MANSOURA UNIVERSITY
FACULTY OF ENGINEERING
ARTIFICIAL INTELLIGENCE ENGINEERING PROGRAM

SIR

PROJECT MEMBERS	ID NUMBER
Mazen Amir Mohamed Bakry	805542174
Adel Mahmoud Adel Shousha	806237372
Mahmoud Mohamed Mahmoud Sadeq	805544800
Youssef Ahmed Farouk Elkotby	808962813
Mohamed Ibrahim Taha Mohamed	806127660
Ramy Waleed Abdullah Al-Baqiry	805774428
Karim Atef Gamal Mashaly	806147227
Mazen Ayman Abdel-Aziz Awad	805502582
Ahmed Mohammed Ahmed Foda	805495108
Omar Amir Elmasry Zidane	805495104
Mostafa Saad Abdo Shaheen	805498138

Supervisor

Dr. Islam Ismael Abdullah

Eng. Mohamed El-Madaawy

Academic Year: 2024 – 2025

Abstract

According to recent educational statistics, teachers spend an average of 5-7 hours per week creating and grading quizzes manually, with essay questions requiring even more time for subjective evaluation. This time-intensive process creates a significant burden on educators and often leads to delayed feedback for students, hindering their learning progress. Hence, the main problem we are addressing is the inefficient manual process of quiz creation and grading, affecting teachers across all educational levels and their students' learning experience.

In SIR, we employ a structured scientific methodology to address these educational challenges through AI-powered automation. Our approach begins with enhancing content processing capabilities by teaching the system to generate diverse, high-quality quiz questions from uploaded PDF materials using advanced AI agents. This is achieved through a robust PydanticAI framework integrated with Gemini API that analyzes educational content and creates both multiple-choice and essay questions. Once the system generates questions, it transitions to intelligent evaluation, where student responses are automatically graded with detailed feedback using AI-powered assessment models.

This process is applied to a variety of educational content categories, from academic subjects to professional training materials, ensuring a versatile and practical tool for educators. To further enhance real-world usability, SIR places the quiz experience within a comprehensive mobile application that connects teachers and students seamlessly. After teachers create and publish quizzes, students can access them through an intuitive interface, receive immediate feedback, and track their progress, ensuring the application's functionality extends beyond simple automation to a complete educational ecosystem that improves learning outcomes in real classroom environments.

ACKNOWLEDGEMENTS

Dear Dr. Islam Ismael Abdullah and Eng. Mohamed El-Madaawy, as we approach the end of our journey as students, we are grateful for the knowledge, support, and guidance that you have provided us throughout our project. Your constant encouragement and unwavering commitment to excellence have helped us achieve our goals. We would like to express our deepest appreciation for your invaluable contributions to our project. Your expertise and willingness to share your knowledge with us have been instrumental in shaping our project's success. Special thanks go to Dr. Abeer Tawakol for her exceptional guidance and mentorship. We appreciate her dedication to ensuring that we produce a high-quality project and her assistance in overcoming any challenges. Your support and commitment to our project have helped us develop not only academically but also professionally. We are grateful for the time and effort you invested in us, and we will always remember your valuable teachings. Again, thank you for everything. We will always be proud to have had the opportunity to work under such distinguished professionals.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Nomenclatures

SIR	Study,Interact,Review
AI	Artificial Intelligence
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
ERD	Entity-Relationship Diagram
Git	Global Information Tracker
GPT	Generative Pre-trained Transformer
GUI	Graphical User Interface
JSON	JavaScript Object Notation
LLM	Large Language Model
RAG	Retrieval-Augmented Generation
SDKs	Software Development Kits
UI	User Interface
UX	User Experience

CHAPTER 1: Project Description

1.1 Problem Statement

1.1.1 Introduction to Educational Assessment Challenges

The ***SIR*** project is a mobile application designed to revolutionize the educational assessment process by leveraging artificial intelligence (AI) to automate quiz creation and grading. This chapter provides a comprehensive overview of the problem addressed, the project's requirements, and its objectives and expected outcomes, highlighting its significance in enhancing educational efficiency and fairness.

In the modern educational landscape, professors and educators face significant challenges in designing and administering assessments that effectively evaluate student learning. Quizzes and tests are fundamental tools for gauging comprehension, but their creation and grading processes are often labor-intensive and fraught with complexities. The manual development of quiz questions requires professors to carefully align content with learning objectives, ensure clarity, and maintain an appropriate level of difficulty, all of which demand substantial time and effort.

1.1.2 Impact on Professors

The time-consuming nature of quiz creation is particularly burdensome given the multifaceted responsibilities of professors, which include delivering lectures, conducting re-search, mentoring students, and fulfilling administrative duties. Crafting high-quality questions, especially for subjective formats like essays, can take hours, detracting from time that could be spent on innovative teaching methods or personalized student support. This challenge is exacerbated in large classes or institutions with multiple sections, where the volume of assessments can be overwhelming.

1.1.3 Fairness and Consistency in Grading

One of the most pressing issues in educational assessment is ensuring fairness and consistency in grading, particularly for subjective questions. Essay responses, which require nuanced evaluation, are susceptible to human bias and variability. Professors may unintentionally apply different standards to different students or vary in their grading rigor over time due to fatigue or other factors. This inconsistency can lead to perceptions of unfairness among students, potentially undermining trust in the educational process and affecting academic outcomes.

Ensuring equitable evaluation across diverse student populations is a critical challenge that demands a solution capable of standardizing assessments without sacrificing quality.

1.1.4 Impact on Student Learning

The quality and timeliness of feedback are crucial for student learning, yet the manual grading process often limits the depth and speed of feedback provided. Detailed com-ments on essay responses, which help students understand their mistakes and improve, are time-intensive to produce. As a result, students may receive delayed or superficial feed-back, hindering their ability to address weaknesses promptly and engage deeply with the material. Additionally, traditional quiz formats may lack interactivity, reducing student engagement and motivation.

1.1.5 The Role of Technology in Education

As educational institutions increasingly adopt digital solutions, there is a growing need for tools that integrate seamlessly into teaching workflows while enhancing efficiency and innovation. The ***SIR*** project addresses these challenges by leveraging AI to automate quiz creation and grading, offering a transformative solution that saves time, ensures fairness, and enhances the teaching experience. By generating questions from uploaded PDF documents, ***SIR*** aligns assessments with course content effortlessly. Its AI-driven grading system provides consistent, objective evaluations of essay responses, reducing bias and ensuring equitable treatment of all students. Furthermore, ***SIR***'s immediate feedback mechanism empowers students to learn from their performance in real-time, fostering a more engaging and effective learning environment.

1.1.6 *SIR*'s Contribution to Teaching

Beyond addressing logistical challenges, ***SIR*** aims to make teaching more enjoyable and innovative. By automating routine assessment tasks, professors can dedicate more time to creative instructional strategies, such as interactive discussions, hands-on activities, or personalized mentoring. This shift allows educators to bring out their best, fostering a dynamic classroom environment that inspires both teachers and students. ***SIR*** represents a step toward modernizing education, aligning with the trend of integrating technology to enhance teaching practices.

1.2 Project Requirements

To effectively address the identified challenges, the ***SIR*** application must meet a comprehensive set of functional and non-functional requirements.

1.2.1 Functional Requirements

1. User Management:

• Provide sign-up and sign-in functionality with role-based access for teachers and students.

• Collect user information including name, email, password, date of birth, and gender during registration.

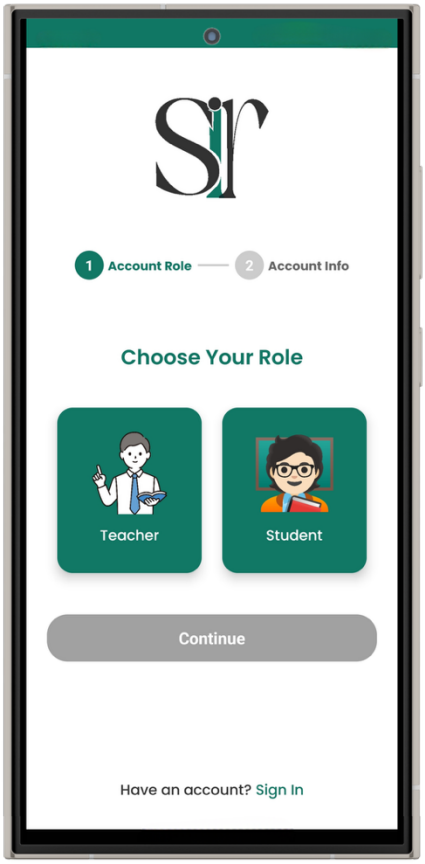


Figure 1: Sir App Role Selection Screen for New User Registration

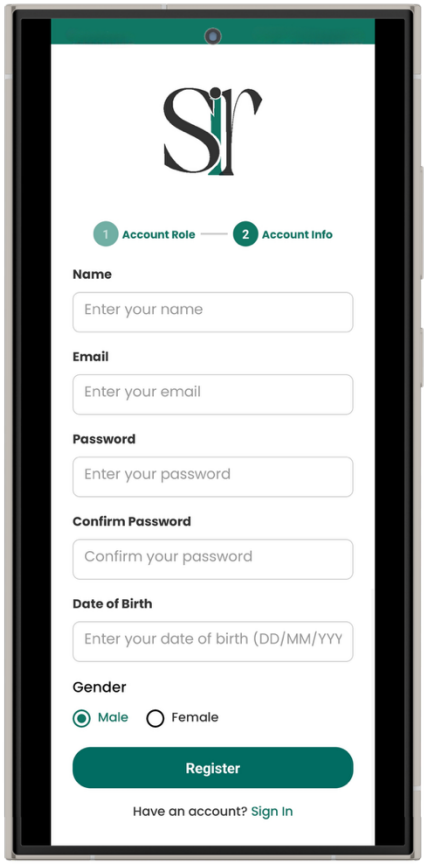


Figure 2: Sir App Account Information Registration Form

2. Class Management:

• Enable teachers to create classes with names and descriptions, generating unique invite codes for student enrollment.

• Allow teachers to manage student lists, including the ability to remove students.

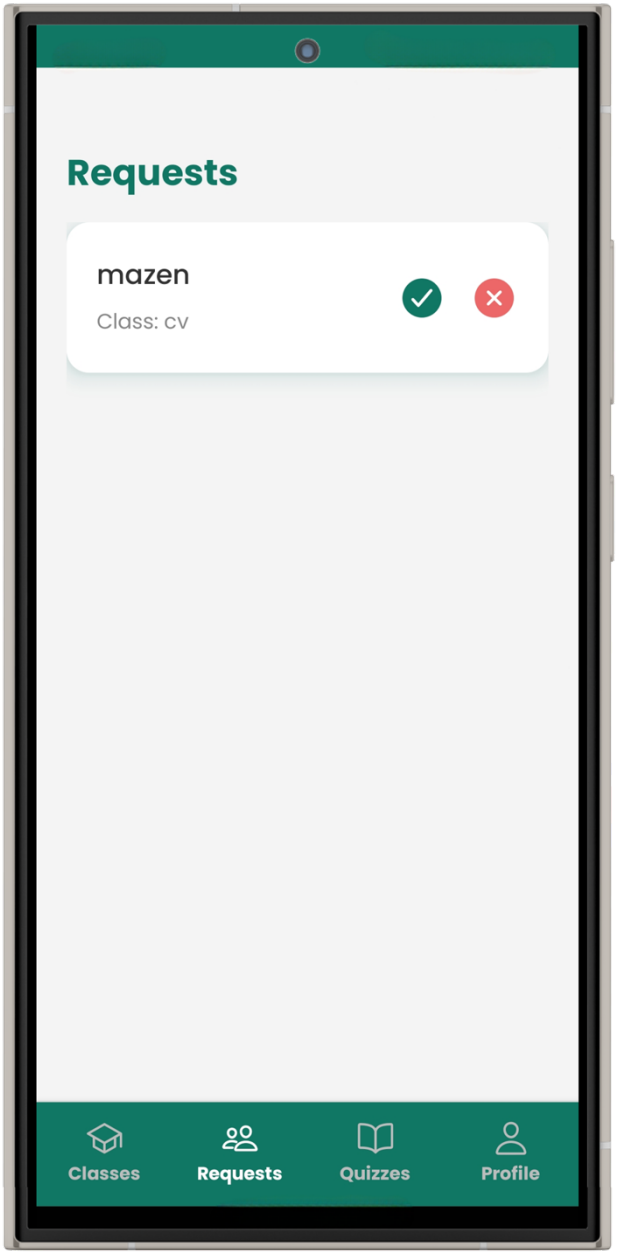


Figure 3: Teacher Requests Page with Student Join Request

3. Quiz Management:

• Support quiz creation with customizable parameters (name, description, du-ration, question types).

• Allow PDF uploads for AI-generated questions, with a review interface for teachers to edit questions.

• Provide options to enable/disable quizzes and control result visibility.

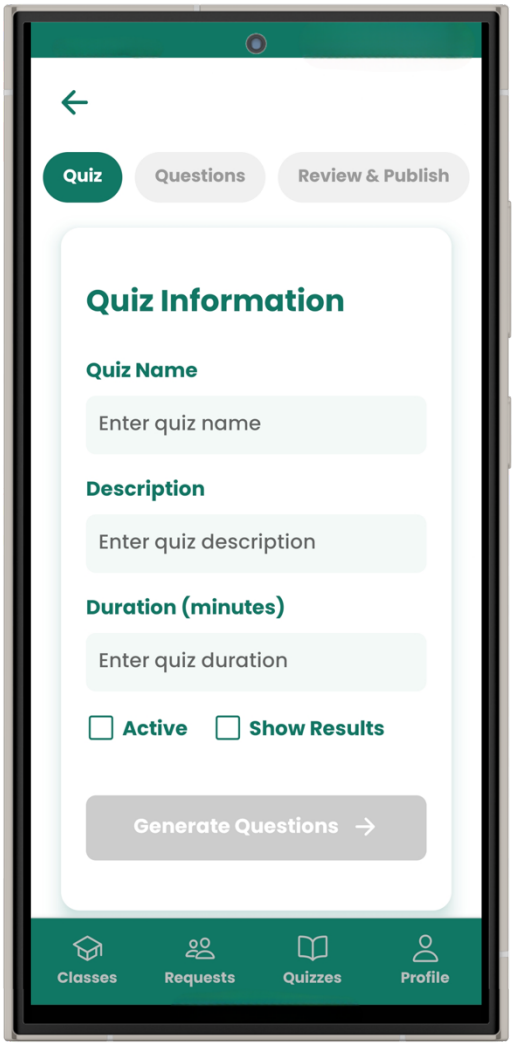


Figure 4: Quiz Creation Form with Basic Information Fields

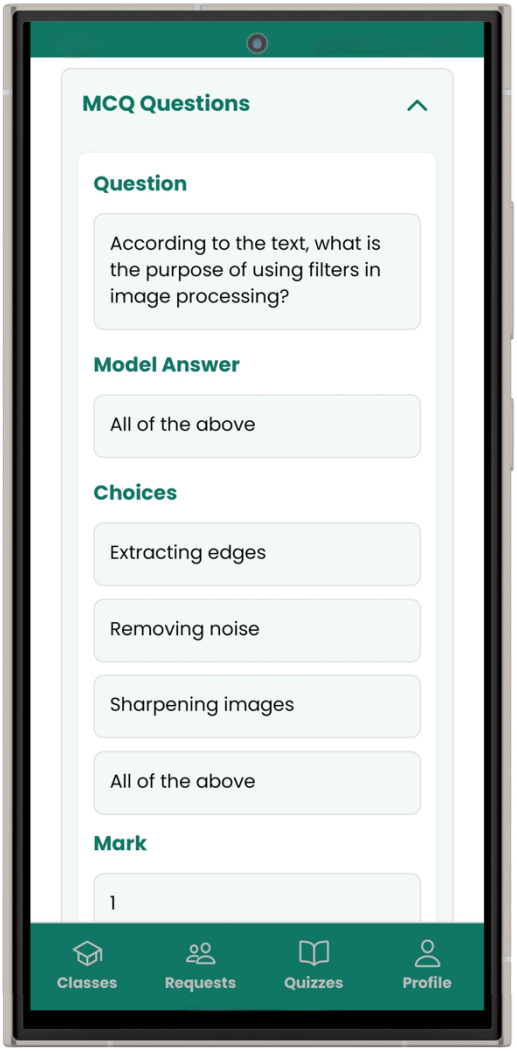


Figure 5: MCQ Question Review Interface with Answer Choices and Scoring

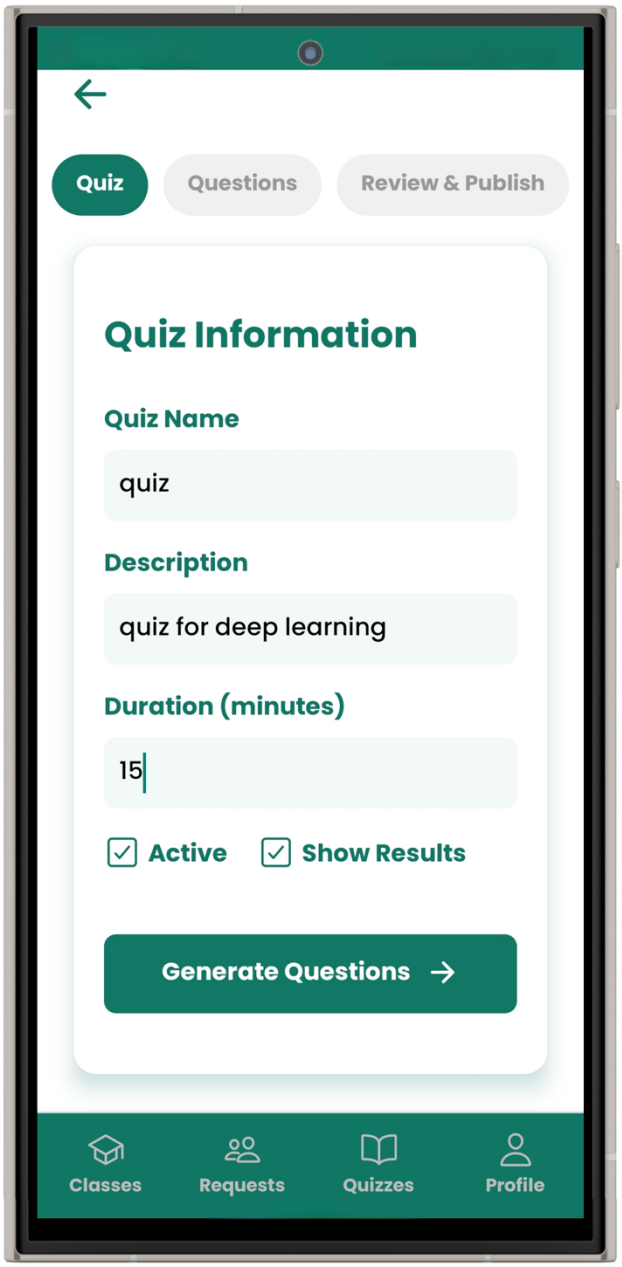


Figure 6: Completed Quiz Information Form Ready for Question Generation

4. Quiz Taking:

• Enable students to join classes using invite codes and access quizzes.

• Present questions sequentially with navigation controls and a timer, including a submission confirmation modal.

