# Online Examination System

Project Title: Online Examination System
Language: C++17
Platform: macOS (Apple Silicon)
Date: 21 February 2026
Build System: CMake 3.16+
GUI Framework: Qt 6.10.2 (Widgets)

# Introduction

The **Online Examination System** is a modern desktop application developed using **C++17** that provides a complete platform for conducting Multiple Choice Question (MCQ) based examinations.

The system is designed with a **dual-interface approach**, supporting both:

- A terminal-based console interface
- A modern native desktop GUI built using **Qt 6 Widgets**

A key distinguishing capability of the system is its **AI-powered question generation**, which integrates with:

- Google Gemini (primary)
- OpenAI GPT (fallback)

This allows examiners to dynamically generate topic-specific questions without maintaining a large static question bank. When AI services are unavailable, the system gracefully falls back to local file-based questions, ensuring reliability.

The project demonstrates strong software engineering principles including modular design, separation of concerns, asynchronous timing, and scalable architecture.

# Objectives

The primary objectives of the Online Examination System are:

- To develop a **configurable MCQ examination platform** supporting timed tests and automatic submission.
- To integrate **AI-powered question generation** for dynamic exam creation.
- To provide **dual user interfaces** (Console + Qt GUI) powered by a shared core engine.
- To deliver **instant evaluation and detailed answer review.**
- To maintain **clean, modular C++17 architecture** following modern best practices.
- To support **flexible exam configuration**, including difficulty, duration, and passing criteria.

# System Requirements

## i. Hardware Requirements

| Component | Minimum | Recommended |
|---|---|---|
| Processor | Intel i3 / Apple M1 | Intel i5+ / Apple M1+ |
| RAM | 4 GB | 8 GB |
| Storage | 50 MB | 100 MB |
| Display | 1280×720 | 1440×900+ |
| Network | Required for AI | Broadband |

## ii. Software Requirements

- macOS 12.0 or later
- CMake 3.16+
- Clang/GCC with C++17 support
- Qt 6.x Widgets
- libcurl
- Homebrew package manager

# System Architecture

## i. High-Level Architecture

```
.
├── ./onlineExam/                          (3,526 total lines)
├── ./├── CMakeLists.txt                        23
├── ./├── .env                                   4
├── ./├── main.cpp                              543
├── ./├── data/
├── ./│   └── questions.txt                     500
├── ./├── include/                         (224 lines)
├── ./│   ├── APIQuestionGenerator.h             28
├── ./│   ├── ConsoleUI.h                        14
├── ./│   ├── EnvLoader.h                        11
├── ./│   ├── ExamConfig.h                       61
├── ./│   ├── Question.h                         30
├── ./│   ├── QuestionBank.h                     17
├── ./│   ├── ResultAnalyzer.h                   24
├── ./│   ├── Student.h                          18
├── ./│   └── TimerManager.h                     21
├── ./├── src/                             (726 lines)
├── ./│   ├── APIQuestionGenerator.cpp          450
├── ./│   ├── ConsoleUI.cpp                      44
├── ./│   ├── EnvLoader.cpp                      56
├── ./│   ├── Question.cpp                       47
├── ./│   ├── QuestionBank.cpp                   60
├── ./│   ├── ResultAnalyzer.cpp                 19
├── ./│   ├── Student.cpp                        13
├── ./│   └── TimerManager.cpp                   37
└── ./└── gui/                           (1,506 lines)/
    ├── ./└── gui/                           (1,506 lines)/├── guimain.cpp                    355
    ├── ./└── gui/                           (1,506 lines)/├── SetupWidget.h / .cpp       41 + 334
    ├── ./└── gui/                           (1,506 lines)/├── ExamWidget.h / .cpp        67 + 283
    ├── ./└── gui/                           (1,506 lines)/├── ResultWidget.h / .cpp      38 + 263
    └── ./└── gui/                           (1,506 lines)/└── MainWindow.h / .cpp        32 +  93
```

# ii. Design Principles

- **Separation of Concerns**

Each layer has a clearly defined responsibility:

- UI handles presentation
- Core handles business logic
- API layer handles external services
- Data layer handles persistence

- **Shared Core Engine**

Both Console and Qt GUI reuse the same:

- Question model
- Timer
- Result analyzer
- Question generator

This ensures:

- code reuse
- easier testing
- consistent behavior

- **Fault-Tolerant Question Loading**

The system uses a **cascading fallback strategy**:

```
Gemini → OpenAI → Local File
```

This guarantees exam continuity even if APIs fail.

# Module Description

## i. Question Module

**Purpose:** Represents a single MCQ.

**Key Data:**

- questionText
- options (vector of 4)
- correctIndex
- difficulty
- topic

**Responsibilities:**

- store question data
- provide getters
- support option shuffling

## ii. QuestionBank

**Purpose:** Manages collection of questions.

**Functions:**

- load from file
- shuffle questions
- provide question list

**Special Feature:**

Uses **std::set-based deduplication** to avoid repeated questions.

## iii. Student

Simple data holder for:

- student name
- student ID

## iv. ExamConfig

Central configuration structure containing:

- question source
- topic
- difficulty
- number of questions
- exam duration
- passing percentage

This makes the system highly configurable.

