4.14 Design

The Design section of the AQA A-level Computer Science NEA is where you show, before and during development, how your proposed solution will be structured. AQA refers to this as the **documented design**, and it is worth **12 marks**. The aim is not to write pages of vague planning, but to communicate a clear technical design that another competent developer could understand and use to build the solution.

Purpose of the Design Section

Your design should prove that you have thought carefully about how the system will work before implementation. It should link directly to the problem, objectives and success criteria from the Analysis section. Every major feature described in your objectives should have some form of design evidence, such as algorithms, data structures, database tables, user interface plans or system diagrams.

What AQA Expects

- **Clear communication:** the design must be understandable to a third party.
- **Appropriate technical detail:** use diagrams, pseudocode, database designs, class diagrams, wireframes or other suitable design tools.
- **Connection to the problem:** the design should clearly support the objectives identified in the Analysis section.
- **Evidence of decomposition:** show how the full problem has been broken down into smaller, manageable parts.
- **Iterative development:** it is acceptable to update the design as the project develops, but changes should be explained.

Recommended Structure

1. Overview of the Proposed Solution

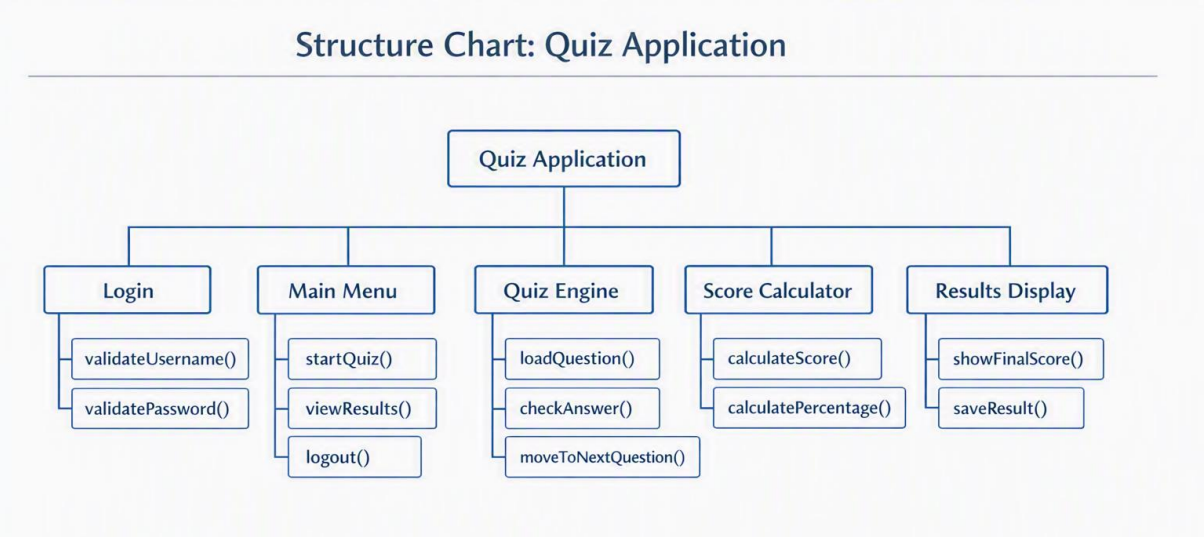
Begin with a short explanation of what the finished system will do. This should not repeat the whole Analysis section, but should summarise the main components of the proposed solution. For example, describe the main users, the key inputs and outputs, and the major processes the system will perform.

2. System Architecture or Structure Diagram

Include a diagram showing the main parts of the system and how they connect. This could be a structure chart, class diagram, component diagram or high-level system diagram. The purpose is to show decomposition: how the project has been divided into modules, classes, functions or screens.

Example: Structure Chart

For a quiz application, the structure chart could show the system broken down into modules such as Login, Main Menu, Question Manager, Quiz Engine, Score Calculator and Results Display. Each module should be linked to the functions or procedures it controls.

