**Chapter 4: Methodology**

**4.1 Introduction**

This chapter outlines the methodology followed in the development of the Quiz AI system. It explains the overall approach used to design, implement, and evaluate the system, highlighting the frameworks, models, and tools that guided the process. The chosen methodology emphasizes flexibility, adaptability, and continuous improvement, ensuring that the project can respond effectively to evolving requirements and feedback. Industry-recognized practices such as Agile software development, layered architecture, and structured testing were adopted to provide a systematic framework for building a reliable and scalable solution. Each section of this chapter discusses a key aspect of the methodology, from the project development approach and system architecture to tools, technologies, data collection, testing, and user interface design.

**4.2 Project Development Approach**

Our project follows the Agile Software Development Model. We chose this approach because it emphasizes flexibility, iterative development, and continuous improvement, which suit our project's dynamic nature. The system we are building includes user authentication, content upload, AI-based quiz generation, public exam creation, and analytics. It needs frequent adjustments based on feedback from stakeholders and changing requirements. Agile lets us make these changes without disrupting the overall development process.

Agile development works in short iterative cycles called sprints. Each sprint produces a functional part of the system. After each iteration, we gather feedback from stakeholders and make necessary refinements. This helps ensure that the final product is functional and meets user needs and expectations.

**The advantages of Agile for this project include:**

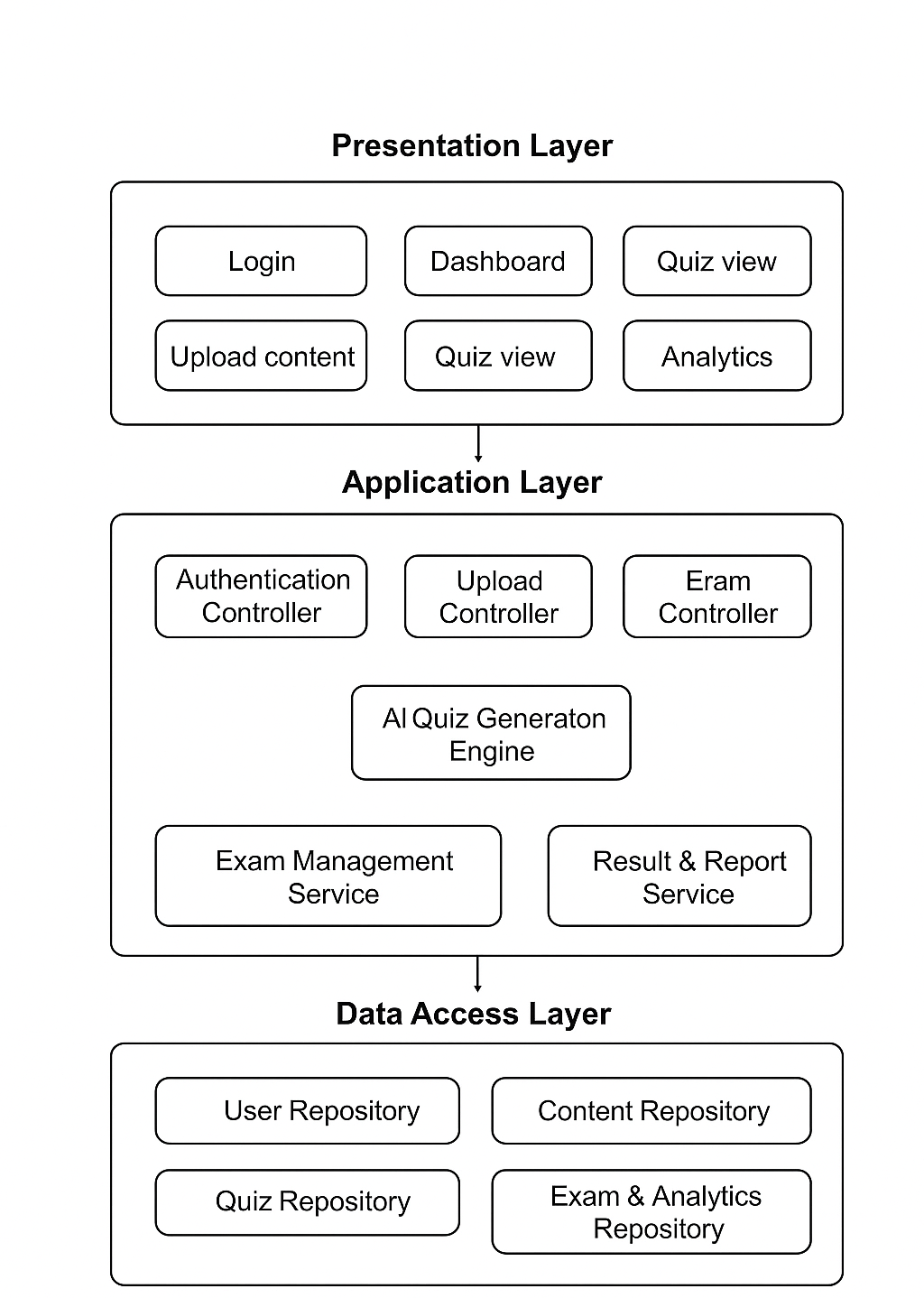
* Adaptability: We can modify or improve features and user interfaces during development without needing complete redesigns or causing significant delays.
* Early Delivery: The system is built step by step, allowing us to deliver functional parts like user login, content upload, or quiz generation early. This enables testing and user validation sooner.
* Customer Collaboration: Agile encourages close communication with stakeholders throughout development. This ensures the product evolves based on actual user requirements rather than assumptions.
* Risk Management: Continuous testing and incremental delivery help reduce the risk of major failures later on. Issues are identified and resolved early.

