True/False on submit |
| Should-have | **Question bank reuse** | Save questions once, reuse across multiple quizzes |
| Should-have | **Short-answer review** | Teacher screen to manually mark short-answer responses |
| Nice-to-have | **Result analytics** | Class average, hardest question, score distribution |

# OOP Design

Core classes & object-oriented architecture

**Python + Flask + MySQL**

| Class | Responsibility |
| --- | --- |
| **User (base)** → Admin, Teacher, Student | Authentication + role; each subtype exposes only its allowed actions |
| **Quiz** | Holds title, time limit, and an ordered list of Question objects |
| **Question (abstract)** → MCQ, TrueFalse, ShortAnswer | Each subclass implements its own `grade(answer)` method — core polymorphism |
| **Choice** | One selectable option belonging to an MCQQuestion |
| **Attempt** | One student's run of a quiz: list of Answers + total score |
| **ScoringService** | Coordinates grading by calling `question.grade(answer)` for each Answer in an Attempt |

## Key OOP Principle: Polymorphism

`ScoringService` never checks *"if MCQ / if True-False"* — it simply calls `grade()` on any Question object.

Each subclass defines what grading means for its type, so adding a new question type later needs no change to the grading logic.

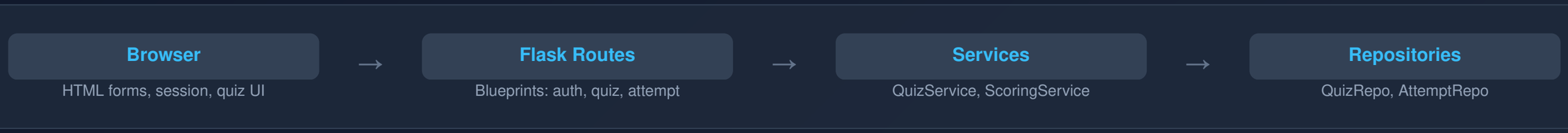
## Repositories (Data Access)

`QuizRepository`, `QuestionRepository`, and `AttemptRepository` isolate all SQL from the Service and Route layers, adhering to repository pattern standards.

# Architecture

Flask layered design pattern

**Python + Flask + MySQL**




## Strict Design Rule

Routes call Services. Services coordinate business logic (e.g., scoring an Attempt). Repositories are the only layer that touches MySQL.

**No SQL and no grading rules live inside app.py or the route functions.**

## Example Request Flow

1. Student submits quiz form
2. `attempt` route receives POST request
3. `AttemptService` builds an Attempt object
4. `ScoringService` grades each Answer
5. `AttemptRepository` saves result to MySQL

# Database Plan

ERD overview & security strategy

**Python + Flask + MySQL**

`ERD Overview: roles → users → quizzes → questions → choices | users + quizzes → attempts → answers`

| Table | Purpose | Key Columns |
|---|---|---|
| **roles** | Admin / Teacher / Student definitions | `id, name` |
| **users** | Login, profile, and role link | `id, name, email, password_hash, role_id` |
| **quizzes** | Quiz metadata | `id, title, time_limit, created_by` |
| **questions** | One question in a quiz, of a given type | `id, quiz_id, type, text, points` |
| **choices** | MCQ options for a question | `id, question_id, text, is_correct` |
| **attempts** | One student's run of a quiz | `id, quiz_id, student_id, score, submitted_at` |
| **answers** | A student's response to one question | `id, attempt_id, question_id, response, is_correct` |

**Security:** Passwords hashed (never stored plain). Parameterized SQL used for all queries to prevent injection.

**Seed Data:** Sample admin/teacher/student accounts, 1 demo quiz per type, completed attempts for testing.

# Timeline + Risks

12-week development plan & risk mitigation

Python + Flask + MySQL

## 12-Week Roadmap (Team of 2)

| Week | Focus | Deliverable |
|---|---|---|
| **Week 1** | Problem validation, topic approval, role split | Problem note + team roles |
| **Week 2** | Proposal writing, ERD & class diagram draft | DOCX draft + architecture |
| **Week 3** | Proposal defense and feedback | Approved / revised proposal |
| **Week 4–5** | Domain models + MySQL schema + seed data | Models + schema + seed data |
| **Week 6–8** | Flask routes, services, repositories, quiz builder | Working feature set |
| **Week 9–10** | Auto-grading, short-answer review, testing | Test report + demo script |
| **Week 11–12** | Final report and defense prep | Final DOCX + PPTX + live demo |

## Key Risks & Mitigation

**Small Team (2 People)**
Limited bandwidth vs 4-person team — lock must-have scope early.

**Short-Answer Subjectivity**
Auto-grading text is unreliable — use a manual teacher review queue.

**Deadline Pressure**
Testing gets rushed — mitigate with weekly milestones & continuous testing.

# Expected Output & Deliverables

Project deliverables and traceability matrix

Python + Flask + MySQL

### 1. Working Software

Flask + MySQL quiz engine: quiz builder, quiz taking, auto-grading, RBAC (Admin/Teacher/Student).

### 2. Testing Evidence

10+ test cases covering scoring logic, access control, and question-type grading paths.

### 3. Documentation

Final DOCX report (design, ERD, implementation, testing) and a defense PPTX deck.

### 4. Live Demo

Teacher builds a quiz → student takes it → instant auto-graded result appears on screen.

## Proposal to Evidence Traceability

| Proposal Item | Final Evidence |
|---|---|
| **Feature list** (builder, taking, grading) | Implemented pages + demo screenshots |
| **OOP design** (Question hierarchy, ScoringService) | Final class diagram + code walkthrough |
| **Database plan** (ERD) | Final schema SQL, seed data, sample queries |
| **Expected output** | Working software, test results, deployment note |

# Thank You

**Interactive Quiz Engine — R&D Project Proposal**

**Team:** Hay Ravanmonyvan & Lim Sak • **Lecturer:** SEK SOCHEAT