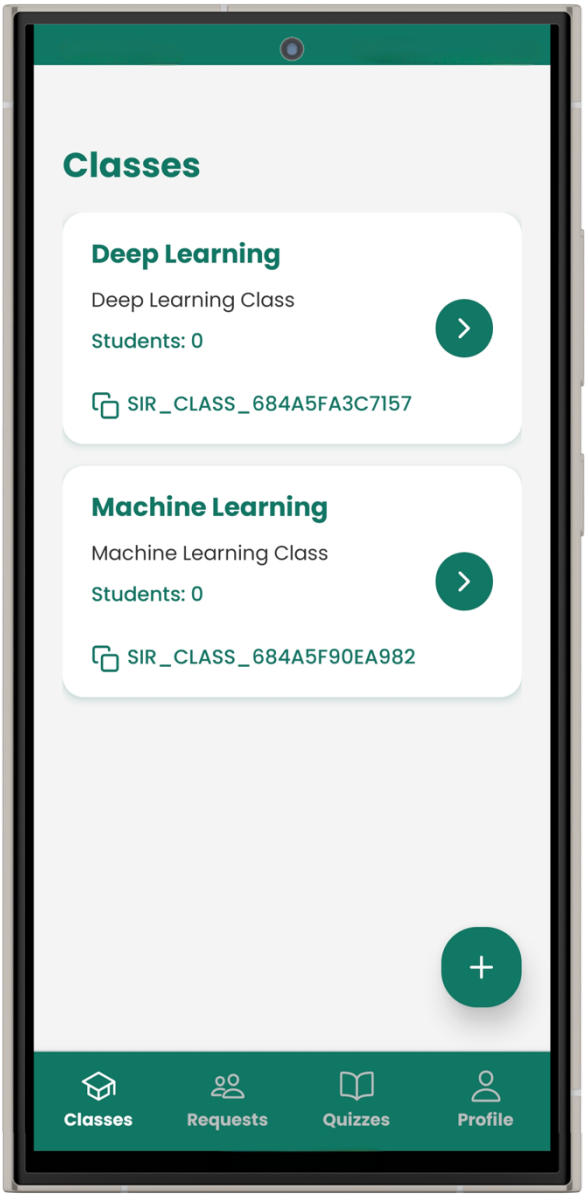


Figure 7: Student Classes Dashboard with Enrolled Classes List

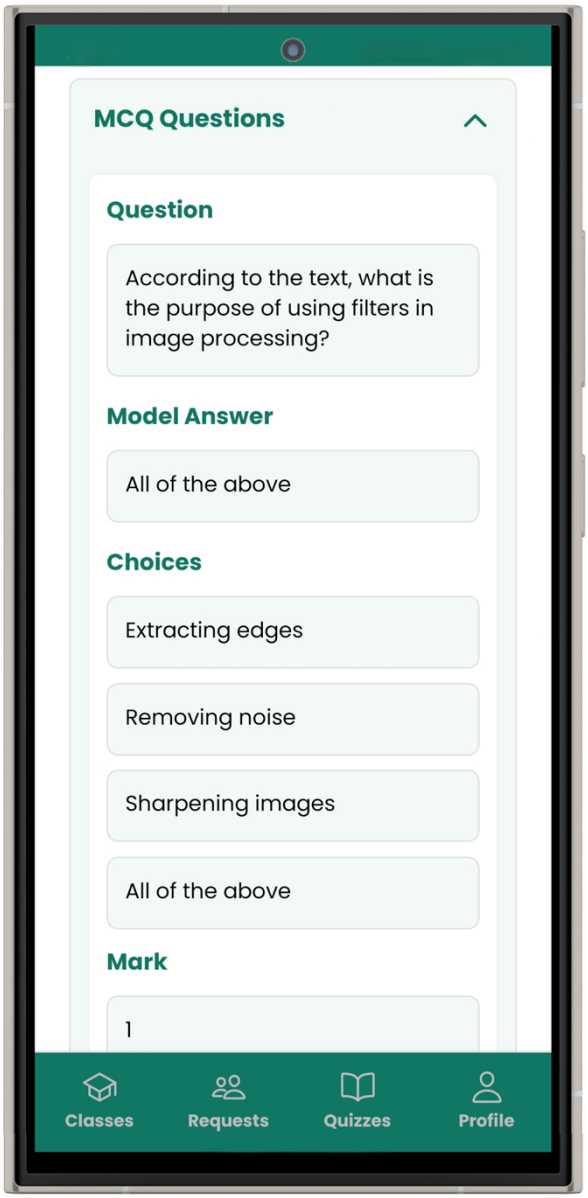


Figure 8: Student Quiz Interface Showing MCQ Question

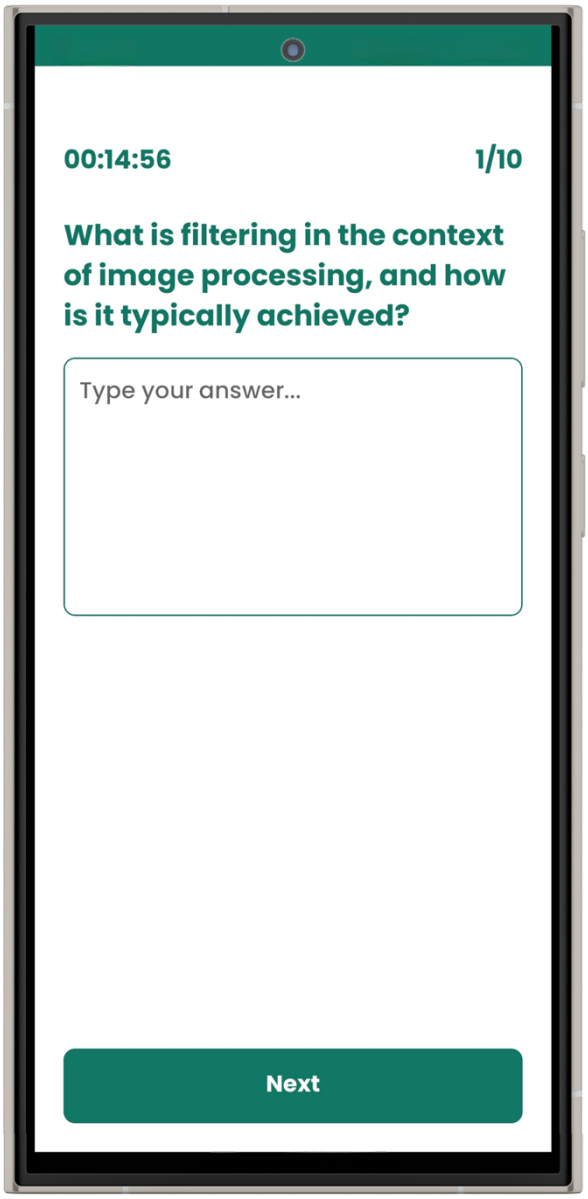


Figure 9: Student Essay Question Interface with Text Input Field

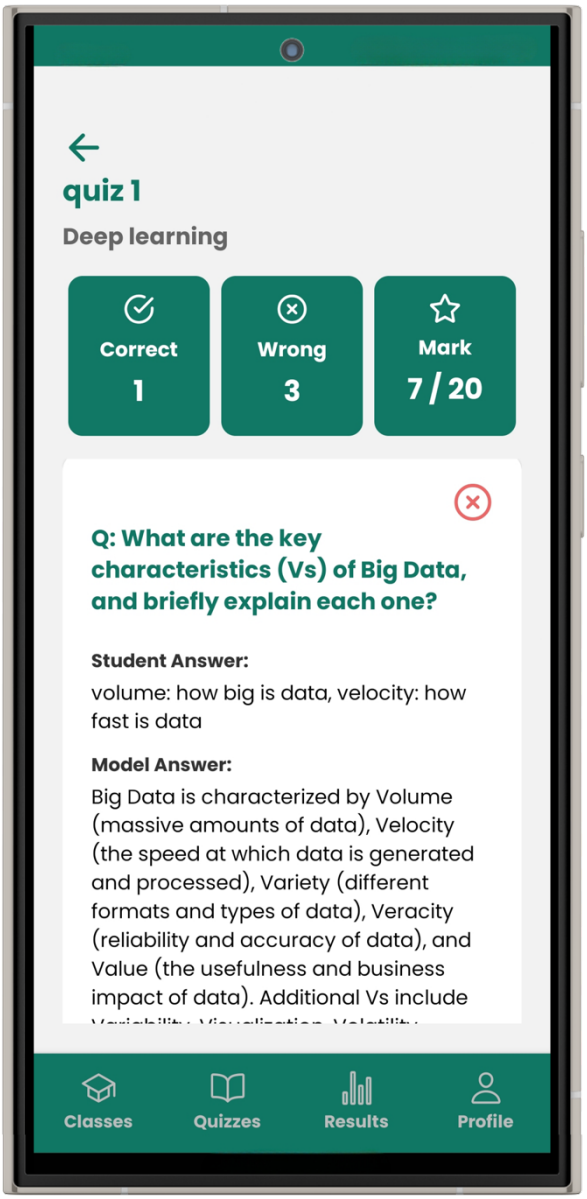


Figure 10: Student Quiz Results Page with Performance Summary and Question Review

5. Results and Analytics:

- Provide teachers with detailed performance analytics and individual student results.
- Allow students to review their results and feedback if permitted by the teacher.

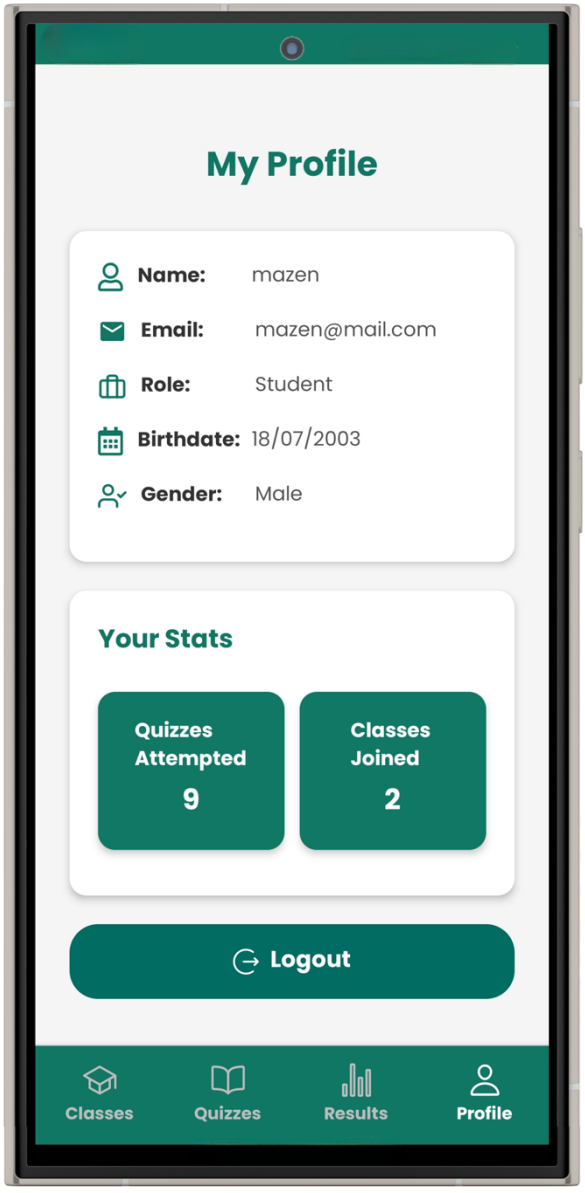


Figure 11: Student Profile Page with Personal Information and Activity Statistics

1.2.2 Non-Functional Requirements

1. Performance:

- Support concurrent users for multiple simultaneous quiz sessions.
- Ensure rapid quiz generation and grading processes.

2. Usability:

- Offer an intuitive interface for mobile and web platforms.
- Minimize the learning curve for users.

3. Security:

- Securely store and transmit user data and quiz content.
- Protect against unauthorized access and data breaches.

4. Scalability:

- Handle increasing numbers of users and quizzes without performance degradation.

5. Reliability:

- Ensure high availability with minimal downtime.
- Implement robust error handling and recovery mechanisms.

1.3 Objectives and Outcomes:

Objectives and outcomes are important for a project because they provide clarity and direction to the project team. Objectives define what the project is trying to achieve, while outcomes specify the specific results that will be delivered by the project.

1.3.1 Project Objectives

The ***SIR*** project is designed to achieve the following objectives, addressing the core challenges faced by educators and enhancing the educational experience:

• **Automate Quiz Creation**: Develop an AI-driven system that generates multiple-choice and essay questions from uploaded PDF documents, significantly reducing the time and effort required from professors.

• **Ensure Fair and Consistent Grading**: Implement AI algorithms to evaluate essay responses objectively, minimizing human bias and ensuring consistent grading stan-dards across all students.

• **Provide Timely and Detailed Feedback**: Enable immediate, AI-generated feedback on quiz performance, including specific comments on essay responses to guide stu-dent improvement.

• **Enhance User Experience**: Design an intuitive, user-friendly interface that simplifies quiz creation, administration, and participation for both professors and students.

• **Promote Scalability and Accessibility**: Build a platform capable of supporting large numbers of users and quizzes, making it suitable for institutions of varying sizes and accessible across mobile devices.

• **Foster Innovation in Teaching**: Free professors from routine assessment tasks, allowing them to explore innovative teaching methods and engage more deeply with students.

• **Support Data-Driven Instruction**: Provide analytics on student performance to help professors identify learning gaps and tailor their teaching strategies accordingly.

1.4.2 Expected Outcomes

Outcomes provide a tangible measure of the project's success. They define the specific results that need to be achieved and establish clear expectations for the project team. In the following section we will list what we expected from ***SIR*** at the end:

• **Time Efficiency for Professors**: By automating quiz creation and grading, professors can allocate more time to teaching, research, and student mentoring, enhancing their productivity and job satisfaction.

• **Improved Assessment Quality**: AI-generated questions ensure alignment with course content, while consistent grading enhances the reliability and fairness of assess-ments.

• **Enhanced Student Engagement**: Immediate feedback and interactive quiz features increase student motivation and participation, fostering a more dynamic learning environment.

• **Fair and Equitable Evaluations**: AI-driven grading reduces bias, ensuring that all students are evaluated consistently, which promotes trust and fairness in the edu-cational process.

• **Data-Driven Insights**: Detailed analytics enable professors to gain insights into student performance, allowing for targeted interventions and improved learning outcomes.

• Innovation in Education: ***SIR***'s integration of AI into assessments sets a prece-dent for modernizing educational practices, encouraging the adoption of technology-driven solutions in teaching.

• **Scalable Educational Solution**: The platform's ability to handle large user bases makes it a viable solution for institutions ranging from small schools to large uni-versities.

CHAPTER 2: Building Blocks – Tools and Technologies

2.1 Introduction

The ***SIR*** application is built upon a robust and modern technology stack meticulously selected to ensure functionality, scalability, and user satisfaction, delivering a seamless educational experience for professors and students. This chapter provides an in-depth exploration of the tools and technologies that form the foundation of ***SIR***, detailing their implementation within the system's ecosystem and the rationale behind each choice. By integrating mobile development frameworks, backend services, AI frameworks, and deployment platforms, ***SIR*** achieves its core functionalities: AI-driven quiz creation, automated grading, and intuitive user interfaces.

The system's architecture comprises a React Native mobile application that communicates with a Laravel backend, which in turn interacts with AI services hosted via FastAPI endpoints. MySQL serves as the reliable database for storing user data and quiz content, while strategic deployment approaches ensure scalability and high availability. Each component, from prototyping to deployment, is chosen to meet the project's stringent requirements, with alternative options carefully evaluated to optimize performance and development efficiency. The following sections outline the technical building blocks, the alternatives considered, and their roles in enabling ***SIR***'s innovative approach to educational assessments, accompanied by a high-level architecture diagram illustrating the system's workflow

Figure 1: High-level Architecture of the ***SIR*** Application A diagram illustrating the interaction between the React Native mobile app, Laravel backend, MySQL database, and FastAPI AI services.

2.2 Prototype

2.2.1 Prototyping in SIR: Validating AI Workflows

Prototyping is a critical phase in developing complex systems like ***SIR***, allowing the team to test and refine AI-driven functionalities before full-scale implementation. The ***SIR*** project utilized n8n, an open-source workflow automation tool, to prototype the AI workflows for quiz question generation and grading.

2.2.2 Options Considered

- **n8n**: A node-based automation platform with extensive API integration capabilities and an active open-source community.
- **Zapier**: A popular automation tool known for its user-friendly interface but limited in customization and costly for extensive use.
- **Make.com:** A visual automation platform offering flexibility but less open-source support compared to n8n.
- **Custom Scripting**: Developing bespoke automation scripts, which offer full control but require significant development time and maintenance.

2.2.3 Evaluation Criteria

The choice of prototyping tool was based on flexibility, cost, ease of integration with AI services, and community support. The tool needed to support rapid iteration and seamless connection with external APIs, such as those used for AI processing.

2.2.4 Why n8n Was Chosen

n8n was selected for its open-source nature, which eliminates licensing costs and provides access to a vibrant community for support and extensions. Its node-based workflow design allows for complex automation sequences, making it ideal for testing AI-driven question generation and grading processes. Compared to Zapier and Make.com, n8n offers greater customization and integration capabilities, crucial for prototyping ***SIR***'s unique AI workflows. Custom scripting, while flexible, was deemed too time-intensive for the prototyping phase, making n8n the optimal choice.

Feature	n8n	Zapier	Make.com	Custom Scripting
Open Source	Yes	No	No	Yes
API Inegration	Extensive	Moderate	Moderate	Full
Community Support	Active	Commercial	Commercial	Limited
Cost	Free/Tiered	Tiered	Tiered	High Development Time
Ease of use	Moderate	High	High	Low

Table 1: Comparison of Workflow Automation Tools

2.3 Artificial Intelligence

2.3.1 AI in Education: Automating Quiz Creation and Grading

Artificial intelligence is the cornerstone of ***SIR***, enabling the automation of quiz question generation from PDF documents and the grading of subjective essay responses. This functionality reduces the administrative burden on professors and ensures consistent, fair

evaluations.

2.3.2 Options for AI Frameworks

• PydanticAI with Gemini API: A Python framework for creating AI agents, paired with Google's Gemini API for advanced natural language processing.

•LangChain with OpenAI: A framework for building applications with large language models, using OpenAI's GPT models for text generation and analysis.

• Custom ML Models: Developing bespoke machine learning models tailored to ***SIR***'s needs, requiring significant expertise and resources.

2.3.3 Evaluation Criteria

The selection was based on ease of integration, performance in processing educational content, cost-effectiveness, and support for natural language tasks. The framework needed to handle PDF parsing, question generation, and essay grading efficiently while being accessible to the development team.

2.3.4 Why PydanticAI and Gemini API Were Chosen

PydanticAI was chosen for its simplicity in creating AI agents that integrate seamlessly with external APIs, reducing development complexity. The Gemini API, developed by Google, offers state-of-the-art natural language processing capabilities, particularly suited for understanding and generating educational content. Its ability to process complex texts and provide coherent outputs makes it ideal for generating quiz questions and grading essays. Compared to LangChain with OpenAI, PydanticAI and Gemini offered a more

cost-effective solution with sufficient performance for ***SIR***'s requirements. Custom ML models were ruled out due to the time and expertise required, which were not feasible within the project's constraints.

2.3.5 Implementation Details

The AI system is implemented via FastAPI endpoints, which provide scalable access to AI functionalities. When a professor uploads a PDF, the PydanticAI agent processes the document using the Gemini API to extract key concepts and generate multiple-choice and essay questions. For grading, the agent compares student responses to model answers, assigning scores and generating feedback on areas for improvement. This process ensures rapid, accurate, and consistent assessment outcomes.

Feature	PydanticAI + Gemini	LangChain+ OpenAI	Custom ML Models
Ease of Integration	High	Moderate	Low
NLP Performance	High	High	Unknown
Cost	Moderate	High	High
Development Time	Low	Moderate	High
Community Support	Growing	Strong	Limited

Table 2: Comparison of AI Frameworks for SIR

2.4 Backend

2.4.1 Backend Architecture: Powering *SIR*'s Core

The backend of the ***SIR*** application is a critical infrastructure that orchestrates secure user authentication, efficient quiz data management, and seamless interactions between the React Native mobile app and AI-driven services, ensuring a robust, scalable, and secure platform for educational assessments. It handles user registration and role-based access, stores and processes quiz questions, student responses, and analytics, while coordinating with AI services to generate questions and grade essays, maintaining data integrity and system reliability.

2.4.2 Options Considered

• **Laravel**: A PHP framework with elegant syntax and comprehensive features for

API development and authentication.

• **Django**: A Python framework known for its ”batteries-included” approach and

rapid development capabilities.

• **Express.js**: A minimalist Node.js framework offering flexibility and high performance.

2.4.3 Evaluation Criteria

The backend framework was evaluated based on team familiarity, built-in features for

authentication and API development, scalability, and ease of database integration.

2.4.4 Why Laravel Was Chosen

Laravel was selected due to the development team's expertise in PHP and the framework's robust features, including Laravel Sanctum for secure API token authentication.

Its Eloquent ORM simplifies interactions with the MySQL database, streamlining data

management. Compared to Django, Laravel offers a more intuitive syntax for the team, while Express.js, although lightweight, lacks the built-in features that Laravel provides out of the box. Laravel's ecosystem and community support further enhance its suitability for ***SIR***'s backend requirements.

2.5 Mobile Development

2.5.1 Mobile App: Delivering a Seamless User Experience

The ***SIR*** mobile application serves as the primary interface for professors and students, providing access to quiz creation, taking, and review functionalities across iOS and Android platforms.

2.5.2 Options Considered

• **React Native**: A cross-platform framework using JavaScript and React for rapid development .

• **Flutter**: A UI toolkit from Google for building natively compiled applications.

• **Native Development**: Separate codebases using Swift for iOS and Kotlin for Android, offering optimal performance but increased development time.

2.5.3 Evaluation Criteria

The choice was based on cross-platform compatibility, development speed, performance, and community support. The framework needed to support a single codebase for both platforms while delivering a native-like experience.

2.5.4 Why React Native Was Chosen

React Native was chosen for its ability to share a single codebase across iOS and Android, significantly reducing development and maintenance efforts. Tools like Expo (https://

expo.dev) streamline development, testing, and deployment, while TypeScript enhances code quality. React Native's large ecosystem and community support provide access to libraries and resources, making it ideal for ***SIR***'s requirements. Flutter, while powerful, was less familiar to the team, and native development was deemed too resource-intensive.

2.6 UI/UX

2.6.1 Designing an Intuitive Interface

The user interface and experience are critical to ***SIR***'s success, ensuring that professors and students can navigate the application effortlessly. The design process leverages industry-standard tools to create a visually appealing and functional interface.

2.6.2 Tools Used

• **Figma**: A cloud-based design tool for collaborative interface design (https://www.figma.com).

• **Adobe Photoshop**: For raster graphics editing, ideal for creating detailed visual assets.

• **Adobe Illustrator**: For vector graphics creation, suitable for scalable UI components.

2.6.3 Why These Tools Were Chosen

Figma was selected for its real-time collaboration features, allowing the design team to work together efficiently across locations. Its vector-based design capabilities are ideal for

creating scalable UI components, ensuring consistency across devices. Adobe Photoshop and Illustrator complement Figma by providing advanced tools for image editing and vector illustration, respectively, which are essential for crafting high-quality visual assets. These tools were chosen for their industry-standard status, robust feature sets, and ability to support ***SIR***'s design requirements.

2.7 Deployment

2.7.1 Deploying *SIR*: Ensuring Scalability and Reliability

Deployment strategies ensure that ***SIR***'s backend and AI services are accessible, scalable, and reliable for users worldwide.

2.72 Options Considered

- **Hostinger**: A cost-effective web hosting service for Laravel applications.
- **Vercel**: A platform optimized for serverless deployments, ideal for FastAPI applications .
- **AWS**: A comprehensive cloud platform offering extensive services but with higher Complexity.

2.7.3 Evaluation Criteria

Deployment platforms for the ***SIR*** application were meticulously assessed to ensure efficient hosting and dependable performance, with evaluations centered on cost-effectiveness, ease of use, scalability, and compatibility with the project's technologies. Cost-effectiveness prioritized affordable solutions to meet budget constraints, ensuring financial viability for an educational tool. Ease of use required intuitive deployment interfaces and streamlined workflows to simplify setup and maintenance for the development team. Scalability demanded the ability to accommodate increasing numbers of users and quiz data, supporting ***SIR***'s growth across diverse educational institutions. Compatibility focused on seamless integration with the

application's technology stack, including Laravel, React Native, and FastAPI, to ensure cohesive system operations.

2.7.4 Why Hostinger and Vercel Were Chosen

Hostinger was selected for hosting the Laravel backend due to its cost-effective pricing and straightforward deployment process tailored for PHP applications, enabling rapid setup and reliable performance. Vercel was chosen for deploying FastAPI endpoints, leveraging its seamless Git integration and automatic scaling capabilities to handle dynamic AI service demands efficiently. In contrast, AWS, despite its extensive features, was deemed overly complex and expensive for ***SIR***'s current requirements, making Hostinger and Vercel the optimal choices for balancing performance, affordability, and technical alignment.

2.8 Business Model

2.8.1 Sustaining *SIR*: Exploring Revenue Streams

To ensure long-term viability, ***SIR*** considers various business models to support ongoing development and maintenance.

2.8.2 Options Considered

• **Freemium Model**: Offer basic features for free with premium features available for a fee.

• **Subscription Model**: Provide full access through monthly or annual subscriptions.

• Institutional Licensing: Sell licenses to educational institutions for faculty and student use.

2.8.3 Why a Combined Approach Was Chosen

A combination of freemium and institutional licensing was selected to maximize accessibility and sustainability. The freemium model allows individual professors to use basic

features at no cost, encouraging adoption, while premium features like advanced analytics or additional AI capabilities generate revenue. Institutional licensing targets schools and universities, providing a steady income stream to support platform enhancements and maintenance

Chapter 3: Artificial Intelligence Agents

3.1 What are AI Agents?

An artificial intelligence (AI) agent refers to a system or program capable of autonomously performing tasks on behalf of a user or another system by designing its workflow and utilizing available tools. AI agents encompass a wide range of functionalities beyond simple natural language processing, including decision-making, problem-solving, interacting with external environments, and executing specific actions. These agents can be effectively deployed to tackle complex tasks within diverse enterprise contexts—from software design and IT automation to sophisticated code-generation tools and conversational assistants. Leveraging advanced natural language processing techniques of large language models (LLMs), AI agents interpret and respond to user inputs in a step-by-step manner, autonomously determining when and how to utilize external tools.

Another perspective describes an AI agent as a software program that interacts dynamically with its environment. It collects relevant data, processes this data autonomously, and takes independent decisions or actions toward achieving predetermined goals. Though humans initially define these objectives, the AI agent itself independently identifies and executes the optimal series of actions required to meet these goals.

For example, consider a contact-center AI agent tasked with resolving customer queries. The agent autonomously interacts with customers, asks clarifying questions, consults internal documents, and formulates effective responses. Depending on customer interactions, it decides independently whether it can resolve the issue itself or if the query needs to be escalated to a human representative. In simpler terms, you can visualize an AI agent as a virtual human assistant:

As shown, an AI agent receives input from the environment, processes information using its internal "brain" and "instructions," references memory (such as previously learned data), utilizes external tools (like APIs, calculators, databases), and finally produces a relevant and effective output.

In our ***SIR*** project specifically, an AI agent autonomously generates quizzes from PDF documents, determines appropriate questions, formulates correct and effective answers, and

then independently grades student responses based on both its stored knowledge and provided context.

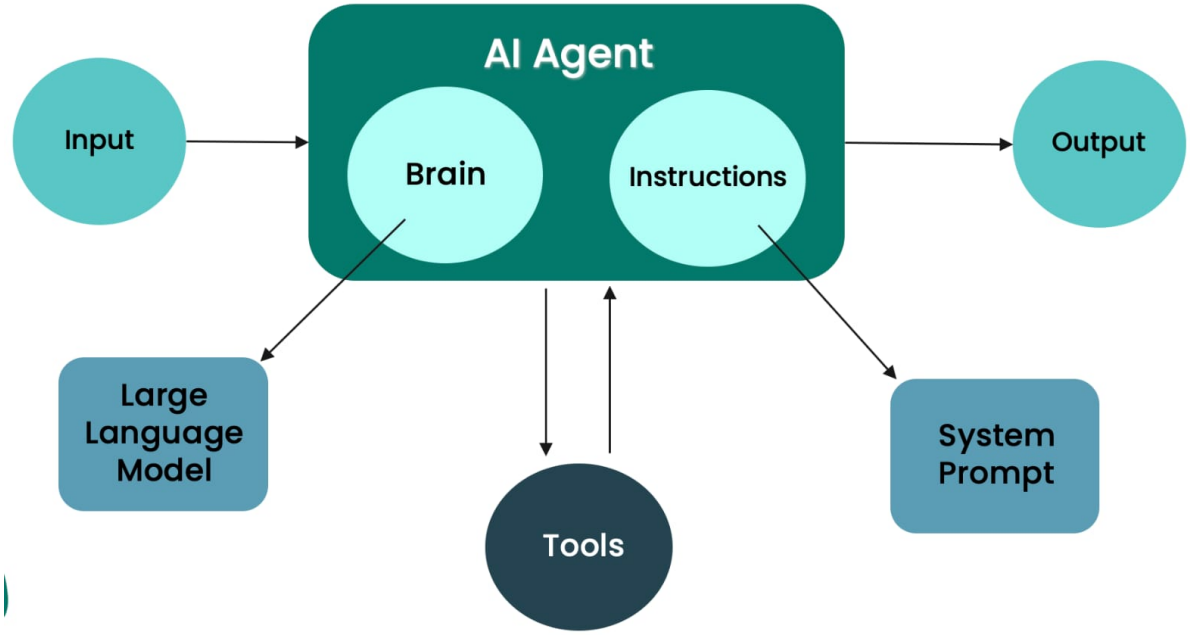


Figure 12 : AI Agent Architecture

3.1.1 Prompting AI Agents

Prompting is the critical skill of crafting explicit, clear, and specific inputs for AI agents, greatly enhancing their effectiveness. A carefully constructed prompt guides the AI clearly, improving task accuracy, relevance, and comprehensiveness.