**The project will progress through the following phases, repeated in cycles:**

1. **Planning**: Define the overall scope of the system, prioritize features, and create a backlog of tasks for each sprint.
2. **Design**: Develop detailed user interface (UI) mockups, database architecture, and integration diagrams to clarify the technical structure.
3. **Implementation**: Code the core functionalities of the system, such as authentication, content upload, AI quiz generation, exam management, and analytics modules. Each feature is built incrementally during sprints.
4. **Testing**: Conduct rigorous testing at multiple levels. This includes unit testing for individual components, integration testing for module interactions, and user acceptance testing (UAT) to validate requirements.
5. **Deployment**: Release working versions of the application to a staging environment for stakeholder review, followed by full deployment to production.
6. **Maintenance & Continuous Improvement**: Address any issues reported post-deployment, release updates, add new features based on feedback, and enhance system performance and usability.

**4.3 System Architecture**

The layered architectural style was chosen for this software system because it organizes functions into separate layers. This structure promotes the separation of concerns by dividing the system into independent layers, each handling a specific group of related tasks. This setup improves maintainability, scalability, and testability by allowing changes within one layer without greatly affecting the others. It also makes the code easier to read and establishes a clear dependency direction, where upper layers depend on lower layers, which reduces coupling. These benefits make the layered architecture an excellent choice for complex systems that need clarity, modularity, and long-term flexibility.



**Diagram 4.1: Software Architecture.**

The diagram shows a Layered Architecture made up of five main layers. Each layer represents a specific level of system abstraction and responsibility.

At the top, the **View Layer** manages user interaction. It handles input and output to provide an intuitive interface. Below it, the **Controller Layer** serves as a middleman. It processes user requests and coordinates responses between the view and the application logic.

The **Service Layer** contains business logic. It ensures that main operations and rules are carried out correctly and efficiently. Under this, the **Domain Layer** manages the system’s data and core entities. It provides a structured view of the application’s business model.

Finally, the **Infrastructure Layer** is at the bottom. It provides essential technical services like database management, external system integration, and network communication. This organized approach improves modularity, maintainability, and scalability. It makes system evolution easier and enhances the separation of concerns.

**4.4 Tools and Technologies**

**Programming Languages**

**C#**  
It is reliable, efficient, and well-suited for building robust backend systems or desktop applications. It integrates seamlessly with .NET libraries, offering strong type safety and excellent performance.  
It reduces runtime errors, provides scalability for large systems, and allows rapid development of stable APIs or services that power the application.

**Python**  
Python is widely used in machine learning and AI development because of its clean syntax, extensive libraries, and community support.  
It enables quick prototyping, easy integration with frameworks like QLoRA and TinyLlama, and fast iteration when building or refining AI components.

**React**  
React is rather ideal for creating web interfaces with a component-based architecture that promotes reusability.  
It also allows a smoother user experience than some alternatives, reduces development effort for the frontend, and ensures scalability as the application grows.

**Frameworks**

**QLoRA**  
QLoRA (Quantized Low-Rank Adaptation) fine-tunes large language models efficiently by lowering memory usage without compromising accuracy.  
It also makes it affordable and practical to adapt advanced AI models to our project requirements.

**TinyLlama API**  
TinyLlama provides a lightweight API for large language model capabilities, focusing on speed and low resource consumption, which is great for the hardware we have access to as developers.  
faster AI responses, reduced latency, and an overall smoother experience for end-users are some of its benefits.  
Additionally due to its highly curated data set it’s does a great job at data extraction and understanding without having to worry about hallucinations.

**Development Tool**

**Microsoft Visual Studio Code**  
Why chosen: VS Code is a lightweight yet powerful code editor with broad language support, integrated debugging, Git control, and an extensive extension marketplace.  
How it helps the project: It increases developer productivity, streamlines workflows across different programming languages, and enables rapid switching between backend, AI, and frontend development.