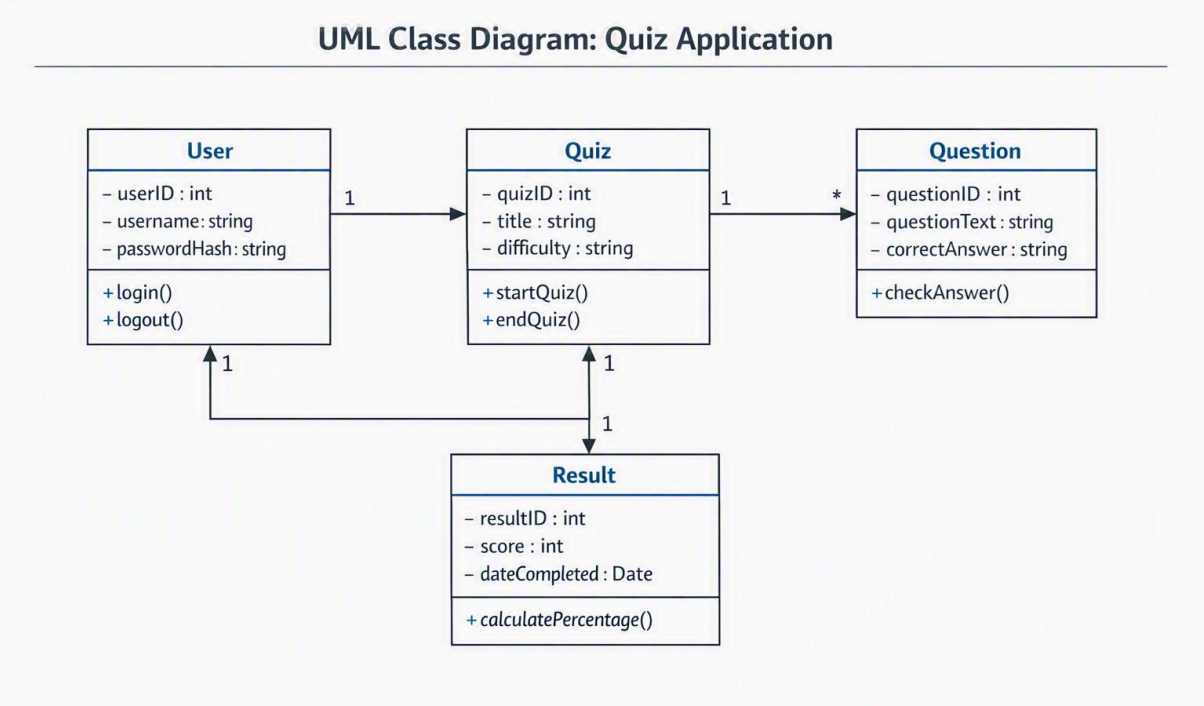


This example shows decomposition because the overall problem is divided into smaller parts. In the NEA write-up, the student should briefly explain what each module does and how it supports the project objectives.

Example: Class Diagram

For an object-oriented quiz application, a class diagram could show the main classes, their attributes, methods and relationships. This helps demonstrate how the system will be organised before coding begins.

Class	Attributes	Methods	Relationship
User	userID, username, passwordHash	login(), logout()	A User can complete many Quizzes
Quiz	quizID, title, difficulty	startQuiz(), endQuiz()	A Quiz contains many Questions
Question	questionID, questionText, correctAnswer	checkAnswer()	A Question belongs to one Quiz
Result	resultID, score, dateCompleted	calculatePercentage()	A Result links a User to a Quiz attempt