Essential elements of effective prompts include:

- **Clarity:** Explicitly stating the goal.
- **Context:** Providing adequate background for comprehension.
- **Constraints:** Clearly defining limitations or specific rules.
- **Examples:** Guiding the AI with practical demonstrations (multi-shot prompting).

To function effectively, an AI agent requires a clearly structured prompt. The quality and structure of this prompt directly determine how well the agent understands its task and executes it. In the ***SIR*** project, prompting plays a central role in both quiz generation and grading. When a teacher uploads a PDF to generate a quiz, the prompt sent to the AI agent must be carefully crafted to ensure accurate, relevant results.

A good prompt is not just a question or command—it's a detailed instruction set that includes multiple key components. The image below outlines the **six essential components** of a well-engineered prompt:

Let's break them down in the context of our project:

1. Objective: Clearly state what the AI is expected to do. This is the high-level goal or task.

2. Context: Provide background information that helps the AI understand the environment, target audience, or situation in which it is operating.

3. Tools: List the tools or APIs the AI has access to and explain what each tool does and when to use them.

4. Instructions: Step-by-step directives or rules that guide the AI's behavior. These should cover what to include, what to avoid, and any formatting rules.

5. Output Requirements: Specify how the response should be formatted or structured. This ensures consistency and easier interpretation of results.

6. Examples: Give one or more examples to demonstrate the desired structure, tone, or level of complexity. This helps set clear expectations for the AI.

Example Prompt from the "*SIR*" Project (Quiz Generation Agent)

This is the actual structure used by the quiz generation agent when a teacher uploads a lecture PDF:

```
Objective:
You are an AI agent that generates multiple-choice and essay questions based on a given text input. The
user provides the number of MCQs and essay questions required, and you generate them accordingly.

Context:
- The agent receives a text passage and the number of MCQs and essay questions required as input.
- Each MCQ must have four answer choices, with one correct answer.
- Each essay question must have a model answer.

Tools:
Use the Gemini API to:
- Parse and summarize the uploaded PDF
- Identify core concepts and definitions
- Extract keywords to guide question formation

Use the output parser to:
- Format the output in a structured JSON format.

Instructions:
1. Read and understand the input text
2. Generate the specified number of MCQs, ensuring relevance to the text.
3. Each MCQ should include:
   - A clear question related to the text.
   - Four answer choices, one of which is correct.
   - A model answer indicating the correct choice.
4. Generete the specified number of essay questions,ensuring they require analytical or explanatory
responses based on the text.
5. Each essay question must include:
   - A clear, open-ended question related to the text.
   - A model answer that provides a well-structured response.
6. Format the output in a structured JSON format.


Output Requirements:
- Ensure MCQs are diverse and cover different aspects of the text.
- Keep essay questions open-ended but specific to the text.
- The model answer should be concise but informative.
- Maintain JSON structure integrity for easy parsing.
- Don't make up questions and answers unrelated to the input text


Examples:

Input:
  "mcq": 2,
  "essay": 2,
  "text": "A text about a subject"

Output:
  "mcq_questions": [
    {
      "question": "What is the capital of France?",
      "model_answer": "Paris",
      "choices": [
        "Paris",
        "London",
        "Berlin",
        "Madrid"
      ]
    },
    {
      "question": "Which planet is known as the Red Planet?",
      "model_answer": "Mars",
      "choices": [
        "Venus",
        "Mars",
        "Jupiter",
        "Saturn"
      ]
    }
  ],
  "essay_questions": [
    {
      "question": "Explain the main causes of climate change",
      "model_answer": "Climate change is primarily caused by human activities including..."
    },
    {
      "question": "Describe the process of photosynthesis",
      "model_answer": "Photosynthesis is the process by which plants convert sunlight..."
    }
  ]
}
```

Figure 13: PydanticAI Agent System Prompt for Question Generation

3.1.2 Types of AI Agents

Simple Reflex Agents

These agents operate based purely on immediate environmental stimuli through predefined rules. They do not retain memory or past states, limiting their effectiveness to simple, structured environments. A thermostat exemplifies a simple reflex agent, turning heating systems on or off based on immediate temperature readings.

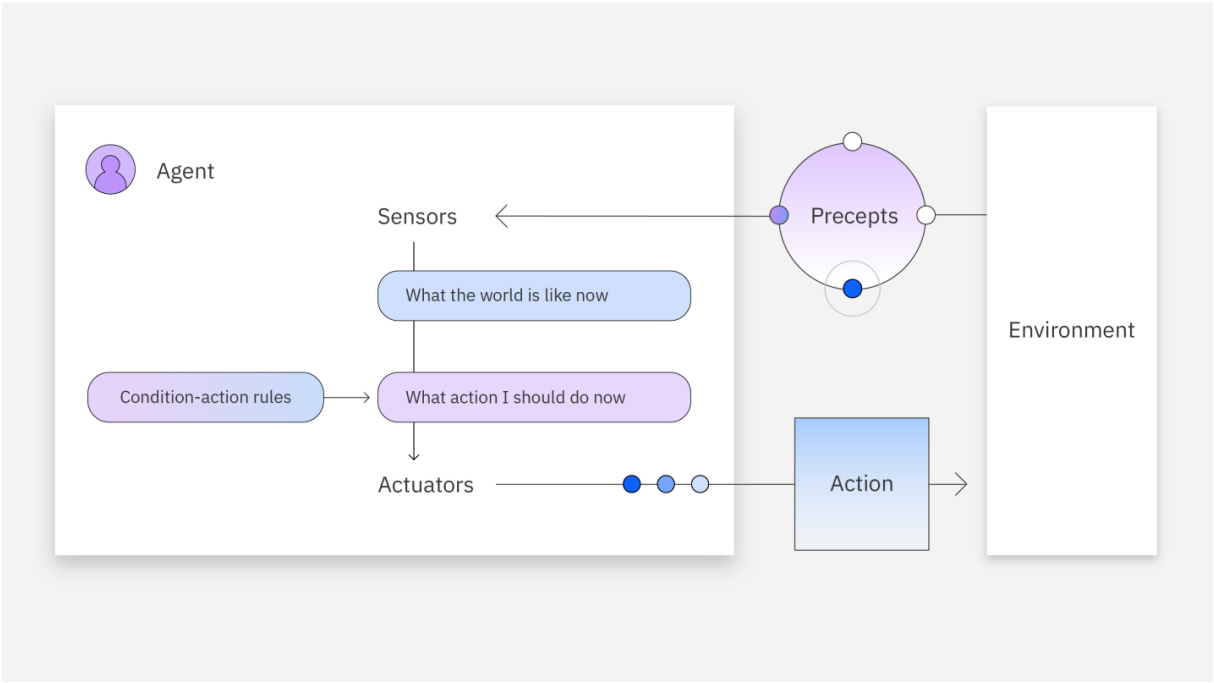


Figure 14: Simple Reflex Agent Architecture with Condition-Action Rules

Model-Based Reflex Agents

More sophisticated than simple reflex agents, these agents maintain an internal state or model of their environment. They utilize past data to inform their decisions, allowing them to handle partially observable environments. For instance, a navigation robot remembers past obstacles to make more informed decisions about future movements.

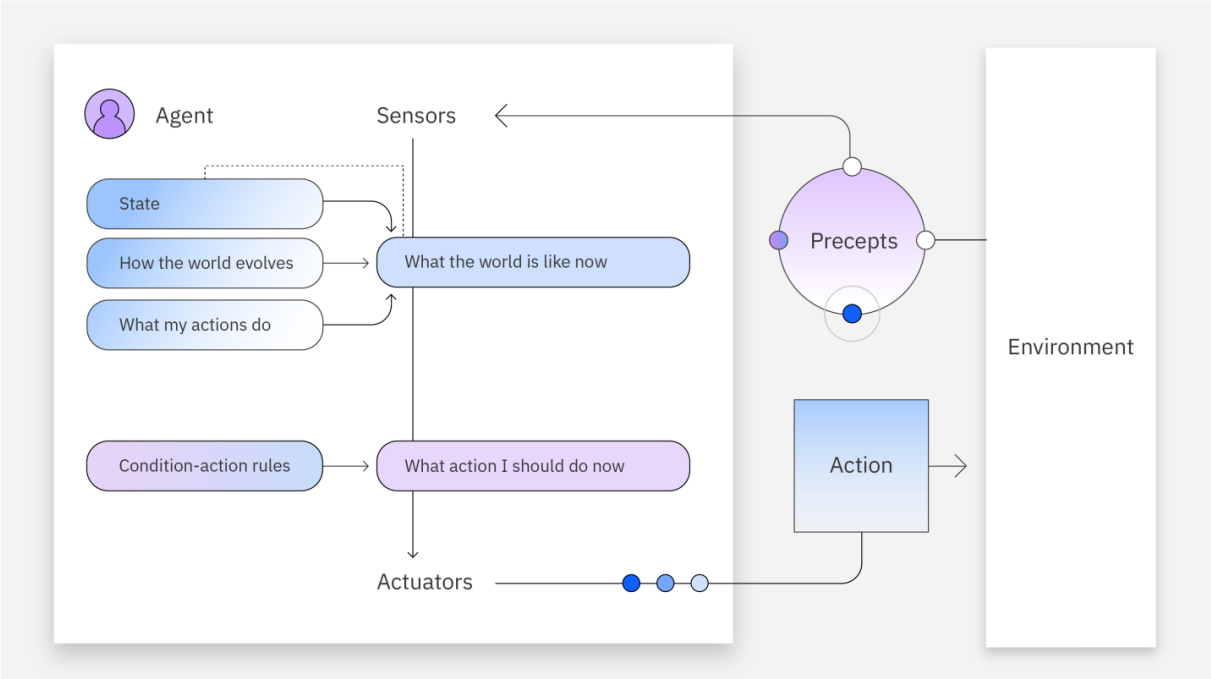


Figure 15: Model-Based Reflex Agent Architecture with Internal State Tracking

Goal-Based Agents

These agents actively consider their ultimate goals when making decisions. Instead of merely responding to immediate inputs, they plan actions strategically to reach defined objectives. For example, a delivery robot navigating to a specific destination plans its route around known obstacles, proactively working towards its goal.

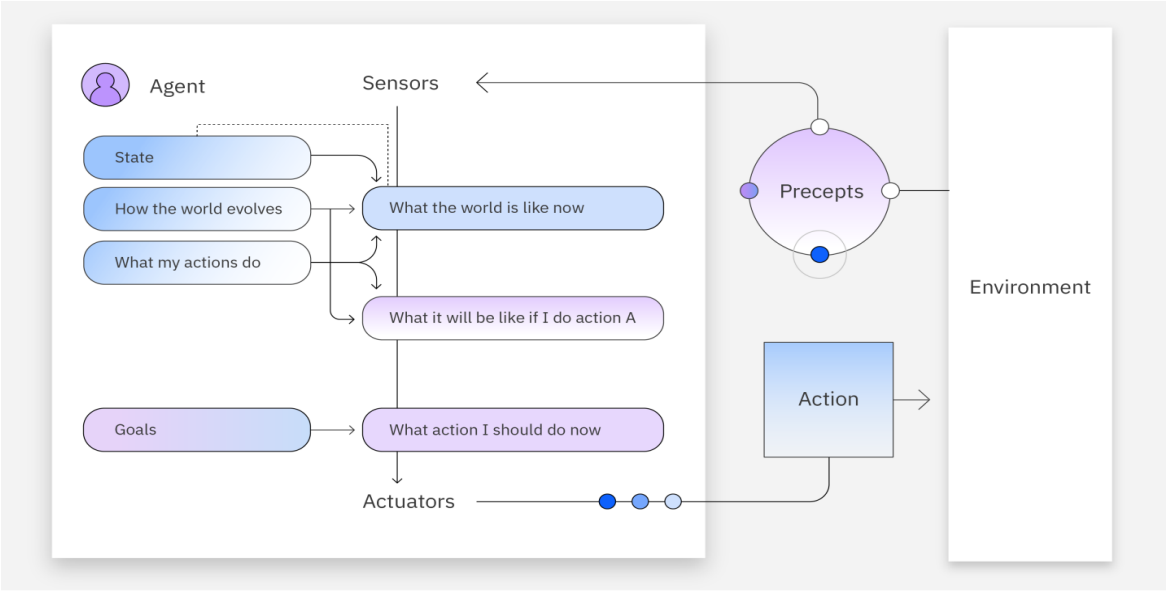


Figure 16: Goal-Based Agent Architecture with Strategic Action Planning

Utility-Based Agents

Utility-based agents extend goal-based decision-making by incorporating a utility function that evaluates multiple possible outcomes. These agents can balance competing priorities effectively. For instance, autonomous vehicles balance factors like speed, fuel efficiency, and safety to choose the optimal driving strategy.

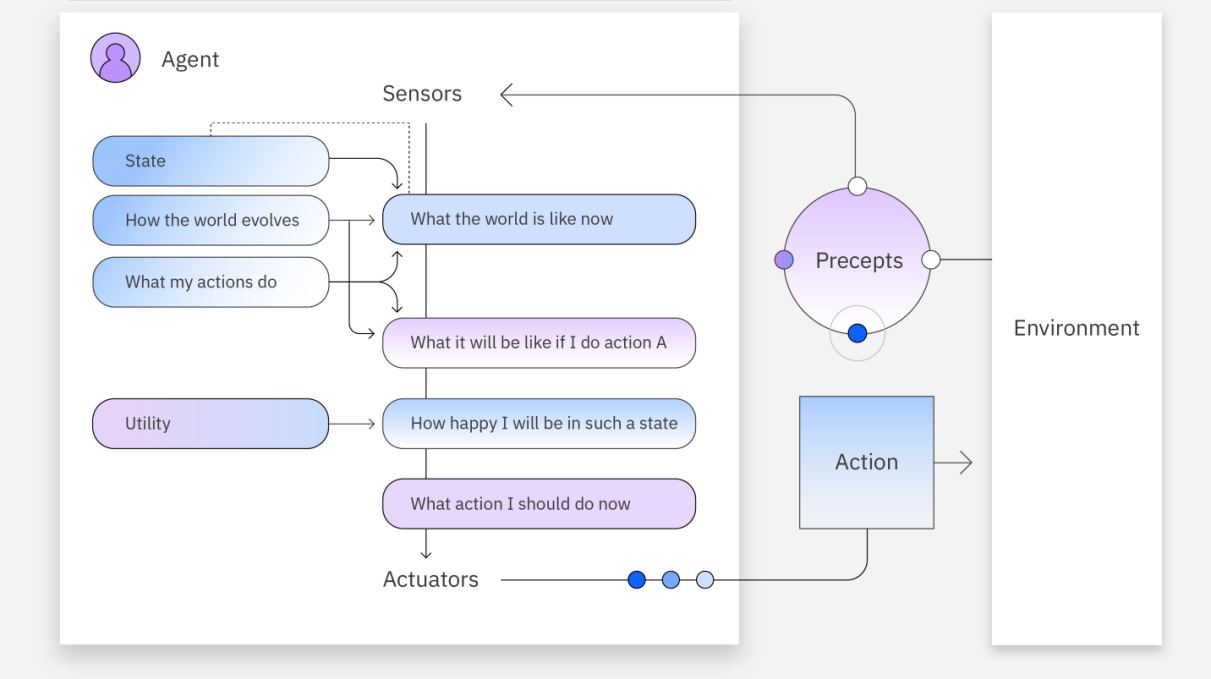


Figure 17: Utility-Based Agent Architecture with Outcome Evaluation Function

Learning Agents

Learning agents continually improve performance through experiences and feedback. They adapt behaviors over time, significantly enhancing effectiveness in dynamic environments. For instance, customer-support AI continually improves its interactions based on customer feedback and past interactions.

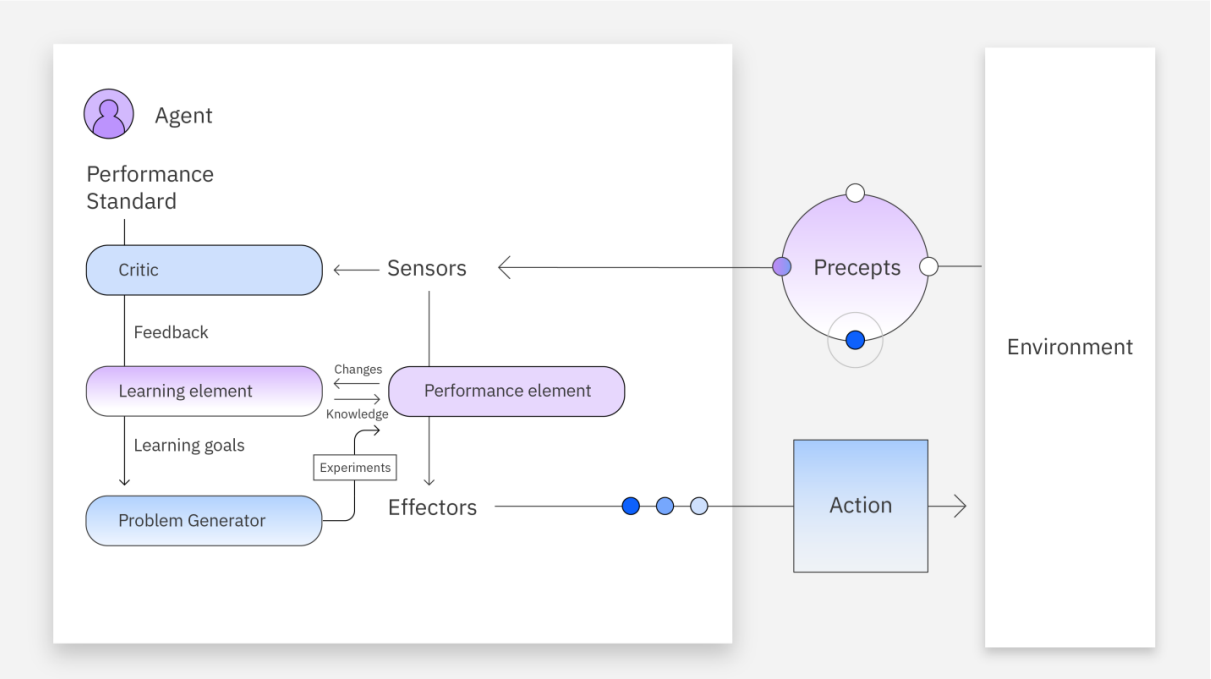


Figure 18: Learning Agent Architecture with Feedback-Based Adaptation

3.1.3 Benefits of AI Agents

Deploying AI agents offers substantial benefits to businesses:

- **Improved productivity:** Automating routine tasks frees human resources to focus on more creative and complex activities.
- **Reduced costs:** AI reduces inefficiencies, minimizes human error, and optimizes business processes.
- **Enhanced decision-making:** AI agents leverage extensive real-time data analysis to support strategic decisions rapidly.
- **Superior customer experience:** Personalized, responsive interactions improve customer satisfaction, loyalty, and retention.

In the ***SIR*** project specifically, AI agents dramatically enhance productivity by automating quiz creation and grading processes, significantly improving teacher efficiency, accuracy, and student satisfaction.

3.1.4 Risks and Limitations of AI Agents

Despite their benefits, AI agents present several challenges:

- **Data Privacy:** Collecting, managing, and storing extensive data raises privacy and security concerns.
- **Ethical Issues:** AI models risk producing biased or unfair results, necessitating regular human oversight and ethical reviews.
- **Technical Complexity:** Effective implementation requires deep expertise in AI technologies, making it challenging for many organizations.
- **Computational Costs:** Training advanced AI agents demands substantial computing resources and costly infrastructure.
- **Multi-Agent Dependencies:** Complex multi-agent systems risk failure through systemic vulnerabilities and dependencies.
- **Infinite Feedback Loops:** Without careful design, agents can become stuck in repetitive loops, necessitating active monitoring.
- **Data Governance:** Mishandling sensitive data or inadequate security measures can lead to severe consequences, emphasizing robust data management practices.

The ***SIR*** project proactively addresses these challenges by embedding rigorous data privacy practices, continuous ethical review, optimized resource management, and robust design to ensure secure, efficient, and reliable operation.

3.2 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is an advanced architecture that enhances the performance of large language models (LLMs) by integrating them with external knowledge bases. Traditional LLMs generate outputs based solely on their trained datasets, limiting responses to information available during training. RAG systems, however, enable these

models to retrieve additional context from external databases dynamically, significantly increasing response accuracy and relevance.

RAG optimizes the capabilities of generative AI models by referencing authoritative external data sources, such as internal documents, specialized datasets, or scholarly articles. It allows these models to provide accurate domain-specific responses without costly retraining.

In the context of the ***SIR*** project, RAG significantly improves quiz creation and grading by dynamically consulting educational materials (uploaded PDFs) when generating questions and grading essays, ensuring quizzes are accurate, relevant, and contextually appropriate.

3.2.1 How Does RAG Work?

The Retrieval-Augmented Generation (RAG) process involves several stages:

1. **Creating External Data**:
    - New data, distinct from the LLM's original training set, is collected from sources like APIs, databases, or document repositories.
    - This data is segmented into manageable chunks and transformed into numerical representations (embeddings) through embedding models. These embeddings are stored in a vector database, creating a rich knowledge library accessible to the AI.
2. **Retrieving Relevant Information**:
    - When users submit queries, these queries are transformed into vector embeddings.
    - The system matches query embeddings with stored document embeddings in the vector database, retrieving highly relevant documents.

3. **Augmenting the LLM Prompt**:
    - Retrieved documents provide additional context, enriching the original user prompt through sophisticated prompt-engineering techniques.
    - The enhanced prompt ensures the LLM's responses are accurate, insightful, and deeply contextualized.
4. **Keeping External Data Updated**:
    - To maintain accuracy, external data and embeddings must be periodically refreshed. Regular updates ensure the system always references the most current, relevant information.

Below is a clear diagram illustrating this process:

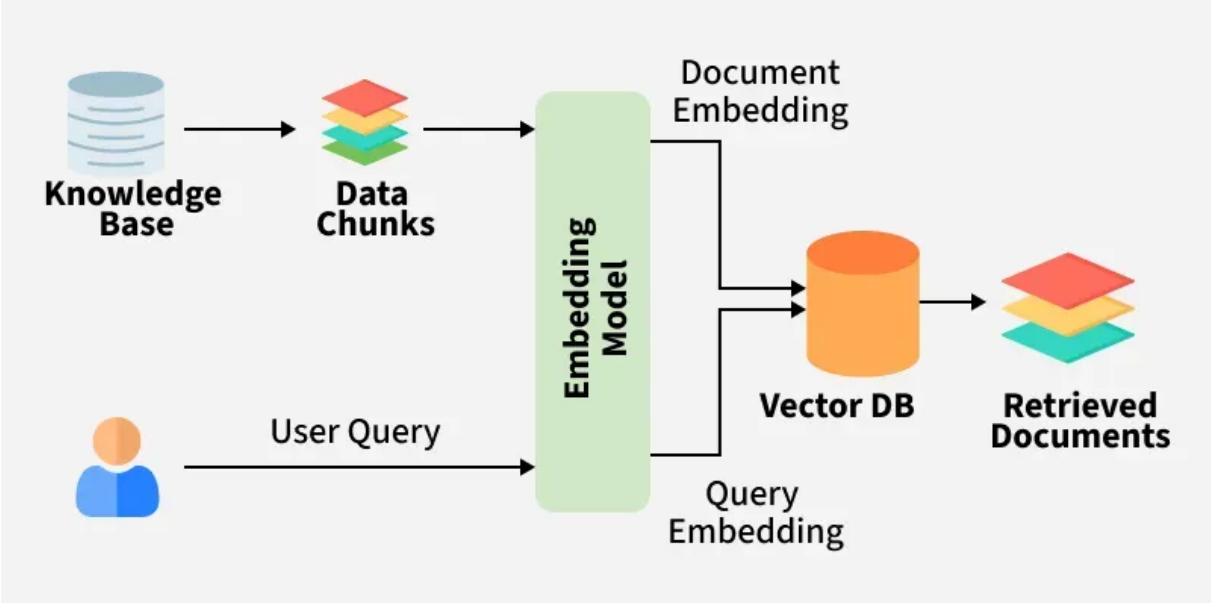


Figure 19: RAG Architecture - Document Embedding and Query Retrieval Process

In our ***SIR*** application, RAG empowers the AI agent to dynamically access and utilize educational content directly from teacher-uploaded PDFs, creating relevant and accurate quizzes that reflect precise curriculum content.