**SQL Server Management Studio**  
SSMS provides a comprehensive and user-friendly interface for managing SQL Server databases. It's a key tool for database administrators and developers, offering a visual way to design, query, and maintain databases. Its powerful features streamline tasks like performance monitoring, security management, and data backup, ensuring the database is stable, secure, and performant.  
As our backend is powered by C#, we'll be using a SQL Server database for data storage. SSMS is the standard tool for working with this database. It allows us to easily design the database schema, write and debug complex queries, and manage the database's health and security. This is crucial for building a scalable and reliable backend that can handle our application's data needs efficiently.

**Overall Impact**

* Backend stability and scalability (C# + VS Code)
* Fast AI development and integration (Python + QLoRA + TinyLlama)
* Modern, user-friendly frontend (React)
* Efficient resource usage with advanced AI (QLoRA quantization, TinyLlama’s lightweight models)

**4.5 Data Collection and Analysis**

The data collection process for the proposed Quiz AI system focuses on preparing educational resources that can be used to fine-tune the TinyLlama model for quiz generation. Since ready-made datasets for the intended task are limited, multiple sources are considered, including open repositories such as Kaggle and Hugging Face, along with instructor-provided materials like books, lecture slides, and past exams. The collected data is organized into structured formats (e.g., JSON) containing pairs of content and questions, instruction–response samples, and exam-style Q&A sets. Each entry is further enriched with metadata such as subject tags, difficulty levels, and answer types to enhance the model’s ability to generate adaptive and context-aware quizzes. After collection, the data undergoes preprocessing, cleaning, and validation to ensure accuracy and consistency. Finally, the analysis phase examines the dataset coverage across different subjects and levels of difficulty, ensuring it is representative enough for fine-tuning and evaluation.

**4.6 Testing Strategy**

Complete, multi-layered testing will be employed to certify that the Quiz AI is accurate, reliable, and meets all functional and non-functional requirements. The strategy will use automated and manual testing techniques throughout the development process.

**4.6.1 Testing Methodology**

The testing approach will be centered on Unit Testing and Burp Suite.

**Unit Testing:**

* **Objective**: To confirm every single method, function, and class operates correctly independently of the other system parts. It is highly recommended to detect logic bugs as early as possible during the development cycle.
* **Method**: Automated tests will be written by developers for the lowest level of units of code. These are testing functions for text processing, quiz question generation, input validation, and data formatting. External dependencies such as calls to AI APIs or file system calls will be mocked out with mock objects so that tests are fast, consistent, and isolated.
* **Tool**: The project will utilize the unit test framework, Python's default unit test tool. The framework provides a solid base upon which to build and run a solid set of tests.

**Penetration Testing & Security Scanning:**

* **Purpose**: To actively scan and fix security vulnerabilities in the application before it is deployed.
* **Procedure**: The deployed application will be scanned for typical web vulnerabilities systematically. This includes testing injection flaws (SQLi, XSS), broken authentication, insecure direct object reference, and other on the OWASP Top 10 list.
* **Tool**: The primary tool for manual and automated security testing will be Burp Suite. It will be used to intercept, observe, and manipulate HTTP/S requests between the server and client in an effort to find security vulnerabilities.

**4.6.2 Error Identification and Correction**

A transparent process will be employed to deal with problems that are found:

* **Identification**: Problems will be identified by automated unit test failures and security scans.
* **Triage & Logging**: Problems from functional bugs to severe security bugs will be logged, prioritized for repair, and sorted based on severity.
* **Resolution**: Bugs will be repaired by developers, while security bugs will be repaired using a patch. Security patches will be tested to ensure that they do not introduce new vulnerabilities.
* **Validation**: The unit test suite and The Burp Suite will be re-run to verify that the fix is working without causing any regressions.

**4.6.3 Test Cases and Validation**

The following table provides examples of test cases for both unit and security testing.

| **TEST CASE ID** | **TYPE** | **DESCRIPTION** | **INPUT EXPECTED** | **OUTCOME** |
| --- | --- | --- | --- | --- |
| TC-U-101 | Unit Test | Test input validation for file upload. | A file with type .exe | Function raises a ValueError. |
| TC-U-102 | Unit Test | Test JSON formatting of a quiz question. | Question data object. | A valid JSON string with correct fields. |
| TC-SEC-201 | Security (Burp Suite) | Test for SQL Injection in login form. | username: admin' -- | Returns a generic error message, not a database error. |
| TC-SEC-202 | Security (Burp Suite) | Test for Cross-Site Scripting (XSS) in quiz output. | <script>alert('test')</script> | Input is sanitized; script tags are not executed. |

**Table 4.1: Summary of Unit and Security Test Cases for System Validation**

**4.7 User Interface Design (Prototype)**

Provide snapshots for the graphical user interface screens of the system (without code). You can use special software, such as Figma.