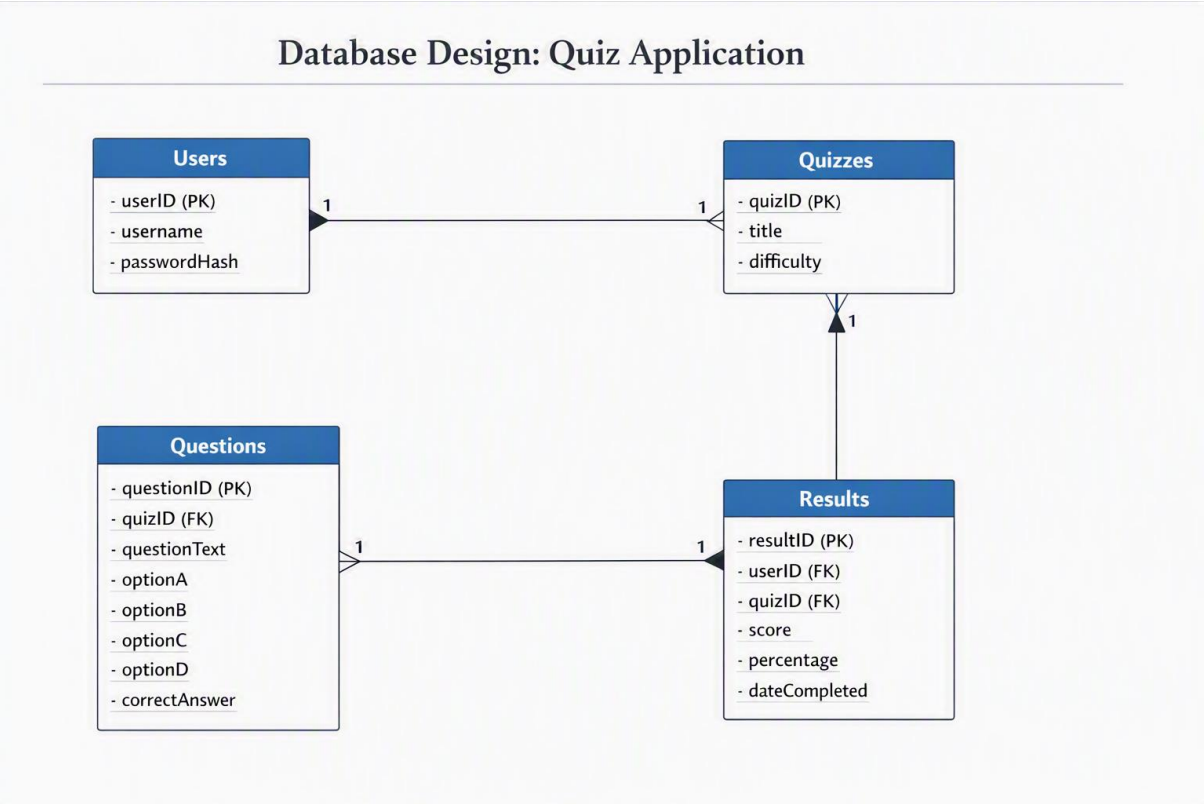


The class diagram should not just list class names. It should make clear which data each class stores, what behaviours it performs, and how classes interact with each other.

Example: Database Design

For a quiz application, a relational database could be used to store users, quizzes, questions and results. The design should show table names, primary keys, foreign keys, field types and relationships.

Table	Key fields	Other fields	Purpose
Users	userID primary key	username, passwordHash	Stores registered user accounts
Quizzes	quizID primary key	title, difficulty	Stores each quiz available in the system
Questions	questionID primary key, quizID foreign key	questionText, optionA, optionB, optionC, optionD, correctAnswer	Stores questions linked to a specific quiz
Results	resultID primary key, userID foreign key, quizID foreign key	score, percentage, dateCompleted	Stores each user's quiz attempt



The relationships could be written as: one Quiz has many Questions, one User has many Results, and one Quiz has many Results. This gives a clear link between the database structure and the features needed by the system.

3. Data Design

Explain how data will be stored, processed and validated. Depending on the project, this may include database tables, entity-relationship diagrams, file structures, arrays, lists, dictionaries, classes or external data sources. For each important data item, explain its purpose, data type and any validation rules.

Data item	Data type	Purpose	Validation
Example: username	String	Stores the user's login name	Must be unique and not blank

4. Algorithm Design

For the most important or complex parts of the system, include pseudocode, flowcharts or written algorithm explanations. Focus on algorithms that demonstrate problem solving, not on simple procedures such as opening a menu. Good choices include searching, sorting, optimisation, validation, calculations, game logic, pathfinding, simulation rules, recommendation logic or database queries.

5. User Interface Design

If the project has a user interface, include sketches, wireframes or screen designs. Annotate each screen to explain the purpose of buttons, input fields, menus, messages and outputs. Make sure the interface design supports the needs of the intended users identified in the Analysis section.

6. Validation and Error Handling Design

Describe how the system will prevent, detect and respond to invalid data or user errors. For example, explain how empty inputs, incorrect formats, duplicate records, impossible values or unauthorised access will be handled. This section helps link the design to later testing.

How to Access the Higher Marks

- Make the design specific to your own project rather than generic.
- Use several appropriate design methods, not just screenshots or rough notes.
- Show the design of complex algorithms and data structures in detail.
- Explain why key design decisions were made.
- Show how the design changed during development if your approach evolved.
- Ensure every major objective from Analysis is covered by the design.

Common Mistakes to Avoid

- Writing a description of the finished program instead of designing it.
- Including only interface sketches with no algorithm or data design.
- Using diagrams without explaining what they mean.
- Not linking the design to the objectives and success criteria.
- Leaving out complex parts of the project because they are difficult to explain.
- Adding design evidence after coding without explaining changes honestly.

Design Section Checklist

- Have I explained the overall solution clearly?
- Have I decomposed the system into manageable parts?
- Have I included suitable diagrams for structure, data or processes?
- Have I designed the main algorithms using pseudocode, flowcharts or clear explanations?
- Have I shown how data will be stored and validated?
- Have I designed the user interface where relevant?
- Have I linked my design back to the objectives from Analysis?
- Have I explained any changes made to the design during development?