3.2.2 Benefits of RAG

Retrieval-Augmented Generation significantly enhances LLM capabilities, delivering numerous benefits:

- **Cost-Efficient AI Implementation**: Avoids expensive retraining processes by dynamically integrating external knowledge.
- **Access to Current, Domain-Specific Data**: Allows continuous updating and referencing of the latest information, enhancing accuracy and relevance.
- **Reduced AI Hallucinations**: Limits the production of incorrect or invented information by referencing authoritative external sources.
- **Increased User Trust**: Transparent referencing of authoritative sources enhances credibility and user confidence.
- **Expanded Use Cases**: Enables generative AI models to effectively support specialized contexts like education, healthcare, finance, and customer support.
- **Enhanced Developer Control and Maintenance**: Facilitates easier debugging, updating, and customization of AI-driven solutions.
- **Improved Data Security**: Enables controlled access to sensitive or confidential information.

In the ***SIR*** project, these benefits ensure accurate quiz generation and grading by consistently referencing teacher-provided learning materials, significantly increasing both teacher trust and student satisfaction.

3.2.3 Use Cases of RAG

Another RAG systems have extensive applications, effectively leveraging their advanced question-answering capabilities across various sectors:

Specialized Chatbots and Virtual Assistants: Companies deploy RAG-powered assistants to handle complex customer queries using internal product manuals, policies, or guidelines.

Research Support: Financial analysts and medical researchers use RAG to dynamically generate highly accurate, contextually relevant reports and analyses.

Content Generation: Authors, educators, and content creators utilize RAG to generate authoritative, well-referenced content efficiently.

Market Analysis and Product Development: Businesses integrate RAG to assess market trends, competitor activities, and user feedback dynamically, enabling agile and informed decision-making.

Knowledge Engines: Enterprises implement RAG-powered internal knowledge engines, assisting employees in accessing accurate and up-to-date organizational information efficiently.

Recommendation Systems: Online platforms deploy RAG to personalize user recommendations, significantly enhancing customer engagement and satisfaction.

In the ***SIR*** project, RAG specifically enables teachers to generate contextually relevant and accurate quizzes directly from classroom materials, ensuring quizzes closely align with educational objectives.

3.3 Agentic RAG

Agentic RAG represents a sophisticated evolution of traditional RAG systems, integrating advanced AI agents directly within the RAG pipeline. This integration enables complex orchestration of knowledge retrieval, verification, and contextual reasoning, substantially enhancing system flexibility, accuracy, and intelligence.

3.3.1 How Does Agentic RAG Work?

In Agentic RAG, AI agents handle key processes in the RAG workflow, including:

- Deciding whether information retrieval is necessary.
- Determining the most suitable retrieval tools (vector databases, web searches, APIs).
- Formulating effective queries dynamically.

- Critically evaluating retrieved data and validating its relevance and accuracy.

Agentic RAG architectures can be organized into:

- **Single-Agent Systems**: Simple router-based architectures, where a single agent intelligently decides between various retrieval options and tools.
- **Multi-Agent Systems**: Advanced orchestration involving multiple specialized retrieval agents. Each agent focuses on specific data sources or tools, coordinated by a central master agent.

In multi-agent setups, different agents specialize in:

- Internal proprietary data retrieval.
- Public information via web searches.
- Real-time interaction through APIs (e.g., email or chat programs).

3.3.2 Benefits and Limitations of Agentic RAG

Benefits of Agentic RAG

- **Enhanced Quality**: Improves data retrieval accuracy through intelligent, agent-based verification.
- **Greater Flexibility**: Enables dynamic selection and orchestration of multiple retrieval tools and information sources.
- **Improved Autonomy**: Allows the system to independently handle complex retrieval and decision-making tasks, minimizing human intervention.
- **Superior User Experience**: Provides highly relevant and authoritative responses, increasing user trust and engagement.

Limitations of Agentic RAG

- **Increased Complexity**: Higher complexity in system orchestration and maintenance.

- **Computational Overheads**: Additional resources required for sophisticated agent-based processing.
- **Dependence on Agent Reasoning**: Potential for incorrect agent decisions necessitating careful monitoring and fallback strategies.

3.3.3 Agentic RAG in the "*SIR*" Project

In our ***SIR*** application, the Agentic RAG implementation is specifically tailored to manage complex educational content dynamically:

- **PDF Parsing and Querying**: An intelligent agent dynamically processes educational PDFs uploaded by teachers, extracting relevant educational concepts accurately.
- **Dynamic Question and Answer Generation**: Agents autonomously evaluate retrieved data, ensure its alignment with educational goals, and generate quizzes directly referencing authoritative sources.
- **Grading Essays with Precision**: Agents utilize advanced reasoning to critically evaluate student essays, referencing original source material dynamically and providing precise, insightful feedback.

Agentic RAG significantly enhances the ***SIR*** project's reliability, accuracy, and educational effectiveness, directly benefiting teachers and students alike by ensuring quizzes reflect accurate, authoritative, and highly relevant educational content.

CHAPTER 4: Prototyping with Node-Based Workflow Automation

4.1 The Rise of No-Code Platforms and n8n

The emergence of no-code and low-code platforms has revolutionized how developers and non-technical users approach automation and rapid prototyping. These platforms democratize software development by providing visual interfaces that eliminate the need for extensive coding knowledge, enabling faster iteration and testing of complex workflows.

n8n, pronounced as "n-eight-n", is a powerful open-source workflow automation platform that exemplifies this no-code revolution. It enables users to automate processes by connecting different applications, APIs, and custom code through a visual, node-based interface.

In its simplest form, n8n allows you to create workflows using building blocks called nodes, where each node performs a specific task such as fetching data, transforming it, or interacting with APIs. These nodes can be connected visually on a canvas, similar to connecting Lego pieces to build complex automation from basic elements.

What makes n8n particularly valuable for prototyping is its low-code environment that removes traditional automation barriers while still retaining the flexibility to add custom JavaScript logic or interface with third-party systems when needed.

For our ***SIR*** project, which leverages AI to automate quiz generation and grading, n8n became an indispensable prototyping tool. It enabled us to quickly develop, test, and refine AI agent workflows without repeatedly redeploying backend infrastructure, allowing us to focus on logic and user experience rather than technical implementation details..

4.2 Why Using n8n?

Selecting the right automation and prototyping platform was a critical decision for ***SIR***, and after researching alternatives like Zapier and Make.com, it was clear that **n8n offered the flexibility, transparency, and developer control** we needed.

Flexibility and Customization: n8n is open-source and source-available, which gave us the freedom to self-host, inspect the codebase, and extend functionality if needed. For an AI-driven

educational tool like ours—where each workflow had to process PDFs, extract text, chunk it for embeddings, pass it through a vector database, and communicate with a Gemini-powered agent—we needed the ability to **insert custom logic, memory handling, and AI communication mid-flow**. With n8n, we could do all of this using a mix of pre-built and custom nodes.

Data Ownership and Privacy: Working with sensitive academic content and student responses means **data privacy is not optional**—it's essential. While cloud platforms like Zapier offer convenience, they take control of data handling and storage. By self-hosting n8n, we retained full control over the data flow, ensuring compliance with our own privacy standards and offering peace of mind to institutions adopting ***SIR***.

Community and Ecosystem: The n8n community is vibrant, knowledgeable, and constantly growing. As we started working on complex workflows involving AI agents, we turned to community nodes, blog posts, and tutorials. Whether we needed a Supabase integration or guidance on calling Google APIs securely, there was a tutorial, forum post, or GitHub repo available. The community is not just a support group—it’s a knowledge base and inspiration hub for workflow builders.

Cost-Effectiveness: For a growing project like ours, budget is a real constraint. Zapier’s and Make.com's plans are priced based on volume—number of tasks or operations. That becomes expensive very quickly when you’re dealing with AI agent calls, vector searches, and feedback generation. n8n offers **free self-hosting**, with paid plans only if you want a managed version. This meant we could allocate our time and resources to building better AI agents rather than maintaining expensive subscriptions.

Templates and Rapid Prototyping: n8n includes over 1,250 ready-made templates for various workflows, which drastically reduced our build time. We started by studying existing workflows related to file handling, chat APIs, and webhook triggers before tailoring them to our project’s specific needs.

4.3 Comparing n8n to Zapier and Make.com

Choosing the right workflow automation tool is not just a matter of preference—it can define your product's development speed, flexibility, and long-term sustainability. While Zapier and Make.com are dominant names in the automation ecosystem, n8n stands apart for developers and teams building AI-native, deeply customizable workflows—like we did with ***SIR***.

Feature	n8n	Zapier	Make.com
Flexibility	High. Supports custom JS, external APIs, logic	Medium. Limited to built-in actions	High. Supports routers, complex flows
Hosting Options	Self-host or cloud	Cloud-only	Cloud-only
Customization	Full (custom nodes, code, vector DBs, etc.)	Limited	Moderate
App Integrations	300+ prebuilt, plus community and custom nodes	5,000+	1,500+
Cost	Free for self-hosting. Paid plans start ~$20/mo	Free tier, paid from ~$19.99/mo	Free tier, paid from ~$9/mo

Table 3: Comparison of Workflow Automation Platforms

Here's how these platforms compare across the five most important dimensions that mattered to us:

4.3.1 Flexibility

When it comes to flexibility, **n8n reigns supreme**. Its source-available and developer-first approach means you can customize every part of a workflow—from how data is formatted to what kind of logic is executed between steps. With n8n, we were able to embed JavaScript

code inside nodes, run custom API requests to Gemini, and even chain memory objects and vector store interactions—something Zapier and Make.com struggle with or outright restrict.

Zapier excels in simplicity but is relatively rigid. It's designed for simple "if-this-then-that" automation flows and doesn't offer much room for tweaking the logic beyond what their UI allows. Loops, branching logic, error handling—all of these are limited and locked behind more expensive plans or simply not possible.

Make.com does offer more complexity than Zapier. Its visual interface includes routers, iterators, and error handlers, which allow you to build workflows with branching logic. However, it still doesn't allow arbitrary code execution natively, nor does it support the kind of custom memory and agent-based behavior that n8n empowers.

For us, building AI agents for quiz creation and grading required **multi-step reasoning**, use of external data (PDF parsing and vector retrieval), and **real-time logic based on user input and memory state**—something only n8n could truly handle.

4.3.2 Hosting Options and Data Ownership

One of the main reasons we selected n8n was the ability to **self-host**. In a project like ***SIR***, where we're processing educational content and student answers, data privacy and ownership are paramount. With n8n, we were able to deploy our own secure instance on private servers and have **complete control over data flow, storage, and retention policies**.

Zapier and **Make.com** are cloud-only platforms. Everything you build and automate runs through their infrastructure. While they have strong security protocols, there's always a tradeoff between convenience and control. If the use case demands GDPR compliance, offline access, or self-managed deployments, these platforms fall short.

In contrast, **n8n's open-source model allows for enterprise-grade hosting on your own terms**. You can run it in Docker, on a Linux server, or even scale it using Kubernetes. This was a non-negotiable for us and ultimately ensured that ***SIR*** could comply with privacy-first deployments for institutions that needed them.

4.3.3 Customization and Developer Control

n8n is not just an automation tool—it's an **automation framework**. The ability to insert JavaScript via **Code nodes**, manipulate data dynamically, and build modular sub-flows gave us developer-level control over every detail of our AI agents. We created workflows that not only handled raw user input but also dynamically queried Supabase vector stores and built prompts for Google Gemini LLMs.

Zapier offers little to no developer customization. You can use built-in filters and logic paths, but you cannot inject real code unless you use their premium "Code by Zapier" module—and even then, it's extremely limited.

Make.com provides more flexibility than Zapier in terms of connecting apps and building conditional paths, but it still lacks **programmable memory, agent management, and flexible embedding/LLM integrations**. You're confined to their drag-and-drop UI and predefined operations.

We needed workflows that could emulate reasoning, loop over datasets, handle failures, and make **contextual decisions based on memory and vector similarity**. With n8n, we were free to build exactly that.

4.3.4 Integration Capabilities

All three platforms offer strong integration libraries—but with very different philosophies.

Zapier shines in this area with over **5,000+ prebuilt app integrations**, giving you immediate access to nearly every major SaaS platform out there. It's an ideal solution for quickly connecting mainstream apps like Gmail, Trello, Google Docs, and Slack.

Make.com also supports over **1,500 integrations**, offering most of the essential connectors businesses need. It allows slightly more control than Zapier over how those integrations behave, especially when it comes to handling input/output formatting and processing large volumes.

n8n offers around **300+ official integrations**, but what it lacks in quantity, it makes up for in power. n8n supports custom HTTP Request nodes, OAuth2 authentication, API key-based connections, and most importantly—**custom nodes**. The community is constantly building

integrations that can be added manually or reused. In our case, we used the **Supabase community node**, Gemini via HTTP nodes, and custom memory management logic.

Even more critically, n8n allowed us to combine **AI workflows** with **third-party APIs** in a seamless, visual way—which no other platform provided.

4.3.5 Cost and Scalability

Finally, cost was an essential factor. Zapier and Make.com both offer **free plans**—but with serious limitations on task counts, app access, and update frequency.

Zapier becomes expensive fast. Once you exceed a few thousand tasks per month or need multi-step flows, pricing jumps significantly. For AI workflows that involve querying models, formatting JSON, storing data, and more—it's easy to burn through your limits.

Make.com is more affordable, especially at the lower end. Paid plans start at a lower price than Zapier and offer more generous operation limits. Still, the more complex your automation becomes, the more quickly you'll hit thresholds.

n8n, on the other hand, is **free to self-host**. For us, this was an enormous advantage. We could build, test, and deploy production-grade AI agents without worrying about usage caps. If you prefer not to manage infrastructure, n8n's cloud plans are reasonably priced and scale predictably.

In the long term, **n8n offers the best price-performance ratio**, particularly for projects like ***SIR*** where each request involves LLM calls, data transformation, and multi-service integration.

4.4 How Does n8n Work?

n8n operates on a powerful concept of **nodes**, which are modular, configurable blocks that form the backbone of any workflow. In our AI-driven quiz app ***SIR***, nodes enabled us to build workflows that processed PDFs, embedded content into vector databases, prompted LLMs, and returned structured outputs—all visually, and without having to spin up backend services for each task.

We organized our workflows around **three main types of nodes** in n8n: **Trigger Nodes**, **Processing (Core) Nodes**, and **Cluster Nodes**. Each plays a distinct role in orchestrating a smooth and intelligent automation process.

4.4.1 Trigger Nodes

Trigger nodes are the **starting point** of any n8n workflow. They define the conditions under which the workflow is launched. In ***SIR***, we used trigger nodes to initiate workflows when the mobile app made HTTP requests to generate or grade quizzes.

Here are a few examples of common trigger nodes:

- **Webhook Trigger**: This is the primary trigger node we used in both of our workflows. It listens for incoming HTTP POST requests and starts the workflow execution. For example, when a teacher uploads a PDF to generate a quiz or a student submits answers for grading, the mobile app sends data to this node to kick off the process.
- **Execute Workflow Trigger**: Useful for testing workflows , we may chain workflows—one for embedding text, another for generating questions—and this trigger can execute one from another.
- **Gmail Trigger**: Though not used in ***SIR***, this node illustrates n8n's integration power. It activates a workflow when a new email arrives in a Gmail inbox—useful for applications involving notifications, support requests, or report parsing.

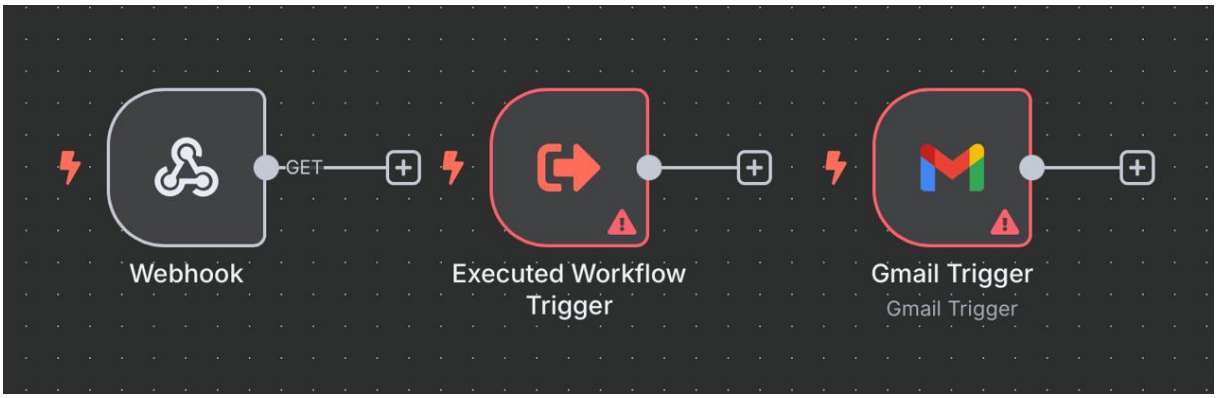


Figure 20: n8n Workflow Trigger Nodes

4.4.2 Processing (Core) Nodes

Core nodes handle the **business logic, data transformation, and interaction** within the workflow. These are the functional muscles of n8n and do the heavy lifting after a trigger has fired.

In the ***SIR*** application, several core nodes played critical roles:

- **Extract PDF Text Node**: This node extracts readable text from an uploaded PDF. It's essential for our AI agent to generate contextually relevant questions. When a teacher uploads a document, this node converts it into raw text that can be processed and embedded.
- **Edit Fields (Set) Node**: This node is a powerful utility for formatting and structuring data. In our workflow, it helped take values such as the number of MCQs, essay questions, and parsed text, and convert them into structured objects that the AI agent could easily interpret.
- **Code Node**: One of the most flexible and essential tools for advanced workflows. The Code node lets you write custom JavaScript logic to manipulate data. In ***SIR***, we used this to:
    - Format AI-generated questions into JSON.
    - Clean and transform input data.
    - Customize the output grading feedback before returning it to the app.

Processing nodes allow a workflow to evolve beyond simple "move data from A to B" and into real intelligent automation.

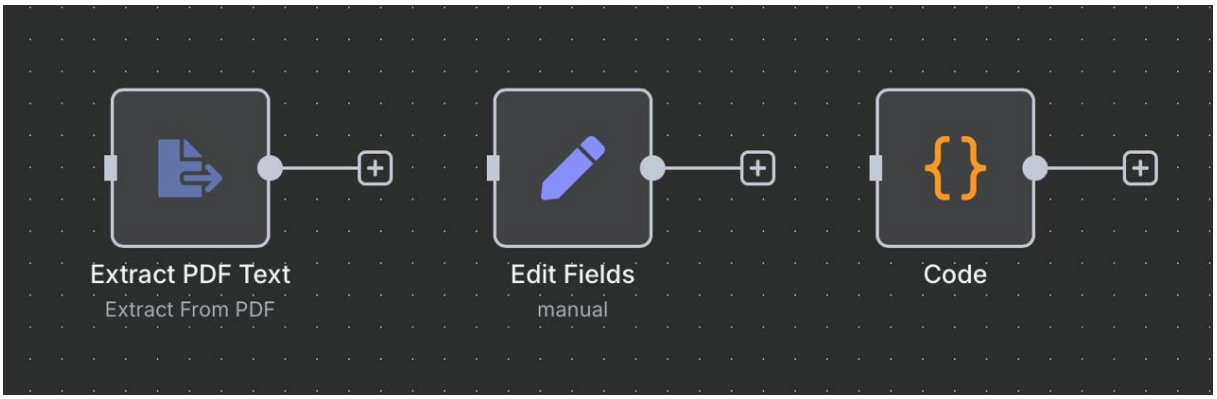


Figure 21: n8n Data Processing Nodes

4.4.3 Cluster Nodes

Cluster nodes are unique to n8n's architecture and offer **modular functionality grouped under a root node**. Instead of a single node handling everything, a cluster comprises:

- A **root node**, which starts a specific process,
- One or more **sub-nodes** that expand or customize its behavior.

These clusters function almost like **plug-and-play modules**, where you can enhance functionality by adding sub-components. They allow workflows to remain **clean and scalable**, while still being powerful. This modular structure will let us reuse logical groups across multiple workflows and simplify maintenance.

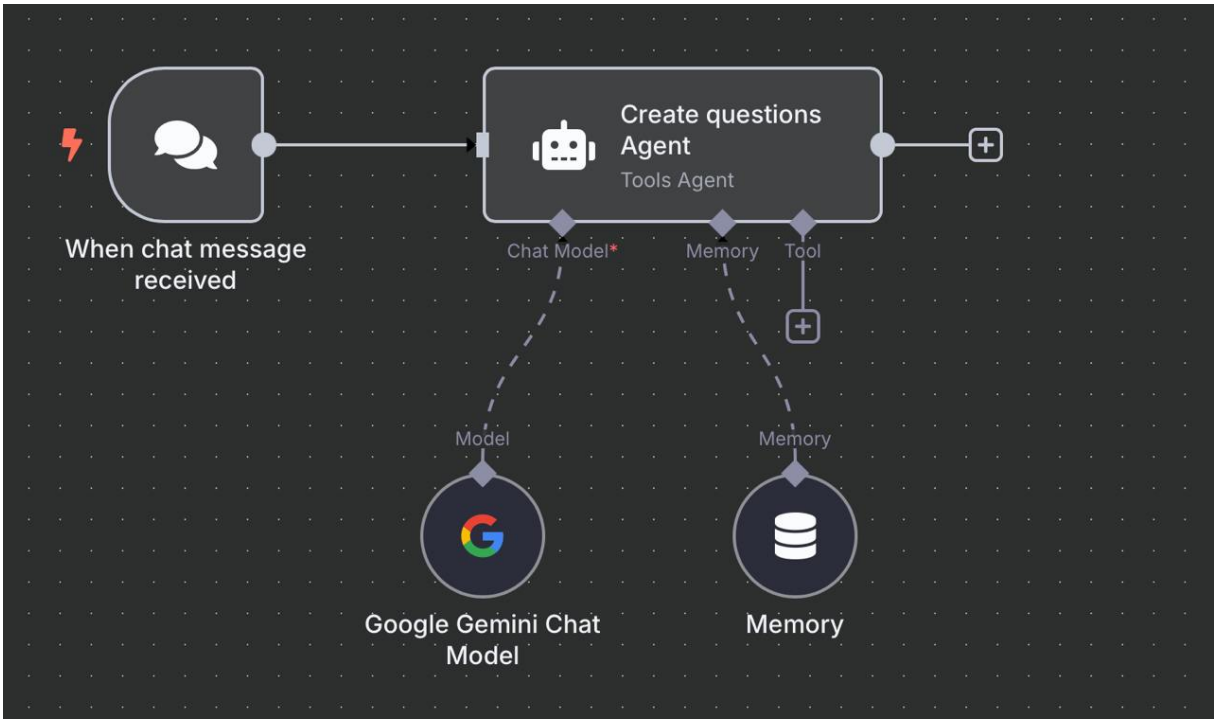


Figure 22 : n8n AI Agent Workflow for Question Generation

4.5 Prototyping AI Agents Using n8n

In the development of ***SIR***, the most groundbreaking feature we built was the AI-powered quiz engine. This involved two intelligent systems:

1. An **AI agent that generates questions from a PDF**, and
2. An **AI agent that grades student answers and provides feedback**.