## v. ResultAnalyzer

Responsible for:

- tracking correct/wrong/skipped
- computing score
- computing percentage
- pass/fail evaluation

## vi. TimerManager

Implements asynchronous countdown using:

- `std::thread`
- `std::atomic`

Features:

- real-time countdown
- auto-submit on timeout
- thread-safe state

## vii. APIQuestionGenerator

Handles AI integration.

**Responsibilities:**

- call Gemini API
- call OpenAI API
- parse JSON responses
- build Question objects
- manage fallback logic

## viii. EnvLoader

Loads environment variables from **.env** file securely without hardcoding secrets.

# Data Design

- **Question File Format**

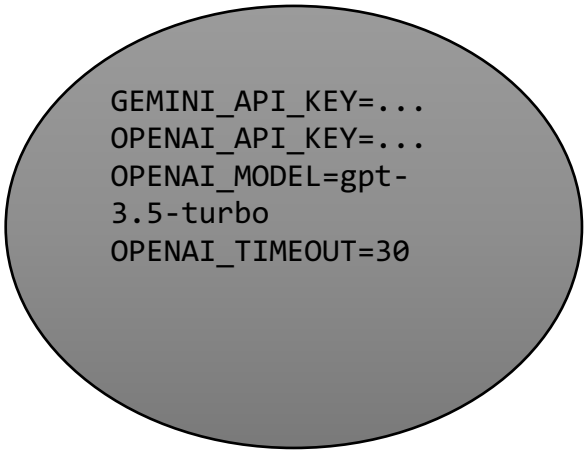
Pipe-delimited format:

question|opt1|opt2|opt3|opt4|correct|difficulty|topic

Example:

What is the capital of
France?|Berlin|Madrid|Paris|Rome|2|easy|geography

- **Environment File**



# User Interface Design

## i. GUI Flow

The Qt application follows a **3-screen wizard**:

1. Setup Screen
2. Exam Screen
3. Result Screen

## ii. Setup Screen

Features:

- student info form
- source selection
- difficulty selector
- exam duration
- validation checks

## iii. Exam Screen

Displays:

- live timer
- progress bar
- question text
- radio button options
- Next / Skip controls

Timer color states:

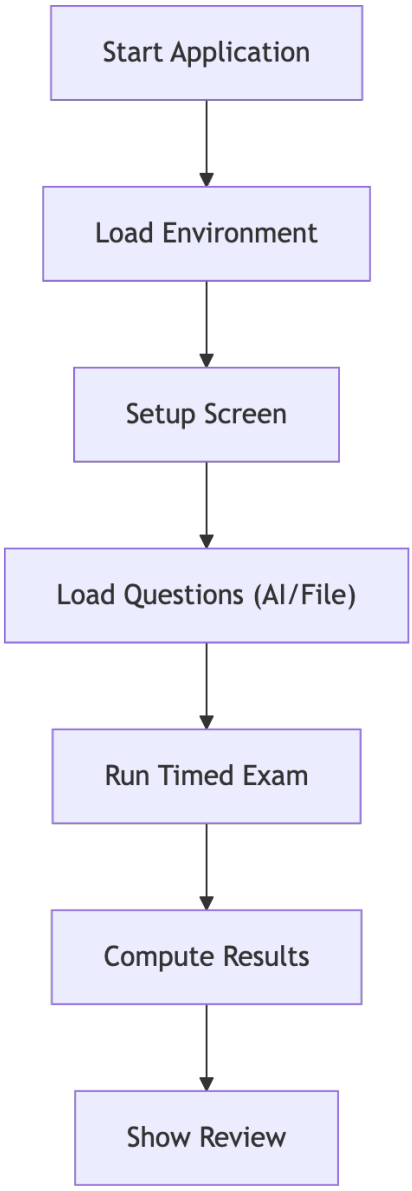
- Blue → normal
- Orange → warning
- Red → critical

## iv. Result Screen

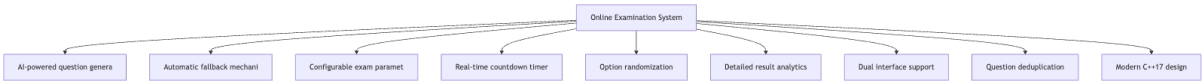
Shows:

- pass/fail badge
- score and percentage
- statistics summary
- detailed review table

## Application Flow :



# Key Features




# Build and Installation

```
brew install qt

brew install cmake

mkdir build && cd build

cmake .. -DCMAKE_PREFIX_PATH=/opt/homebrew/opt/qt

cmake --build . -j10
```

# Testing Strategy

The system was manually tested for:

- configuration validation
- API fallback
- timer expiration
- scoring accuracy
- GUI navigation
- console flow
- duplicate handling

All major functional paths were verified.

# Limitations

- no database persistence
- no authentication system
- standalone desktop only
- limited local question diversity
- custom JSON parser (not fully robust)
- API rate limits

# Future Scope

High-value upgrades include:

- SQLite integration
- PDF result export
- question editor GUI
- user authentication
- multi-topic exams
- cross-platform builds
- analytics dashboard
- network-based exams

# Conclusion

The **Online Examination System** successfully demonstrates a robust, modular, and extensible platform for conducting MCQ-based examinations.

The project highlights strong competencies in:

- Modern C++17 development
- Qt GUI engineering
- API integration
- modular software architecture
- asynchronous programming
- scalable system design

With further enhancements such as database storage and analytics, the system can evolve into a full-scale production examination platform.