Both agents were prototyped and using **n8n**, which allowed us to visually orchestrate complex, multi-step workflows involving AI models, vector databases, memory components, and structured responses—all without writing a traditional backend orchestration layer.

We'll walk through each AI workflow in detail and explain how n8n enabled this with modular logic, clarity, and efficiency.

4.5.1 Workflow 1: Question Creation Agent

This workflow is responsible for taking a PDF document (such as class notes or slides) and generating a specified number of multiple-choice and essay questions using AI.

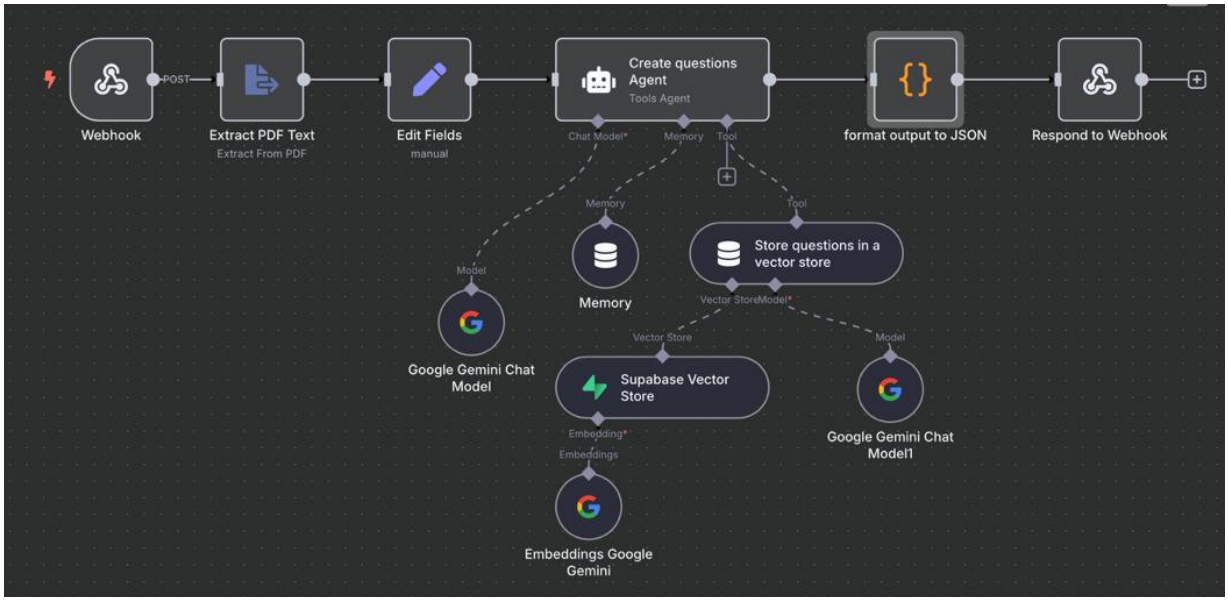


Figure 23: n8n Workflow for AI-Powered Quiz Question Generation from PDF

Workflow Breakdown:

1. Webhook Node (Trigger Node): The process begins when a POST request is made from the mobile app. The request contains:

- A PDF document (uploaded or as a URL),
- The number of MCQ and essay questions the teacher wishes to generate.

This node acts as the entry point for the entire workflow.

2. Extract PDF Text Node (Core Node): This node parses the uploaded PDF and converts its contents into plain text. This textual data becomes the basis for both question generation and future grading via vector embedding.

3. Edit Fields Node (Core Node): Here, we reformat the input by organizing it into a structured object that includes:

- Extracted text,
- The number of each type of question,

This structure ensures that the AI agent receives clear and consistent input.

4. Create Questions Agent Node (AI Tools Agent): This node acts as the central brain of the workflow. It performs the following tasks:

- Prompts the **Google Gemini 2.0 Flash Model** with a system message to generate a defined number of MCQs and essay questions from the extracted text.
- Leverages internal memory to maintain prompt context during execution.
- Sends the generated questions to the next stage of the pipeline.

5. Embedding and Vector Storage Cluster: This is a critical cluster that supports future grading operations. While the questions are being generated, a parallel flow performs the following:

- Chunks the extracted text into manageable pieces.
- Sends the chunks to the **Google Gemini Embedding Model** to create vector representations.
- Stores these vectors in the **Supabase Vector Store** for later retrieval and similarity matching.

6. Format Output to JSON (Code Node): The AI's raw output is passed to a Code Node where it is formatted into a structured JSON object. This object includes:

- Each question,
- The correct answer (for MCQs),
- The model answer (for essay questions),

7. Respond to Webhook (Final Output Node): Finally, the formatted quiz is returned to the mobile app, allowing the teacher to preview and publish it for students.

4.5.2 Workflow 2: Grading Agent

This workflow grades student responses, especially essay-type questions, by comparing them to model answers and relevant context from the source PDF.

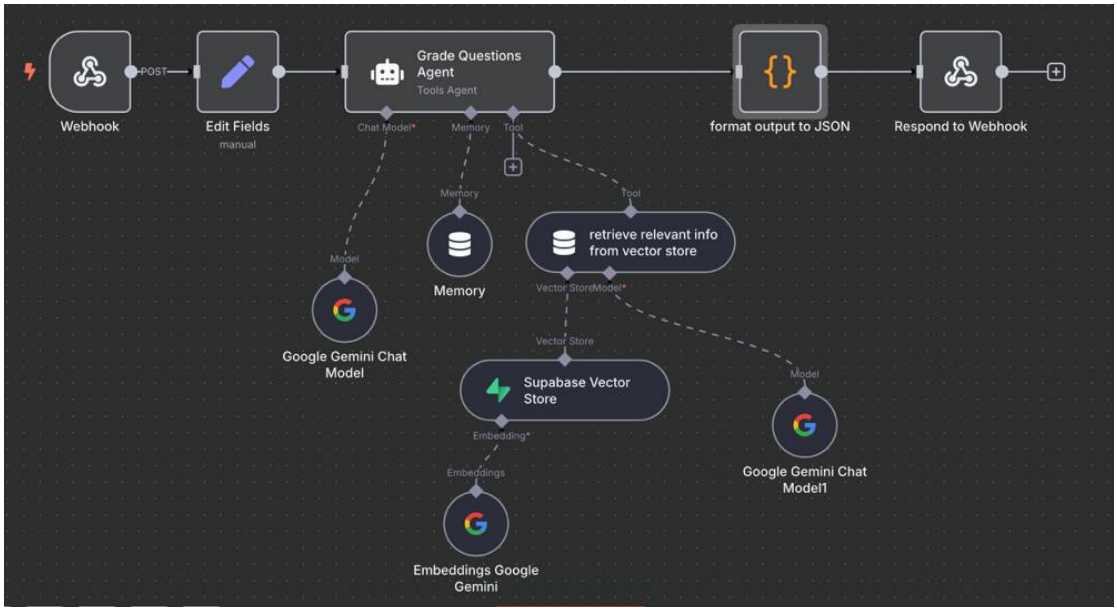


Figure 24: n8n Workflow for AI-Powered Quiz Question Grading

Workflow Breakdown:

1. Webhook Node (Trigger Node): The mobile app sends a POST request when a student submits their quiz. This request includes:

- The student's answers,
- The corresponding model answers,

This triggers the grading workflow.

2. Edit Fields Node (Core Node): Here, the incoming data is processed and structured so that the grading agent can interpret it efficiently. It maps each answer to its corresponding question and model answer, organizing them into a consistent structure.

3. Grade Questions Agent Node (AI Tools Agent): This agent handles the evaluation logic. It:

- Sends the student answer, model answer, and question prompt to the **Google Gemini Chat Model**.
- Uses memory and structured system prompts to evaluate correctness, completeness, and conceptual accuracy.
- Assigns marks and produces feedback text for each question.

4. Retrieve Relevant Info from Vector Store (Cluster Node): To ensure context-aware evaluation, this node:

- Sends a query to the **Supabase Vector Store**.
- Retrieves the most semantically relevant sections from the original class material (i.e., the PDF used to generate the questions).
- Feeds this context back to the grading agent, enhancing its understanding and judgment.

5. Memory Nodes: These nodes help the grading agent keep track of prior outputs or instructions, ensuring consistency in evaluation criteria across multiple answers or students.

6. Format Output to JSON (Code Node): After the questions are graded and feedback generated, the output is formatted into structured JSON. This includes:

- Question ID and student response,
- Mark assigned,
- AI-generated feedback.

7. Respond to Webhook (Final Output Node): The final result is sent back to the app, where the student or teacher can view:

- The score
- Detailed feedback.

CHAPTER 5: Implementing Production-Ready AI-Agents

5.1 Transitioning from Prototype to Production

As our ***SIR*** application matured from prototype to production, we needed a way to move beyond visual workflows and build **stable, type-safe, and production-ready AI agents**. While n8n allowed us to rapidly prototype our **Create Questions** and **Grade Questions** agents visually, it lacked the type enforcement and runtime validation needed for long-term maintainability. This is where **PydanticAI** came in.

PydanticAI is a Python framework that makes it easier to build reliable and structured GenAI applications. Built by the same team behind **Pydantic**, the data validation library used across FastAPI, OpenAI SDKs, LangChain, and more, PydanticAI brings type safety, schema validation, and Python-native design to AI agents.

The framework was inspired by the success of **FastAPI**, which transformed web development with intuitive design and typed request handling. Similarly, PydanticAI brings that same developer-friendly experience to **LLM-based applications** by using Pydantic's validation engine for both inputs and outputs.

At the core of PydanticAI is a powerful abstraction: it allows developers to define an agent, structure its expected output with Pydantic models, inject tools and dependencies, and run it—all with clean Python code. This made it the perfect fit to **rebuild the n8n-prototyped agents** for question generation and grading into robust backend services.

5.2 Why Use PydanticAI?

5.2.1 Built by the Core Pydantic Team

PydanticAI's origin matters. It's built by the same team behind Pydantic, a tool that is trusted by OpenAI SDKs, LangChain, Hugging Face Transformers, and many of today's AI and backend systems. That credibility directly translates into confidence when building mission-critical GenAI applications like ***SIR***.

5.2.2. Model-Agnostic and Modular

One of the biggest advantages of PydanticAI is its **model-agnostic nature**. In ***SIR***, we use **Google's Gemini 2.0 Flash model**, and PydanticAI made integration seamless. Whether it's

OpenAI, Anthropic, Ollama, DeepSeek, or custom local models, PydanticAI abstracts the communication layer and lets developers focus on logic and outcomes.

5.2.3 Pythonic Design with Structured Responses

PydanticAI aligns perfectly with Python's best practices. It allows us to define result schemas using BaseModel, ensuring that every output generated by the LLM is **predictable, validated, and structured**. For our question generator, we define clear constraints for each MCQ and essay—down to how many answer choices are required, and what fields must exist.

This level of **type safety** is not just convenient—it's essential when building AI tools where downstream systems (like our mobile app) rely on consistent, structured data.

5.2.4 Logfire for Real-Time Debugging

While we haven't yet enabled **Pydantic Logfire** in ***SIR***, its ability to provide real-time observability, trace prompts, and monitor performance is a compelling reason for future adoption. As our app scales and supports more teachers, Logfire will help us identify response inconsistencies, monitor agent latency, and iterate faster.

5.2.5 Dependency Injection and Streaming Support

PydanticAI offers optional support for **dependency injection**, making it easy to supply tools, services, or configurations to the agent without hardcoding them into the prompt logic. It also supports **streamed responses**—letting applications begin processing output even before the model finishes generating the full response.

This makes the framework incredibly **scalable**, **responsive**, and adaptable to complex use cases like ours, where embedding, grading, and context retrieval all need to work together across multiple agents.

5.3 PydanticAI Compared to Other AI Agent Frameworks

When selecting a framework to build intelligent agents for ***SIR***, we evaluated several leading Python-based frameworks. Each had its unique philosophy, strengths, and ideal use cases. While tools like **LangChain**, **LangGraph**, and **CrewAI** offer powerful orchestration capabilities, **PydanticAI** stood out for its simplicity, reliability, and alignment with structured application development.

Below is a side-by-side comparison to illustrate how PydanticAI differs from these other frameworks:

Feature	**PydanticAI**	**LangChain**	**LangGraph**	**CrewAI**
Core Focus	Data validation and serialization	Orchestrating LLM interactions	Graph-based agent workflows	Multi-agent collaboration and task delegation
Primary Use Case	Defining and validating data structures for AI applications	Building complex LLM-powered applications with chains, tools, and memory	Creating intricate, multi-step agent workflows with visual representation	Designing systems with multiple agents working together to solve tasks
Key Strengths	Rigorous data validation and type safety	Extensive tool integrations and modularity	Visual workflow design and debugging	Efficient agent collaboration and task management
Key Weaknesses	Primarily focused on data handling, not directly for building agents	It can be complex for simpler applications	Less mature and widely adopted compared to LangChain	Still under active development, with limited documentation
Typical Use Cases	API development, data processing pipelines, ensuring data quality	Chatbots, question answering systems, document search, and more	Complex business processes, scientific simulations, and multi-agent systems	Customer support, research, and any domain requiring coordinated teamwork

Table 4: Comparison of AI Agent Frameworks

5.3.1 PydanticAI vs LangGraph

LangGraph is a high-control, graph-based framework used for building hierarchical and multi-agent systems. It supports advanced flows like token-level streaming, fault tolerance, and persistent memory. It's best suited for **enterprise-scale applications** where teams manage agent networks and workflows with visual interfaces (LangGraph Studio).

However, LangGraph's complexity can be a barrier for simpler or iterative development workflows. For ***SIR***, where our agents are modular but focused, **PydanticAI's lightweight yet powerful API** was a better fit. We didn't need multi-agent routing or hierarchical flows—what we needed was precise schema validation, clean outputs, and model flexibility.

5.3.2 PydanticAI vs LangChain

LangChain is one of the most popular frameworks for building LLM-powered applications. It shines in orchestration—managing tools, memory, and chains—but lacks the deep type enforcement and schema control that PydanticAI offers out of the box.

In fact, many LangChain applications rely on Pydantic models to structure and validate outputs. Where LangChain orchestrates the *how*, PydanticAI ensures the *what* is correct. If LangChain is the pipeline, PydanticAI is the safety net.

For our structured tasks—like generating multiple-choice questions with exactly four answer choices and validating that format—**PydanticAI was the right foundation**.

5.3.3 PydanticAI vs CrewAI

CrewAI focuses on collaborative, multi-agent environments. It's optimized for task delegation, agent collaboration, and project management among different roles. It's ideal for R&D teams managing experiments, training, and deployment cycles.

However, CrewAI doesn't offer strong schema validation or single-agent precision. In ***SIR***, where we're building deterministic workflows for education (not exploratory agent systems), **PydanticAI's discipline and control gave us the reliability we needed**.

5.4 How We Built Our AI Agents with PydanticAI

Our implementation of both agents followed PydanticAI's simple yet powerful structure. Below is a breakdown of how we turned the *Create Questions Agent* from a prototype into a production-grade agent.

Step 1: Install Dependencies

Install the necessary packages:

```
pip install pydantic-ai pydantic python-dotenv pypdf supabase
```

We'll also need:

- Access to a supported LLM provider (OpenAI, Gemini, etc.)
- Supabase account + API key with a vector store enabled

Step 2: Define Dependencies

Create a `dataclass` to structure inputs like PDF file path and Supabase client.

```
from dataclasses import dataclass
from supabase import create_client, Client

@dataclass
class PDFAgentDependencies:
    pdf_path: str
    supabase_client: Client
```

Step 3: Extract PDF Content

Add code to extract text from the PDF using `pypdf`.

```
from pypdf import PdfReader

def extract_pdf_text(pdf_path: str) -> str:
    reader = PdfReader(pdf_path)
    text = ""
    for page in reader.pages:
```

```
        content = page.extract_text()
        if content:
            text += content.encode("utf-8", "ignore").decode("utf-8")
    return text
```

Step 4: Define the Result Model

Define the expected structure of the model output using Pydantic.

```
from pydantic import BaseModel, Field
from typing import List

class MCQQuestion(BaseModel):
    question: str = Field(..., description="Multiple-choice question")
    model_answer: str = Field(..., description="Correct answer")
    choices: List[str] = Field(..., min_items=4, max_items=4)

class EssayQuestion(BaseModel):
    question: str = Field(..., description="Essay question")
    model_answer: str = Field(..., description="Model answer")

class ResultType(BaseModel):
    mcq_questions: List[MCQQuestion]
    essay_questions: List[EssayQuestion]
```

Step 5: Define the System Prompt

Create the system instructions for the agent.

```
system_prompt = '''
## Overview
You are an AI agent that generates multiple-choice and essay questions
based on a given text input.
...
## Final Notes
- Ensure MCQs are diverse and cover different aspects of the text.
- Keep essay questions open-ended but specific to the text.
- Maintain JSON structure integrity.
'''
```

Step 6: Create Tool for Embedding PDF to Supabase Vector Store

Create a tool using `@agent.tool` that takes the PDF text and stores embeddings in Supabase.

```
from pydantic_ai import RunContext
from pydantic_ai.agent import tool
import openai

@tool
async def embed_pdf(ctx: RunContext[PDFAgentDependencies], text_chunk: str)
-> str:
```

```
    # Get vector embedding (e.g., using OpenAI)
    response = openai.Embedding.create(
        input=text_chunk,
        model="text-embedding-ada-002"
    )
    embedding = response["data"][0]["embedding"]

    # Store in Supabase
    supabase = ctx.deps.supabase_client
    data, _ = supabase.table("pdf_embeddings").insert({
        "text": text_chunk,
        "embedding": embedding
    }).execute()

    return "Embedded successfully"
```

Step 7: Instantiate the Agent

Now glue everything together:

```
import os
from dotenv import load_dotenv
from pydantic_ai import Agent

load_dotenv()

# Setup Supabase
supabase_url = os.getenv("SUPABASE_URL")
supabase_key = os.getenv("SUPABASE_KEY")
supabase: Client = create_client(supabase_url, supabase_key)

# Path and extracted content
pdf_path = "/Users/adel/Documents/Project2_Sir/x.pdf"
pdf_text = extract_pdf_text(pdf_path)

# Input payload
user_input = {
    "mcq": 2,
    "essay": 2,
    "text": pdf_text
}

# Define and run the agent
agent = Agent(
    "google-gla:gemini-2.0-flash",
    deps_type=PDFAgentDependencies,
    result_type=ResultType,
    system_prompt=system_prompt
)

# Run agent
result = agent.run_sync(user_input, deps=PDFAgentDependencies(
    pdf_path=pdf_path,
    supabase_client=supabase
))
```

CHAPTER 6 : Backend

6.1 Importance of the Backend in Modern Applications

The backend is a critical component of any software application. It runs on the server and is responsible for managing data, implementing business logic, processing client requests, and ensuring security. In our project ***SIR***, the backend is the backbone of the entire system—it stores quizzes, handles grading through AI, manages authentication, and enforces access control based on user roles.

Beyond performance, the backend safeguards data. By keeping sensitive operations and data server-side, we minimize security risks such as data breaches. Laravel, the framework chosen for this project, supports encryption, token-based authentication, and role-based access control, making it a suitable choice for educational platforms handling user data.

6.1.2 Laravel as a Powerful PHP Framework for Backend Development

Laravel is one of the most popular PHP frameworks for backend development. It follows the MVC (Model-View-Controller) architectural pattern and is known for its elegant syntax and developer-friendly features. Laravel simplifies tasks like routing, authentication, migrations, and caching.

In the ***SIR*** app, Laravel allowed us to quickly structure our API for mobile access, manage a complex quiz system with students and teachers, and create reusable logic with controllers and repositories. Laravel Sanctum made implementing secure token-based authentication straightforward for mobile interaction.

6.2 Why Using Laravel

6.2.1 Use of MVC and Repository Pattern

Laravel naturally enforces the **MVC** (Model-View-Controller) pattern. Here's how it works in ***SIR***:

- A **route** receives a request (e.g., /quizzes).

- That request is passed to a **controller** (e.g., QuizController) which acts as a middleman.
- The controller delegates core logic to a **repository**, which contains functions for querying the database and applying filters.
- The repository interacts with **models** (e.g., Quiz, Question) which represent database tables.

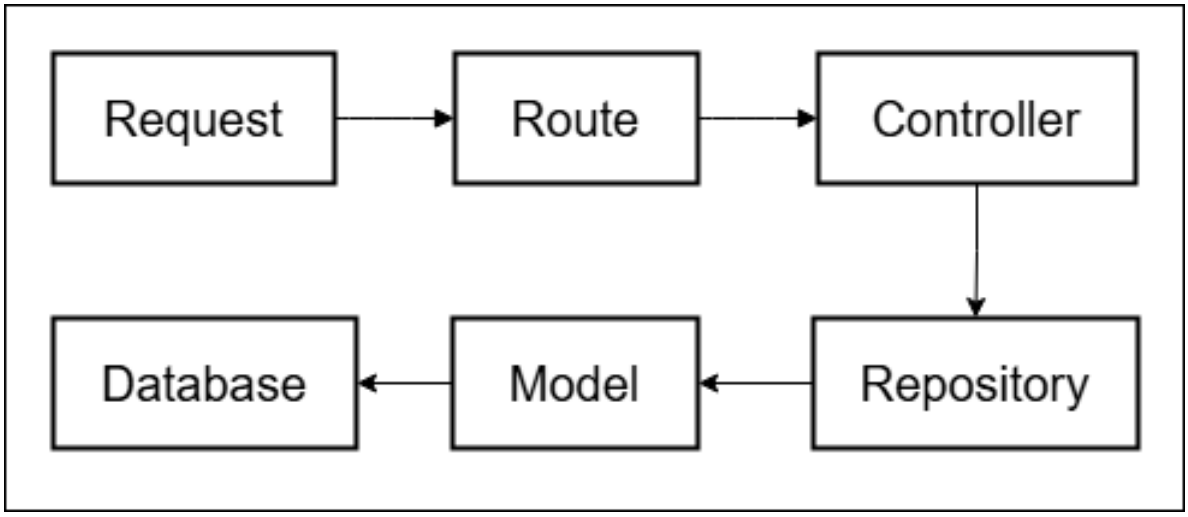


Figure 25: Backend Request Handling Flow

This separation of concerns improves maintainability and testing, allowing each part of the logic to be independently upgraded or debugged.

6.2.2 Migrations of Database

The Laravel migrations are version-controlled PHP files that define your database schema. They let teams collaborate easily and ensure that every developer or deployment environment uses the same structure.

In ***SIR***, one such migration creates the quizzes table:

```
public function up(): void
{
    Schema::create(table: 'quizzes', callback: function (Blueprint $table): void {
        $table->id();
        $table->string(column: 'name');
        $table->text(column: 'description');
        $table->boolean(column: 'is_active')->default(value: true);
        $table->integer(column: 'duration')->default(value: 0);
        $table->foreignId(column: 'class_id')->constrained(table: 'classes')->onDelete(action: 'cascade');
        $table->softDeletes();
        $table->timestamps();
    });
```

Figure 26: Laravel Migration for the Quizzes Table

This migration translates into the following schema:

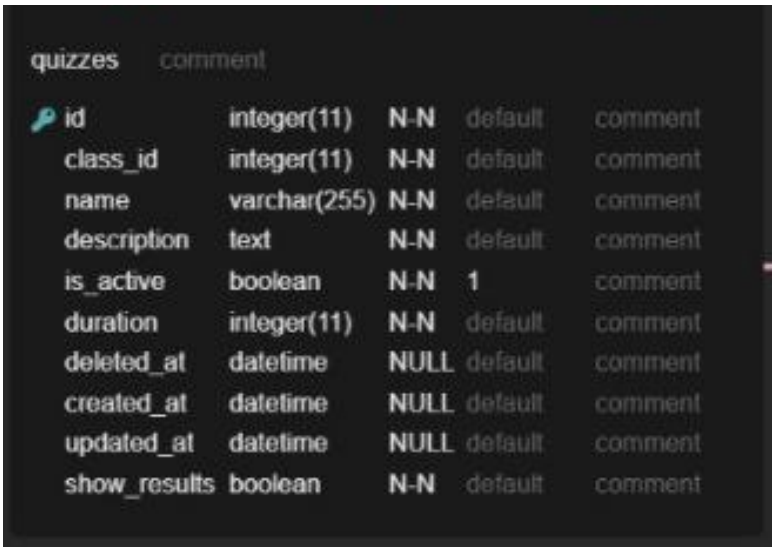


Figure 27: Quizzes Table Schema

You can see how each field (like name, description, duration) is stored and indexed. Relationships are also enforced via foreignId which links quizzes to classes.

6.2.3 ORM (Eloquent)

Laravel's Eloquent ORM (Object-Relational Mapper) simplifies working with databases. It maps tables to PHP classes and rows to objects. So instead of writing raw SQL, we use expressive syntax:

```
$quizzes = Quiz::where('is_active', true)->get();
```

This makes the code readable and manageable. Relationships like hasMany, belongsTo, and hasOneThrough are natively supported and allow linking teachers, students, quizzes, and answers easily.

6.2.4 Authentication (Sanctum)

Laravel Sanctum provides token-based authentication suitable for mobile apps. After login, a token is issued and attached to each subsequent request for identity verification.

```
Authorization: Bearer {token}
```

This ensures secure communication between the mobile frontend and backend. Only two public endpoints are accessible without tokens:

```
POST /auth/login
POST /auth/register
```

All other endpoints are protected via `auth:sanctum`.

6.3 Database Design

6.3.1 What is Database Design

Database design involves modeling the data for optimal storage and retrieval. It defines the structure—tables, fields, relationships—that reflects the application logic. This is especially important for ***SIR***, which handles quizzes, questions, answers, roles, and class assignments.

6.3.2 Steps of the Database Design

- **ERD (Entity-Relationship Diagram)**: Illustrates how tables relate to one another.

- **Relational Data Model**: Every quiz belongs to a class, every question belongs to a quiz, and every answer belongs to a student-question pair.

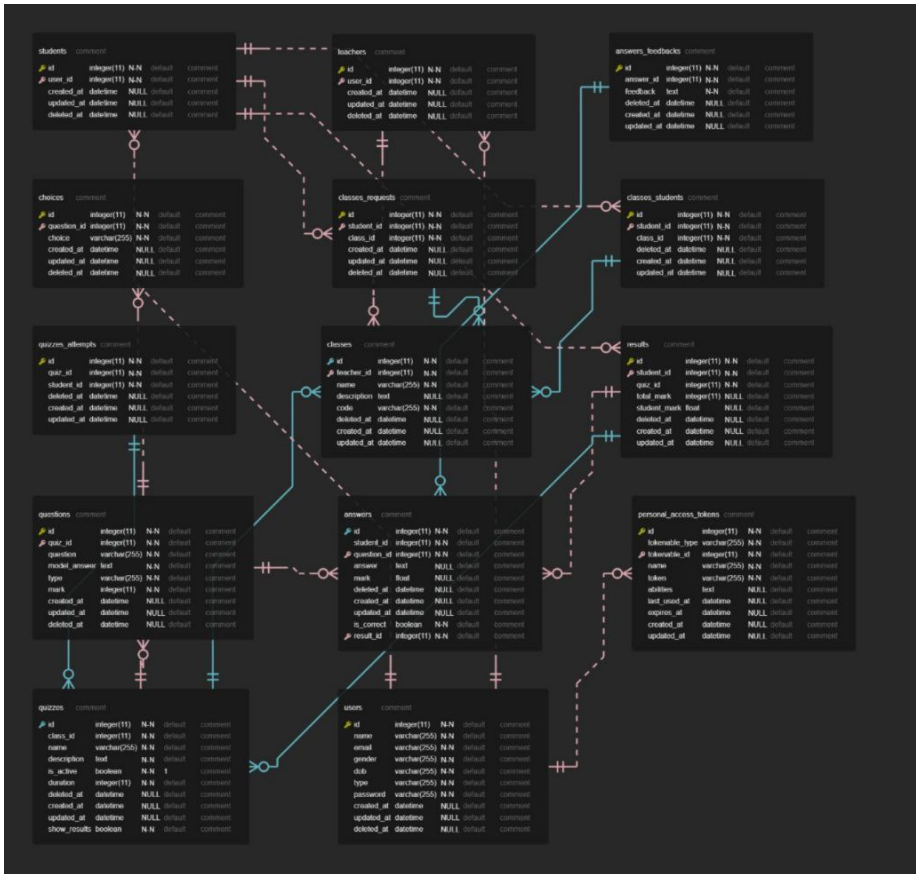


Figure 28: Sir Application Database ERD

6.3.3 Sample Tables

Let's examine a few core tables:

- **quizzes**: holds metadata like name, description, is_active, duration, class_id.
- **questions**: includes fields like question, model_answer, type (MCQ or essay), mark.
- **answers**: stores student responses, marks, and feedback.

These tables are connected:

- quizzes.id → questions.quiz_id
- questions.id → answers.question_id
- students.id → answers.student_id

These foreign keys ensure data integrity and cascading deletions.

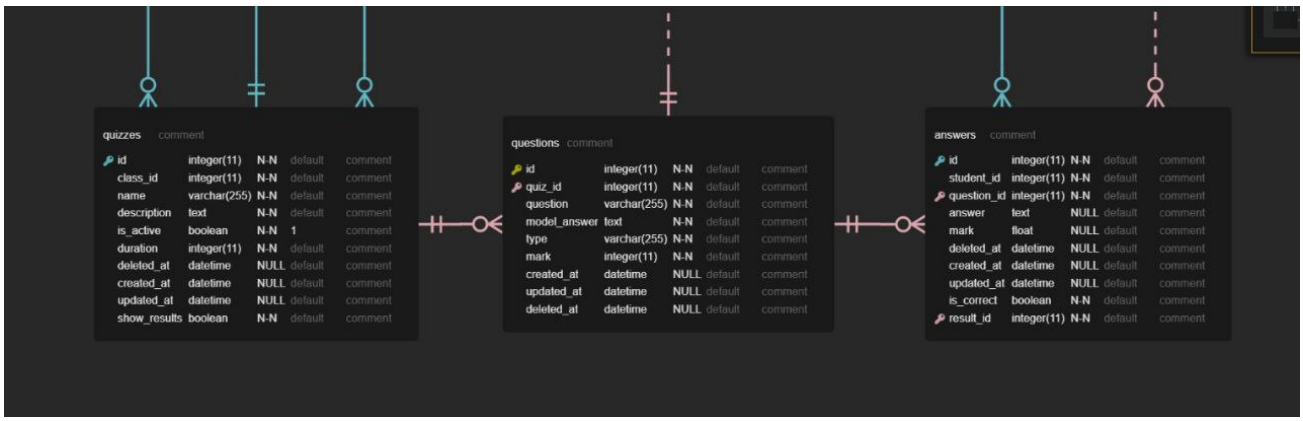


Figure 29: Questions-to-Answers Relationship Diagram

6.4 Application Design

6.4.1 Authentication (Sanctum)

Laravel Sanctum is used in *Sir* to provide secure, token-based authentication for the mobile application. When a user (either a teacher or student) logs in, Sanctum generates a **personal access token** that is used to authenticate future API requests.

Sanctum stores these tokens in a table called `personal_access_tokens`.

```php
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create(table: 'personal_access_tokens', callback: function (Blueprint $table): void {
            $table->id();
            $table->morphs(name: 'tokenable');
            $table->string(column: 'name');
            $table->string(column: 'token', length: 64)->unique();
            $table->text(column: 'abilities')->nullable();
            $table->timestamp(column: 'last_used_at')->nullable();
            $table->timestamp(column: 'expires_at')->nullable();
            $table->timestamps();       You, 2 months ago • First Patch & Created Main Tables
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists(table: 'personal_access_tokens');
    }
};
```

Figure 30: Laravel Migration for Personal Access Tokens Table

Each token is tied to a specific user model instance and stores information like when the token was last used or when it expires. This ensures that every API call from the mobile app can be validated against a secure backend record.

All authenticated API routes in ***SIR*** are protected using Laravel's `auth:sanctum` middleware. This middleware verifies the token included in the `Authorization` header of the request.

The routes are grouped under this middleware to protect them:

```php
//Auth Routes

Route::post(uri: 'auth/login', action: [UserController::class, 'login']);
Route::post(uri: 'auth/register', action: [UserController::class, 'register']);

//Resources

Route::middleware(middleware: 'auth:sanctum')->group(callback: function (): void {

    Route::get(uri: 'user', action: [UserController::class, 'user']);
    Route::resource(name: 'classes',controller: ClassController::class);
    Route::delete(uri: 'classes/{id}/students/{student_id}/remove', action: [ClassController::class, 'removeStudent']);
    Route::get(uri: 'classes/{id}/quizzes', action: [ClassController::class, 'getQuizzes']);
    Route::resource(name: 'classes_requests',controller: ClassRequestsController::class);
    Route::post(uri: 'classes_requests/{id}/accept', action: [ClassRequestsController::class, 'accept']);
    Route::post(uri: 'classes_requests/{id}/reject', action: [ClassRequestsController::class, 'reject']);
    Route::resource(name: 'classes_students',controller: ClassStudentsController::class);       You, last month • init&clas
    Route::resource(name: 'quizzes',controller: QuizController::class);
    Route::get(uri: 'quizzes/{id}/attempt', action: [QuizController::class, 'attempt']);
    Route::post(uri: 'quizzes/submit', action: [QuizController::class, 'submit']);
    Route::resource(name: 'results',controller: ResultController::class);

});
```

Figure 31: Laravel Sanctum Authentication and Resource Route Definitions

This pattern ensures that any unauthorized request—those without a valid token—will be immediately rejected with a `401 Unauthorized` response.

Additionally, since ***SIR*** supports two types of users (teachers and students), role checks are added inside the controller methods using helper functions like `isTeacher()` and `isStudent()` to enforce further access control beyond basic authentication.

6.4.2 Quiz Attempt Logic

SIR enforces rules on quiz attempts to ensure fairness and accuracy. Here's how it works:

- **One Attempt Only**: When a student tries to take a quiz, the backend checks if an attempt exists in the quizzes_attempts table.
- **Timer Enforcement**:
    - If the time has expired, return 422 - Quiz time is over.
    - If the quiz is ongoing, resume with remaining time.
- **Enrollment Checks**:
    - The student must be enrolled in the class.
    - The quiz must have is_active = true.
- **After Submission**:
    - If show_results = true, student sees feedback.
    - Otherwise, just sees the result status.

This logic ensures a reliable and fair testing experience.

6.5 API Routes in Postman

An essential aspect of backend development is ensuring that the available API endpoints are well-documented and easy to test. For the ***SIR*** application, we leveraged Postman to document, test, and share our API in a developer-friendly interface.

The API is designed to support secure communication between the mobile frontend and the Laravel backend, handling everything from authentication to quiz management, grading, and user roles.

6.5.1 Accessing the Documentation

The ***SIR*** API is publicly available through Postman via the following link:

🔗 ***SIR* API Documentation on Postman**
(https://documenter.getpostman.com/view/33043197/2sB2qUn4yY)

This interactive collection enables developers to explore endpoints, view examples, and test real requests using pre-filled request templates.

6.5.2 Structure of the API Collection

The API routes in the ***SIR*** project are grouped into meaningful categories in Postman:

- **Authentication**
- **User Management**
- **Class Management**
- **Quiz Management**
- **Question Management**
- **Quiz Attempts**
- **Answers & Grading**
- **Result Retrieval**

Each section includes appropriate HTTP methods (`GET`, `POST`, `PUT`, `DELETE`), authentication requirements, and example payloads or responses.

6.5.3 Authentication Endpoints

Authentication is handled using Laravel Sanctum. These endpoints are used to register and authenticate users and issue personal access tokens for subsequent requests.

Endpoint	Method	Description	Authentication
`/auth/register`	POST	Registers a new user (teacher or student)	❌ Public
`/auth/login`	POST	Logs in a user and returns a token	❌ Public

Example successful login response:

```
{
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOi..."
}
```

Once authenticated, this token is used in the `Authorization` header for all future requests:

```
Authorization: Bearer {token}
```

6.5.4 User Management

These routes allow users to view and update their own profiles.

Endpoint	Method	Description	Authentication
`/user/profile`	GET	Retrieve authenticated user's profile	✅ Required
`/user/profile`	PUT	Update profile details (e.g., name, email)	✅ Required

These routes are protected using `auth:sanctum`. The backend identifies the user based on the bearer token and returns or updates the relevant data accordingly.

6.5.5 Class Management

Teachers can manage classes—create, edit, or delete them—while students can view the classes they are enrolled in.

Endpoint	Method	Description	Access
`/classes`	GET	Get all classes	All roles
`/classes`	POST	Create a new class	Teachers only
`/classes/{id}`	GET	Get details of a specific class	All roles
`/classes/{id}`	PUT	Edit class info	Teachers only
`/classes/{id}`	DELETE	Delete a class	Teachers only

Role checks are implemented in controller methods using helper functions like `isTeacher()` to prevent unauthorized access.

6.5.6 Quiz Management

Quiz-related routes allow teachers to create, activate, and manage quizzes while allowing students to access only the ones they are eligible to take.

Endpoint	Method	Description	Access
`/quizzes`	GET	List all quizzes for a class or user	All roles
`/quizzes`	POST	Create a quiz	Teachers only
`/quizzes/{id}`	GET	View quiz details	All roles
`/quizzes/{id}`	PUT	Update quiz metadata	Teachers only
`/quizzes/{id}`	DELETE	Remove a quiz	Teachers only

Each quiz has attributes like `is_active`, `duration`, and `show_results`, which impact how it behaves during the attempt phase.

6.5.7 Question Management

Each quiz can include multiple questions. These routes manage question creation, retrieval, and categorization (MCQ or essay).

Endpoint	Method	Description	Access
`/quizzes/{quiz_id}/questions`	GET	Retrieve questions of a quiz	All roles
`/quizzes/{quiz_id}/questions`	POST	Add a new question to a quiz	Teachers only

Each question contains:

- `question`: the question text
- `model_answer`: used in AI grading
- `type`: either `mcq` or `essay`
- `mark`: assigned weight for the question

6.5.8 Quiz Attempt & Answer Submission

Students interact with quizzes through a sequence of endpoints that initiate, resume, and submit attempts.

Endpoint	Method	Description	Access
`/quizzes/{quiz_id}/attempt`	POST	Start or resume a quiz attempt	Students only
`/quizzes/{quiz_id}/submit`	POST	Submit answers for grading	Students only

During a quiz attempt, the backend:

- Checks if the student already attempted it (only one attempt allowed)
- Validates that the student is enrolled in the class
- Enforces timing using server-side validation

6.5.9 Result and Feedback

After a quiz is submitted, students can check their results—only if `show_results = true` is enabled by the teacher.

Endpoint	Method	Description	Access
`/quizzes/{quiz_id}/results`	GET	View result summary, feedback, and score	Students only

Grading may be automatic (especially for MCQs) or AI-assisted (for essay answers). Feedback is stored in the `answers`table and linked to specific questions for clarity.

6.5.10 Testing with Postman

To use the Postman collection effectively:

1. **Import** the collection from the shared link.
2. **Set up environment variables**, such as:
    - `base_url`: URL of your local or deployed backend
    - `token`: Set this after logging in
3. **Authorize requests** by including your bearer token:
4. `Authorization: Bearer {{token}}`
5. **Use pre-filled examples** in each request to test both happy paths and edge cases (e.g., submitting expired quizzes, unauthorized access).

This allows you to simulate real workflows like:

- A student registering, enrolling in a class, attempting a quiz, and receiving feedback.
- A teacher creating classes and quizzes, assigning students, and monitoring performance.

CHAPTER 7: React Native

7.1 INTRODUCTION

In today's mobile-first world, developing applications that are user-friendly and responsive across platforms is essential. For the mobile development phase, we focused on creating a robust, user-friendly interface for both iOS and Android. The frontend in mobile applications refers to the part of the app that users see and interact with. It includes everything from the app's design and layout to its buttons, text, images, and other interactive elements.

7.2 USING FRAMEWORKS

Our mobile frontend development relied on frameworks rather than native platform-specific coding (e.g., Objective-C/Swift for iOS or Java/Kotlin for Android). This decision was made to simplify the development process, speed up development time, and maintain high code quality.

Frameworks provide pre-built libraries, templates, and components that can be seamlessly integrated into the project. This allows developers to focus on writing application-specific logic while the framework handles the lower-level implementation details.

Additionally, frameworks follow standardized conventions and guidelines, enabling better collaboration among developers. The modular structure they enforce makes the code more organized, maintainable, and easier to scale.

Most frameworks also come equipped with built-in solutions for common tasks such as navigation, state management, and animations, reducing the need for repetitive code. This results in faster development cycles, consistent code quality, and higher productivity. Features like live or hot reloading further enhance the development process by providing instant feedback on code changes which was perfect for our project.

7.3 WHAT IS REACT NATIVE?

React Native is a framework based on the React library that enables developers to build mobile applications using a component-based architecture. This approach divides the app into reusable components, making the codebase easier to understand, debug, and maintain. Each component encapsulates its own logic and design, allowing it to be reused across the application.

React Native uses JSX, a syntax extension that combines JavaScript with XML-like tags, to define components. Instead of using HTML elements (like or <button>), React Native provides platform-specific components (like <View>, <Text>, and <TouchableOpacity>). JSX also allows developers to write conditional logic directly within the component's layout, enabling dynamic rendering based on app state or user interaction.

React Native leverages a **shadow tree** and communicates directly with native platform elements through a **bridge**. This bridge translates JavaScript code into native code, enabling high-performance updates to the UI. When a component's state changes, React Native efficiently updates the shadow tree and synchronizes the changes with the native views.

This architecture allows React Native to deliver near-native performance while maintaining the flexibility and productivity of JavaScript development. Its ability to use a shared codebase for both iOS and Android makes it a standout framework for cross-platform mobile application development.

7.3.1 Why Use React Native?

When deciding on a technology for our mobile application, we evaluated various options and ultimately chose React Native. This decision was driven by React Native's unique strengths that align with our project goals. Here are the key reasons why React Native stood out as the ideal choice for mobile app development:

1. Rapid Development : React Native promotes rapid development by enabling developers to build mobile apps with reusable components, hot reloading, and a shared JavaScript codebase. This approach simplifies cross-platform development for both iOS and Android, significantly reducing development time and effort. The ability to see real-time changes during development enhances productivity and accelerates the iterative process.

2. Code Reuse: One of React Native's strongest features is its support for code reuse. The codebase is shared between iOS and Android, reducing duplication and ensuring consistency across platforms. Additionally, React Native offers pre-built components and libraries tailored for mobile UI, which developers can leverage to streamline development and speed up delivery.

3. Efficiency and Organization: React Native's component-based architecture promotes efficiency and organization. This modular approach allows developers to divide the app into reusable, manageable components, making it easier to debug, maintain, and scale. React Native's structure fosters cleaner, more maintainable code, which is especially beneficial in larger projects with multiple developers.

4. Developer Productivity: React Native significantly enhances developer productivity with features like **live reload** (providing instant feedback on code changes) and **hot reload** (allowing state retention during updates). These features make it easier to iterate and experiment without restarting the application. Additionally, the robust ecosystem of tools and libraries available in React Native empowers developers to focus on building application logic rather than dealing with repetitive or low-level tasks.

5. Community Support: React Native has a large and active developer community, which is a valuable resource for developers. The community provides extensive documentation, tutorials, third-party libraries, and tools, ensuring that developers can find solutions to challenges and adopt best practices. Continuous contributions from the community help keep React Native updated and relevant, making it a reliable choice for modern mobile development.

7.4 React Native Architecture

The React Native architecture is the foundation of how we built ***SIR***, a cross-platform application designed to provide seamless accessibility solutions for users on both Android and iOS. Leveraging React Native's capabilities, we implemented a robust system to ensure efficient data flow, smooth UI rendering, and responsiveness in user interactions. This architecture operates through a series of steps, broken into three main phases:

1. Render Phase

React executes the JavaScript code to create a React Element Tree. This tree is then used to generate a React Shadow Tree in C++.The React Shadow Tree mirrors the structure of the React Element Tree but operates at a lower level for efficient layout calculations.

2. Commit Phase

The React Shadow Tree undergoes layout calculation using a library called Yoga, which determines the size and position of each element. After this, the Shadow Tree is promoted to the "next tree" ready for mounting.This phase ensures the tree is optimized and ready for rendering, with asynchronous operations improving performance.

3. Mount Phase

The React Shadow Tree is transformed into a Host View Tree that corresponds to the platform's native views. For example, on Android, it creates `android.view.ViewGroup` and `android.widget.TextView`, while on iOS, it creates `UIView`.At the end of this phase, the UI is rendered on the screen.

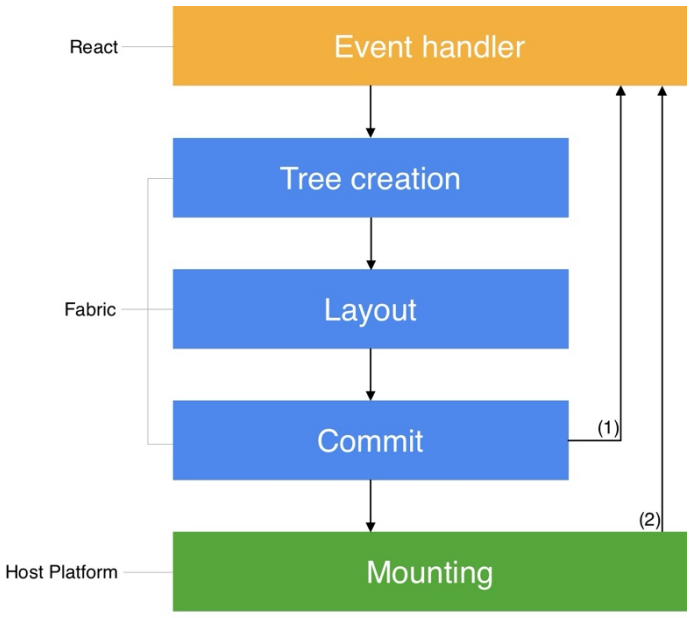


Figure 32: React Rendering Lifecycle with Fabric Architecture

7.5 Cross-Platform Implementation

The introduction of a core C++ renderer, called Fabric. Fabric provides a unified and efficient rendering logic across platforms, allowing us to focus on creating a consistent experience for all users.

Key features of the Fabric renderer utilized in ***SIR*** include:

- **Shared Core:** A single C++ implementation ensures uniform behavior and reduces platform-specific discrepancies in ***SIR***'s UI.
- **Reduced Overhead:** By integrating Yoga directly into the core renderer, this minimized the performance overhead for layout calculations, especially for interfaces and screens in ***SIR***.
- **Immutability Enforcement:** Leveraging C++'s immutability features, this ensured thread-safe operations for ***SIR***'s UI updates, preventing concurrency issues during heavy usage.

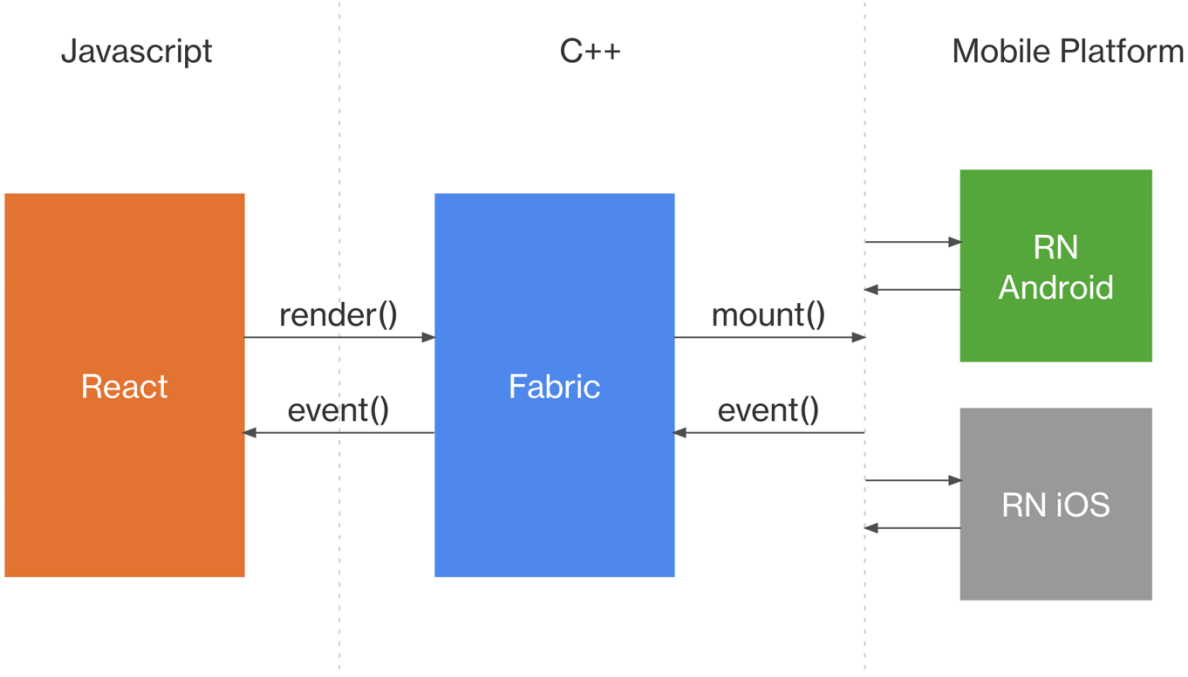


Figure 33: React Native Communication Flow using Fabric Architecture

The architecture's flow can be visualized in three main layers:

1. JavaScript Layer

- **React:** This layer contains the JavaScript code for ***SIR***'s logic, defining the structure and behavior of components such as the interfaces for choosing translation modes and

displaying results. React creates the React Element Tree and interacts with the Fabric renderer through render() and event() methods.

2. C++ Core (Fabric)

- **Fabric Renderer:** The central component that:
  - Accepts React's render() calls to create and manage the React Shadow Tree.
  - Handles layout calculations using Yoga for elements like ***SIR***'s translation selection buttons and recording interfaces.
  - Communicates with both the JavaScript layer and the host platform layer to synchronize rendering and user interactions.

3. Mobile Platform Layer

- **RN Android and RN iOS:** Fabric interacts with platform-specific APIs to render ***SIR***'s UI and handle user interactions. It uses mount() to place components into the native view hierarchy and event() to process actions like taps and gestures.

Example Flow Using *SIR*:

1. **Render:** React calls render() to pass the React Element Tree to Fabric. For example, this could represent ***SIR***'s main interface where users can create or join classes
2. **Mount:** Fabric uses mount() to create platform-specific views. For instance:
   - On Android, we created ViewGroup elements for the join interface and the quiz display.
   - On iOS, we used UIView components for interactive elements like quiz timer and classes screens.
3. **Event Handling:** When a user taps the "Join" button in ***SIR***, an event is generated and sent back to Fabric. Fabric processes this event and communicates it to React, which updates the app's state to start the recording or display the translated results.

By utilizing React Native's architecture, we built ***SIR*** to ensure efficient rendering, seamless updates, and a consistent experience for users across both platforms.

7.6 Expo: Simplifying Mobile Development

In the process of creating the ***SIR*** **app**, Expo played a vital role in streamlining our development workflow. By providing a comprehensive toolkit tailored for mobile app development, Expo allowed us to focus on building features that aligned with ***SIR***'s mission of delivering a seamless and user-friendly experience. Its tools and services simplified testing, configuration, and deployment, enabling us to iterate quickly while maintaining high-quality standards. With Expo, we could prioritize what truly matters: ensuring the ***SIR*** app is visually appealing, intuitive, and responsive to the needs of our users, all while reducing the complexity of mobile app development.

How Expo Simplifies Development

1. **Preconfigured Environment**:

   Expo takes care of all the complex native code implementations for both iOS and Android platforms, effectively handling the heavy lifting involved in mobile app development. This allowed our team to focus entirely on writing JavaScript, without needing to dive into the intricacies of platform-specific configurations or setups. By eliminating the need for such detailed platform-level work, Expo significantly accelerated our development process. It not only saved us a considerable amount of time but also ensured that our workflow remained streamlined and efficient, enabling us to deliver high-quality results with greater speed and consistency.

2. **Expo Go**:

   The **Expo Go** app enabled us to test the ***SIR*** app instantly on physical devices or simulators without recompilation. This real-time feedback made debugging faster and ensured the app performed as intended across different devices.

3. **Over-the-Air (OTA) Updates**:

   OTA updates provided us with the ability to deliver bug fixes and introduce new features directly to ***SIR*** users without the delays typically associated with app store approval processes. This functionality ensured that ***SIR*** stayed current and responsive to user feedback, maintaining a seamless and efficient experience for our audience. By bypassing traditional update hurdles, we could quickly adapt and continuously enhance the platform to meet evolving user demands.

4. **Expo Libraries**:

   Expo provided us with a rich set of libraries that simplified common development tasks for the app:

   - **expo-asset**: Streamlined image caching and static resource management, improving app performance.
   - **expo-font**: Allowed us to incorporate custom fonts, like Poppins, enhancing the app's design and branding to reflect our app identity.
   - Other tools for notifications, permissions, and app functionalities reduced development effort while maintaining functionality.

5. **EAS Build and Deploy**:

   With **Expo Application Services (EAS)**, we could build and distribute production-ready versions of the app effortlessly for both iOS and Android. This streamlined process saved time and ensured a smooth transition from development to production.

7.7 TypeScript in the *SIR* App

During the development of the ***SIR*** **app**, we chose to use **TypeScript** as our primary programming language. This decision was driven by TypeScript's ability to enhance the development process, providing tools that contribute to the app's reliability, maintainability, and scalability. TypeScript proved to be an essential part of creating a seamless and user-friendly experience for ***SIR***'s users.

Key Benefits of Using TypeScript :

- **Robust Type-Checking System**:

  Ensuring the accuracy of data throughout the app was critical. TypeScript enforces strict rules on the types of data used in the code, allowing us to catch potential issues early in development. This reduced the likelihood of bugs and ensured smooth functionality across the app.

- **Improved Code Clarity and Comprehension**:

  By explicitly defining data types, TypeScript made the app's codebase easier to read and understand. This not only streamlined collaboration among developers but also made the code easier to maintain and enhance over time. Clear type definitions also acted as helpful documentation for the app.

- **Error Prevention**:

  With the interconnected features of the app, TypeScript served as a safeguard against unexpected behaviors. It helped detect and prevent errors during development, leading to a smoother development process and a more stable application.

- **Enhanced Maintainability**:

  TypeScript's consistent and predictable patterns ensured that the app's codebase was well-organized and easy to maintain. This makes it simpler to introduce new features or updates in the future without introducing unintended issues.

7.8 Folder Structure of the *SIR* App:

To ensure clarity and scalability, the ***SIR*** app is organized into a well-defined folder structure. This structure facilitates efficient collaboration, maintainability, and seamless scalability as the project grows. Here's an overview of the key directories and their functionalities:

Figure 34 : *SIR* Project Directory Structure

1. app: This directory contains the main application screens and layout components.It includes files like `audio-to-sign.tsx`, `splash-screen.tsx`, and other key pages that make up the core user interface and functionality of the ***SIR*** app.

2. components:

Houses reusable components to maintain consistency throughout the app.Includes modular components like `button.tsx` and `index-box.tsx`, which are used across different parts of the app to ensure a unified design and functionality.

3. api:

Manages API integration and client setup.Contains configurations for Axios and React Query in files like `api-provider.tsx` and `client.tsx`.

4. stores:

Centralized state management using the **zustand** library to handle app-wide states, ensuring smooth and efficient screen transitions, with implementation in files like `resources.tsx` and `result.tsx`.

5. assets:

Organizes all static resources like images, fonts, and icons.Stores visual and multimedia assets in subdirectories (`fonts`, `images`), making it easier to manage and reference files like `Splash.gif`, `logo-2.gif`, and `index_image_1.png`.

6. types:

types: Defines shared TypeScript types and interfaces used across the application. This folder centralizes data structure definitions, ensuring type safety and consistency for props, API responses, and state management.

7.9 Navigation

For seamless transitions between screens, the app leverages **React Native Stack Navigation**. This navigation solution provides a smooth and intuitive user experience, ensuring that users can effortlessly navigate through the app's interface.

The navigation flow is managed by registering all application screens within a **stack navigator**. This configuration is centralized in the `Layout.tsx` file, which serves as the core navigation hub for the app. React Native Stack Navigation offers a robust and flexible solution for handling navigation, built upon the reliable foundation of the React Navigation library.

This setup ensures that transitions between screens are fluid and user-friendly, contributing to an overall polished and cohesive app experience.

7.10 State Management

To effectively manage the app's state and data, we employed the **Zustand library**, a powerful and lightweight solution designed for scalability and flexibility. Zustand acts as a dependable system for organizing and managing state across the app, ensuring seamless communication between different parts of the application.

In practical terms, Zustand provides a centralized repository, or "store," where we define and organize essential data and actions using plain JavaScript objects and functions. This approach keeps the state management process simple, efficient, and easy to maintain, even as the application evolves.

Technically, Zustand's API is built around React hooks, enabling direct and efficient access to the app's state. Components connected to the store automatically re-render when the state they depend on changes, eliminating the need for complex data-passing mechanisms or excessive boilerplate.

Zustand also tackles common pitfalls in state management, such as context loss and React concurrency, making it a robust choice for managing the ***SIR*** app's data flow. Its balance of simplicity and power ensures a smooth and consistent user experience throughout the application.

7.11 API Integration

In the ***SIR*** app, seamless communication between the mobile application and external services is essential for delivering accurate results and providing a smooth user experience. To achieve this, we integrated APIs within the mobile app.

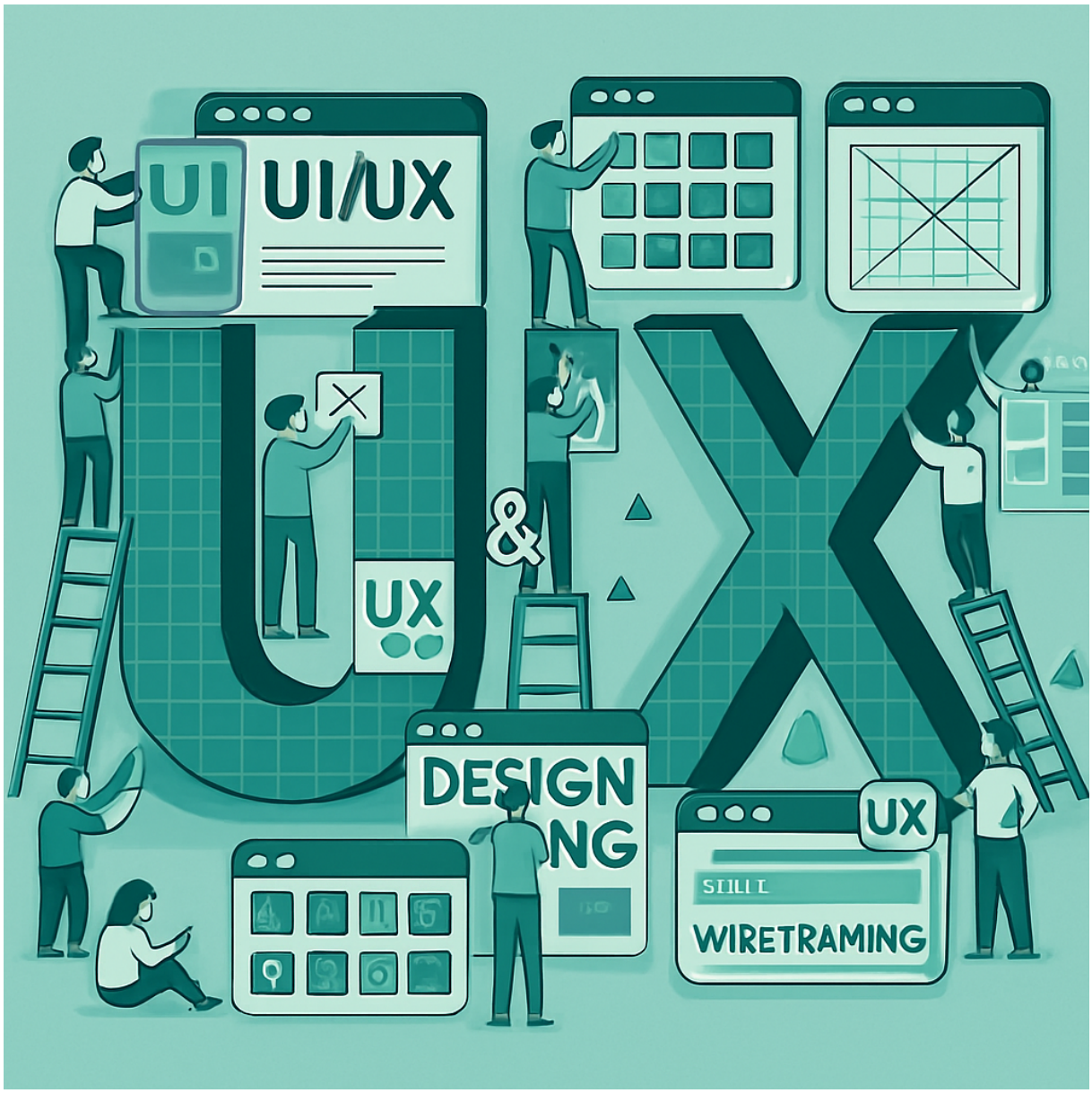
API Integration in the Mobile App

To handle API requests in the mobile app, we chose to combine the **@tanstack/react-query library** with **Axios**, leveraging the strengths of both tools:

- **React Query**:
  - Ensures improved performance through intelligent caching and background fetching.
  - Automatically updates the app's state when API data changes, reducing the need for manual state management.
  - Handles complex API scenarios like retries and stale data handling, ensuring the app remains responsive even during network issues.
- **Axios**:
  - Provides a robust solution for sending HTTP requests with easy-to-use syntax.
  - Offers built-in error handling and request cancellation, allowing for more efficient API interactions.
  - Simplifies the process of setting headers, query parameters, and payloads for API requests.

This combination of **React Query** and **Axios** empowers the ***SIR*** app to deliver a reliable and efficient experience, ensuring data is fetched and displayed in a user-friendly manner.

CHAPTER 8: User Interface / User Experience (UI/UX)



8.1 INTRODUCTION

The success of an educational platform hinges on its ability to merge **User Interface (UI)** and **User Experience (UX)** seamlessly. While UI governs the visual and interactive elements, UX defines the overall usability and satisfaction of the product. Together, they ensure the platform is not just functional but also engaging and efficient for its users. This chapter explores how ***SIR***, our educational dashboard, was designed with a focus on **clarity, accessibility, and role-specific workflows** to create a streamlined experience for both students and teachers.

- **User Interface (UI): The Visual Layer**

  UI encompasses the app's visual design—buttons, navigation menus, color schemes, and layouts that users interact with. For ***SIR***, the UI was designed to be:

  - **Role-specific**: Distinct yet consistent interfaces for students (e.g., quiz attempts, class enrollment) and teachers (e.g., quiz creation, request management).
  - **Minimalist and intuitive**: Prioritizing clear hierarchies, readable typography, and familiar patterns (e.g., a unified "+" icon for enrollment/quiz creation).
  - **Accessible**: High-contrast colors, scalable text, and keyboard-friendly navigation to accommodate diverse needs.

- **User Experience (UX): The Interaction Journey**

  UX focuses on the end-to-end experience—from login to task completion. ***SIR*'s UX** was crafted to:

  - **Reduce friction**: Straightforward workflows (e.g., one-click quiz activation for teachers, auto-graded results for students).
  - **Adapt to roles**:
    - **Students**: Emphasis on quick access to quizzes, progress tracking, and class updates.

- **Teachers**: Tools for efficient quiz generation (MCQ/written), AI-assisted grading, and student request management.

- **Promote engagement**: Features like AI-generated questions and real-time feedback loops keep users motivated.

By unifying **UI clarity** with **UX efficiency**, ***SIR*** ensures both students and educators spend less time navigating and more time achieving their goals.

8.2 BRANDING

Branding is more than a logo or color palette—it's the **identity** of a product and the emotional connection it fosters with users. For ***SIR***, branding plays a pivotal role in communicating the platform's mission of **seamless education, collaboration, and empowerment** for students and teachers. A strong brand ensures ***SIR*** is not just recognizable but also resonates with its academic audience, building trust and long-term engagement.

Brand Identity :

1. **Clarity**
    - Reflects the platform's **intuitive design** and streamlined workflows.
    - Ensures users (students and teachers) can focus on learning and teaching—not navigation.
2. **Collaboration**
    - Embodies ***SIR***'s role as a **bridge** between students and educators.
    - Highlights features like class enrollment, quiz feedback, and AI-assisted grading that foster interaction.
3. **Innovation**
    - Showcases ***SIR***'s **smart tools**, such as AI-generated quizzes and automated results.

- Positions the platform as a forward-thinking solution for modern education.

Visual Elements

- **Logo:** The ***SIR*** logo embodies elegance and modernity with bold typographic design. It merges formality with simplicity, reflecting the platform's academic purpose and digital fluency. The dark tones with a pop of green symbolize trust, intelligence, and growth—values core to the educational journey. The overlapping characters in the logo suggest harmony and the merging of roles—student and teacher—on a single platform.

Figure 35 : *SIR* Logo Design

- **Color Palette:** Color is one of the most powerful elements in design—it sets the emotional tone, guides attention, and builds recognition. In ***SIR***, the color palette was carefully selected to reflect the platform's focus on clarity, trust, and forward-thinking education. The use of **deep charcoal** and **teal green** represents professionalism, balance, and innovation. The minimalistic color scheme supports the user experience by reducing visual clutter, allowing students and teachers to focus on content rather than distractions.

- **Teal Green** (#00796B): symbolizes growth, intelligence, and active learning. It is used to highlight key actions and navigational elements.
- **Charcoal Black** (#2E2E2E): evokes a sense of stability and seriousness, forming the primary text and background foundation.
- **Soft Grey Accents**: are used for secondary information and interface dividers, improving legibility without overwhelming the user.

Together, these colors reinforce ***SIR***'s identity as a modern, dependable platform for education.

Figure 36: *SIR* Color Palette

- **Typography:** Typography is a key component in almost every digital experience. However, its complexity and industry-specific language makes it a common source of misalignment and confusion. You don't have to be a typography expert to design digital interfaces, but it's important to know some of these terms in order to have meaningful conversations with others on your team. Communicating clearly with team members about typography can help teams:
    - Increase legibility (and therefore usability) of an interface.
    - Improve visual polish and professional appearance.
    - Create a more consistent brand identity.
    - Cut down on costly revisions and iterations.

Aa

English Font

Poppins

Type	Font Size
Heading 1	20px
Heading 2	18px

Body – Large	16px
Body – Regular	14px
Body – Small	12px
Caption	12px
BUTTON	16px

Figure 37: English Typography Overview

Why Branding Matters for *SIR*

A strong brand ensures that ***SIR*** stands out in the growing landscape of educational technology while staying true to its mission of **enhancing learning through seamless collaboration**. It creates a **cohesive experience** across all touchpoints—from the dashboard interface to institutional partnerships—building **trust, recognition, and loyalty** among students and educators.

By aligning its branding with its **core values of clarity, collaboration, and innovation**, ***SIR*** not only attracts users but also **inspires them to engage more deeply** with the platform. For students, it becomes a reliable companion in their academic journey. For teachers, it transforms into an **indispensable tool** that simplifies assessment and fosters better learning outcomes.

Ultimately, ***SIR***'s branding reinforces its role as more than just an app—it's a **bridge between education and technology**, empowering users to achieve more with less friction.

8.3 USERFLOW

User flow is the path a user takes to complete a specific task within an application. It maps out every step, from the initial interaction to the final goal, ensuring a seamless and intuitive experience. For ***SIR***, designing an effective user flow was essential to create a smooth, user-friendly dashboard that supports both students and teachers in managing classes, quizzes, and results efficiently. Whether it's a student requesting to join a class or a teacher creating AI-generated quizzes, each interaction is streamlined to support ease of use and maximize productivity.

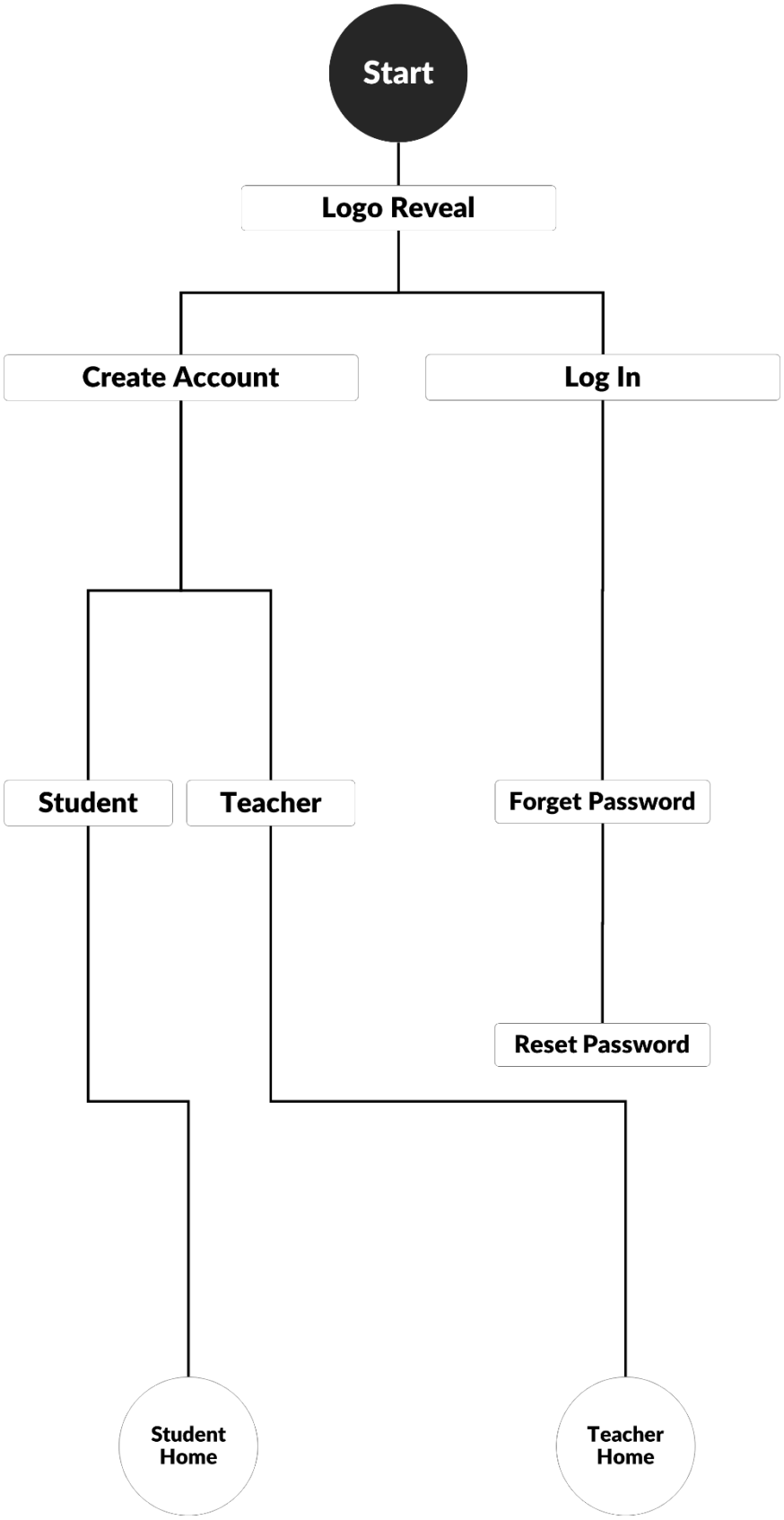


Figure 38: Application Start

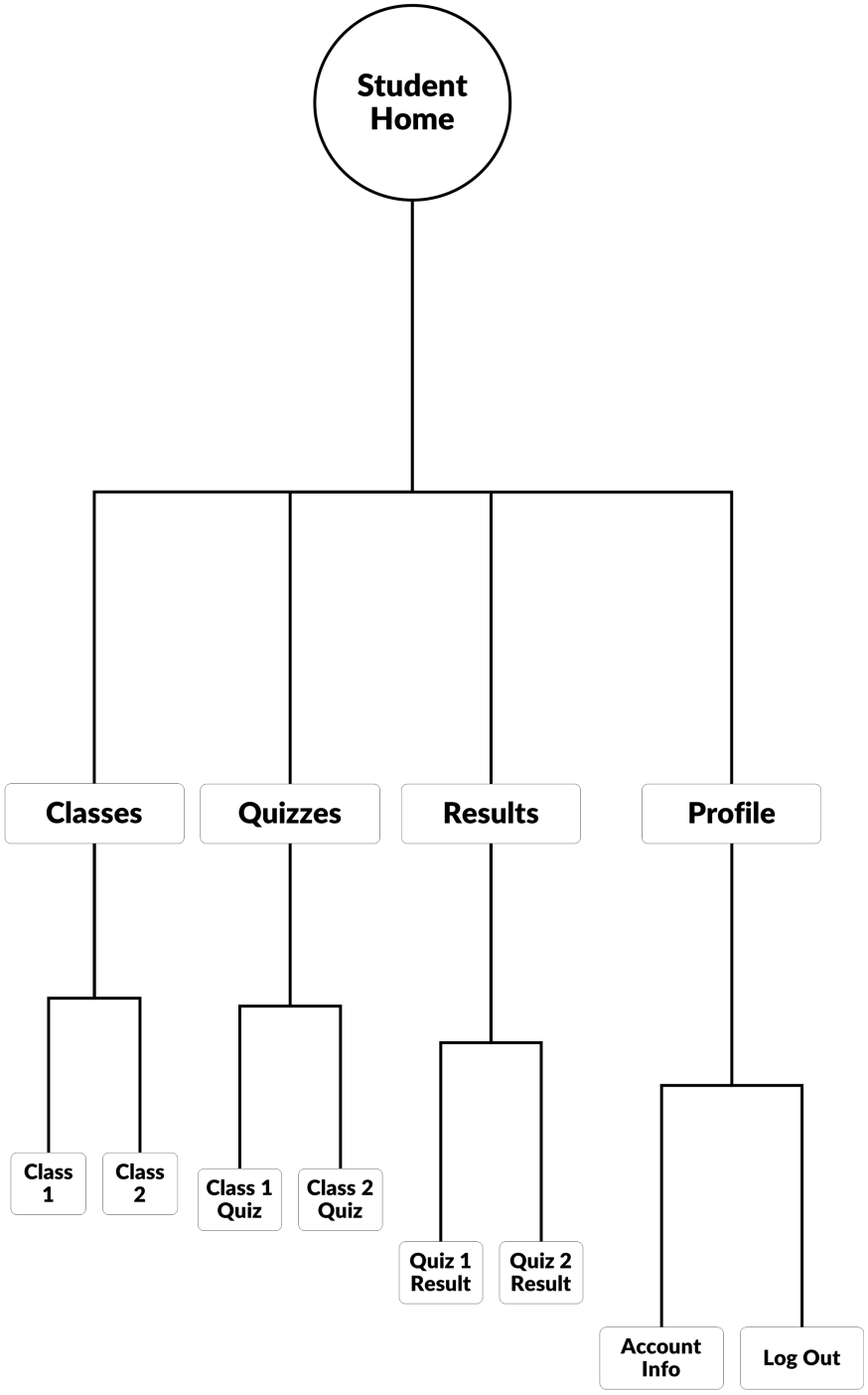


Figure 39: Student Userflow

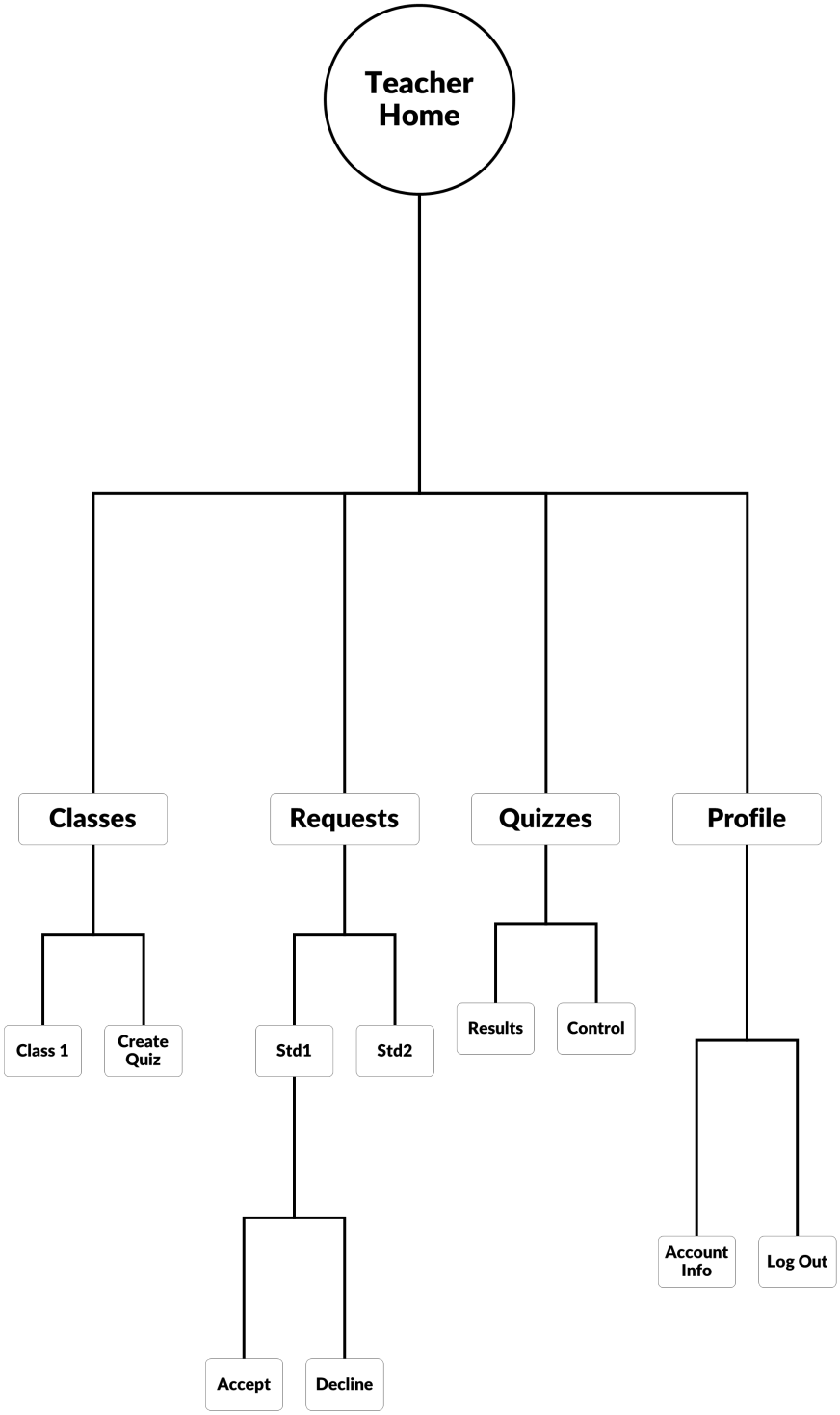


Figure 40: Teacher Userflow

8.4 TOOLS USED:

The design and development of ***SIR*** leveraged industry-standard tools to create a seamless, visually cohesive, and functional platform. Each tool addressed specific aspects of UI/UX design, prototyping, and asset creation, ensuring alignment with ***SIR***'s goals of clarity, collaboration, and innovation.

- **1. Figma**
- **Purpose:** UI/UX Design, Prototyping, and Collaboration
- **Role in *SIR*:**
    - Designed wireframes and high-fidelity mockups for both student and teacher dashboards.
    - Created interactive prototypes to test user flows (e.g., quiz submission, class enrollment).
    - Enabled real-time team collaboration for iterative feedback on layouts, navigation, and accessibility.
    - Used auto-layout and components to ensure consistency across role-specific interfaces.
- **2. Adobe Photoshop**
- **Purpose:** Asset Creation and Image Optimization
- **Role in *SIR*:**
    - Designed custom icons (e.g., quiz, results, profile tabs) with a unified visual style.
    - Edited and optimized visual assets for the platform (e.g., banners, tutorial graphics).
    - Created marketing materials (e.g., app previews, institutional pitch decks).
- **3. Adobe Illustrator**
- **Purpose:** Vector Graphics and Brand Identity
- **Role in *SIR*:**

- Designed scalable vector assets, including the ***SIR*** logo, typography, and interface elements.
- Ensured brand consistency across platforms (web, mobile, print).
- Developed illustrations for onboarding screens and empty states (e.g., "No quizzes yet").

Why These Tools Were Chosen

The combination of **Figma, Adobe Photoshop, and Illustrator** provided an end-to-end design solution for ***SIR***:

- **Figma** facilitated agile, collaborative design—critical for balancing student/teacher needs.
- **Photoshop/Illustrator** ensured professional, on-brand visuals that inspire trust in an educational context.
- Together, they enabled ***SIR*** to **merge functionality with aesthetic appeal**, creating a platform that users find both intuitive and engaging.

8.5 Conclusion

The design of ***SIR*****'s** User Interface (UI) and User Experience (UX) reflects its core mission: **to simplify and enhance the educational process for both students and teachers**. By merging a **clean, role-specific UI** with **intuitive, goal-driven UX**, ***SIR*** ensures that users can focus on learning and teaching—not navigation. The strategic emphasis on **branding clarity, optimized user flows, and professional design tools** (Figma, Adobe Suite) underscores the meticulous planning behind creating a **cohesive, efficient, and engaging** platform.

These elements work in harmony to:

- **Empower students** with quick access to quizzes, results, and class materials.
- **Equip teachers** with AI-assisted tools for effortless quiz creation and grading.
- **Bridge the gap** between education and technology through seamless collaboration.

As we transition into the final chapter, we explore ***SIR*'s deployment and scalability**, focusing on how modern frameworks (e.g., React, Firebase) and cloud-based solutions were leveraged to ensure:

- **Reliable performance** during high-traffic periods (e.g., exam seasons).
- **Secure data handling** for sensitive academic records.
- **Cross-platform accessibility** (web, mobile) for flexible learning.

This phase marks the culmination of ***SIR***'s development journey—transforming a vision into a **functional, impactful tool** that redefines classroom interaction.

CHAPTER 9: Deployment

9.1 Introduction

Deployment is a critical part of the development lifecycle, bridging the gap between building an application and making it available to real users. For our project ***SIR***, the deployment strategy was intentionally designed around modularity, scalability, and ease of management.

The application consists of **two separate backends**, each written in a different programming language, serving distinct purposes:

9.1.1 Laravel Backend – Deployed on Hostinger

The core backend of ***SIR*** is built using **Laravel v10.10**, a robust PHP framework. This part of the system is responsible for:

- Managing user authentication and authorization (teachers and students).
- Handling class creation and student enrollment.
- Managing quiz creation, publishing, attempts, and result retrieval.
- Serving data to the mobile frontend through RESTful API endpoints.

To host this backend, we chose **Hostinger**, a reliable web hosting provider that supports SSH access, Git-based deployments, and Laravel-compatible environments. Hostinger's control panel allowed us to connect our private Git repository securely and push updates with minimal friction.

We used SSH access to run Laravel-specific setup commands like composer install, php artisan migrate, and environment configuration updates. Laravel's .env environment file was modified to fit production settings including the database connection, app URL, and security keys.

In production, HTTPS encryption was enabled via Hostinger's built-in **SSL certificate manager**, ensuring that all data exchanged between the app and API is securely encrypted.

9.1.2 FastAPI AI Backend – Deployed on Vercel

The second backend component of ***SIR*** was the **AI agent service**, responsible for:

- Generating quiz questions from a PDF file.

- Grading essay questions using the model answers.
- Delivering real-time feedback to students based on their answers.

This microservice was built using **FastAPI**, a modern Python web framework optimized for performance and async processing. We chose **Vercel** for its support of serverless functions, automatic builds, zero-downtime deployments, and simple integration with Git.

This AI backend is completely separate from the main Laravel backend to preserve performance and prevent heavy AI operations from blocking standard API traffic.

The FastAPI project was structured under a directory containing main.py, requirements.txt, and a vercel.json configuration file. This setup defined how Vercel should build and serve the project. Once initialized with the vercel CLI and connected to our account, the project was deployed with a single command: vercel --prod.

Using Vercel for deployment also meant that:

- We could roll back deployments instantly.
- Logs were readily available for debugging.
- Environments were lightweight and scalable.

9.1.3 Why We Chose This Strategy

Splitting the application into two backend deployments was a **deliberate architectural choice** for the following reasons:

- **Separation of Concerns**: The Laravel backend handles CRUD operations, user logic, and mobile API responses. The Python backend is isolated to run AI workloads using PydanticAI and Gemini API.
- **Language-Specific Optimization**: PHP is better supported in shared hosting environments like Hostinger, whereas Python AI code is better suited for cloud platforms with serverless capabilities.
- **Cost Efficiency**: Vercel offers generous free-tier support for serverless APIs, making it perfect for our Python-based inference layer.

In summary, the deployment of ***SIR*** is structured, scalable, and robust. It aligns with modern DevOps practices by leveraging:

- Git-based deployment on Hostinger for the Laravel API.
- Serverless deployment via Vercel for the AI agent microservice.

This dual-backend approach gave us flexibility during development and confidence in deploying to a real production environment.

9.2 Backend Deployment on Hostinger

9.2.1 Deployment Overview

The Laravel backend of ***SIR*** was deployed on **Hostinger**. Hostinger provides integrated Git deployment and SSH access, making it convenient to manage Laravel projects through CLI and version control.

9.2.2 Deployment Steps

Repository Setup (Git Deployment)

- The Git repository was connected directly from GitHub using Hostinger's control panel under the "Private Git Repository" section.
- An **SSH key** was added to GitHub for secure access.
- The correct branch (`main`) was selected.
- Set up auto-deployment to trigger updates on git push using ssh key.

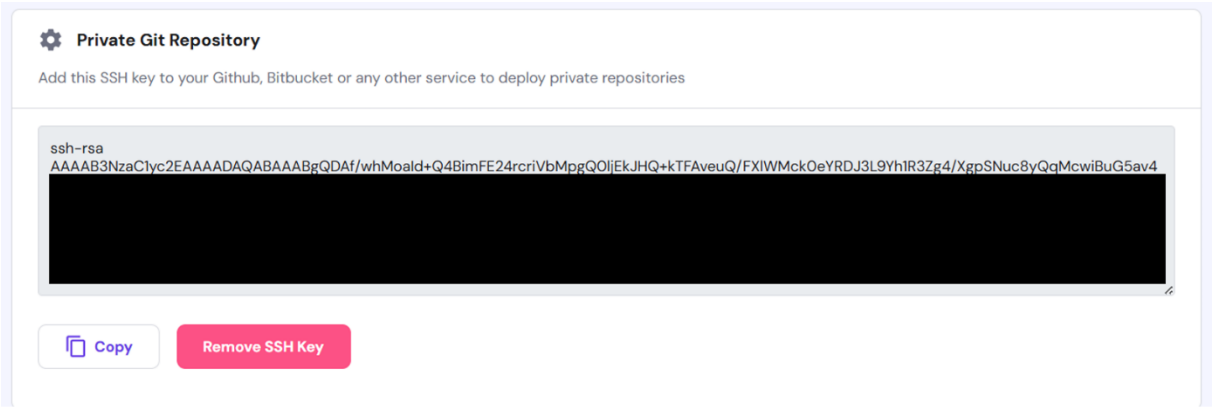


Figure 41: SSH Key Configuration for Private Git Repository Access

This Git integration ensured that any code updates pushed to the repository were automatically pulled and reflected on the live server.

SSH Access & Initial Setup

SSH access was established using the credentials and port provided by Hostinger:

```
ssh -p 65002 u354186246@89.117.139.129
```

Once connected, the terminal was used to navigate to the correct Laravel project directory:

```
cd domains/sirapi.mazenamir.com
```

Then the necessary Laravel dependencies were installed:

```
composer install --optimize-autoloader --no-dev
```

The `.env` file was copied and configured for the production environment:

```
cp .env.example .env
nano .env
```

Application key was generated:

```
php artisan key:generate
```

Database Migration & Seeding

The database was migrated with the following command:

```
php artisan migrate
```

This created all required tables for users, quizzes, results, and other entities.

SSL Certificate

Hostinger's built-in SSL tools were used to secure the API domain (`sirapi.mazenamir.com`) with HTTPS. This is essential for encrypted communication, especially since ***SIR*** handles login credentials and sensitive grading data.

9.3 FastAPI Deployment on Vercel

9.3.1 Deployment Overview

For the AI functionality — generating questions and grading essay answers using PydanticAI — we built a FastAPI microservice that runs Python code and interacts with the Gemini API.

Instead of deploying this backend on a traditional server, we leveraged **Vercel** for a serverless experience.

9.3.2 Deployment Steps

Preparation

- A new directory (`deployment/`) was created to house the FastAPI project.
- The folder included:
  - `main.py` (the entry point FastAPI file)
  - `.env` for secrets (e.g., Gemini API key)
  - `requirements.txt` listing dependencies
  - `vercel.json` for deployment configuration

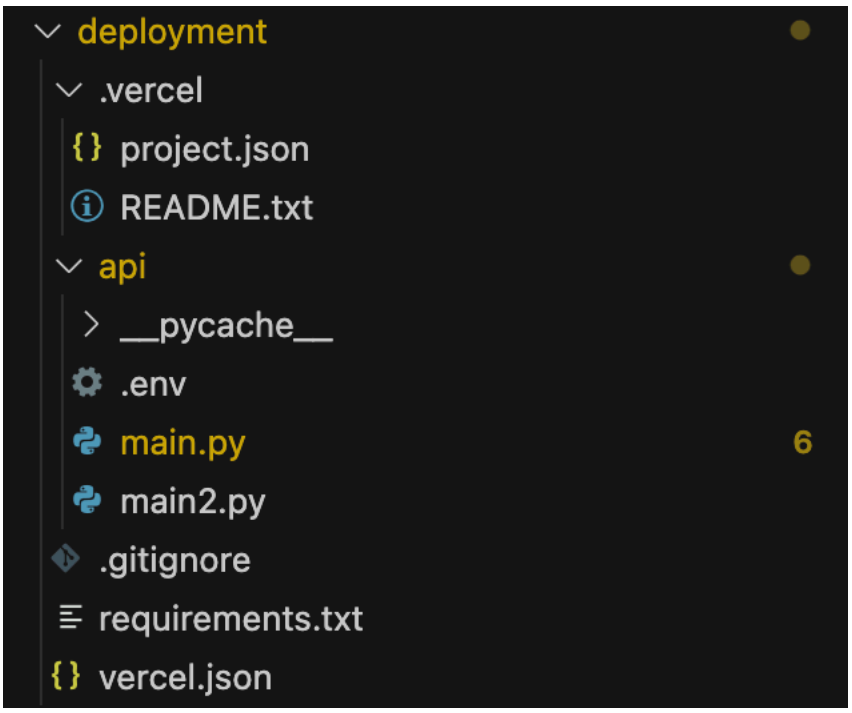


Figure 42: Vercel Deployment Folder Structure

Configuration with Vercel

Installed the **Vercel CLI**:

```
npm install -g vercel
```

Initialized the project with:

```
vercel login
vercel init
```

Created the required config in `vercel.json`:

Figure 43: Vercel Configuration File for Python API Deployment

After configuration, the project was deployed with:

```
vercel --prod
```

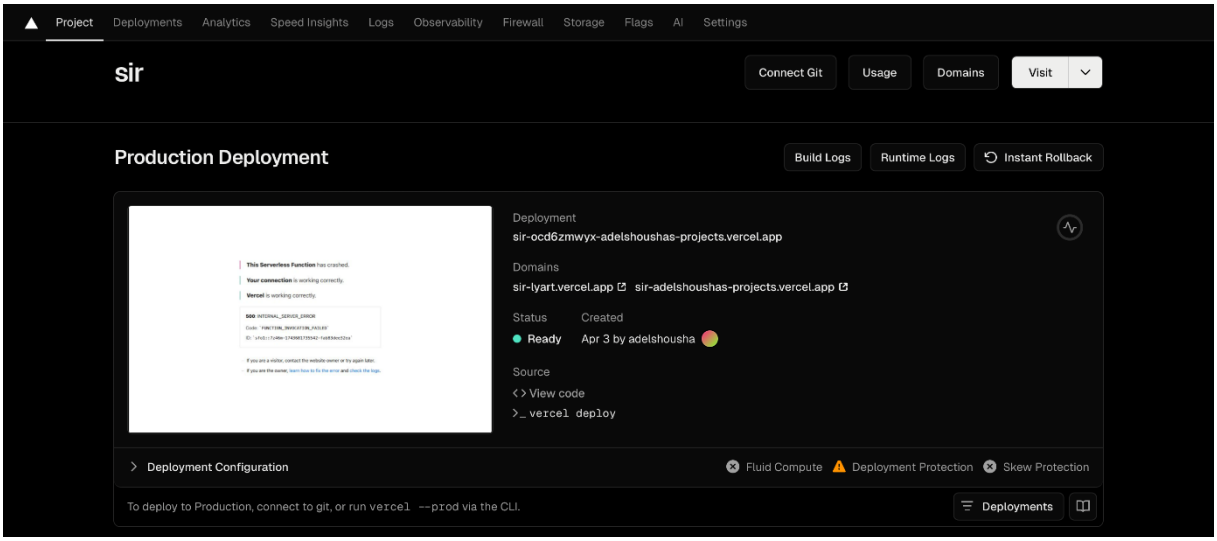


Figure 44: Vercel Production Deployment Dashboard for Sir Application

Domain & Deployment Success

The app was deployed to a Vercel-generated domain:

```
https://sir-adelshoushas-projects.vercel.app
```

The build logs confirmed successful deployment.

If there was a crash (as seen in earlier attempts), logs were available to help debug — usually related to environment variables or missing dependencies.

9.4 Conclusion

Deploying ***SIR*** required a thoughtful, dual-approach strategy—leveraging **Hostinger** for the Laravel-powered main backend and **Vercel** for the Python-based AI agent. This separation allowed each service to operate in its ideal environment, improving maintainability and performance. By combining traditional hosting with modern serverless deployment, ***SIR*** achieves both stability and scalability, setting a strong foundation for future growth and real-world use.

CHAPTER 10: Business Model

10.1 Introduction

In order to successfully deliver our AI-powered chatbot application, ***SIR***, which is designed to assist educators in generating exam questions efficiently, we adopted the Business Model Canvas as a strategic planning tool. The Business Model Canvas provides a comprehensive framework that helps us visualize and align the core components of our business idea in a clear and structured manner. It enables us to define the value ***SIR*** offers, identify our target users, and understand how we intend to deliver this solution effectively.

By using this model, we are able to break down our concept into nine essential building blocks: value proposition, customer segments, channels, customer relationships, revenue streams, key resources, key activities, key partners, and cost structure. Each of these components plays a crucial role in shaping the sustainability, functionality, and scalability of ***SIR***.

This approach not only ensures that our technical development is aligned with real user needs, but also helps us present ***SIR*** in a way that is academically thorough and business-oriented. The following sections present a detailed breakdown of each component of the Business Model Canvas as applied to our

10.2 Value Proposition

With ***SIR***, we aim to provide a practical and intelligent solution to a common challenge faced by educators: the time-consuming and repetitive task of creating exam questions. Our value proposition is centered on saving time and reducing effort for teachers and university professors by enabling them to generate high-quality exam questions quickly and efficiently. ***SIR*** supports multiple question formats—such as multiple-choice, true/false, and essay-type questions—and uses AI to suggest questions based on the subject matter and syllabus content.

By minimizing human error and improving the overall process, ***SIR*** adds real value to the educational workflow.

10.3 Customer Segments

We have identified our main customer segments as individuals and institutions involved in the education sector. These include university professors, school teachers, educational institutions, and curriculum developers. Each segment benefits differently from ***SIR***. For example, university professors may use it to generate diverse exam banks for different levels, while school teachers may find it helpful in preparing quizzes more quickly. By understanding the specific needs of these groups, we have tailored ***SIR*** to serve them in the most effective way possible.

10.4 Channels

To deliver ***SIR*** to our target users, we are relying on several key channels. Our primary platform is a mobile application, which ensures accessibility and ease of use for educators on the go. Additionally, we plan to promote ***SIR*** through social media platforms, which are effective tools for reaching individual educators and education communities. Strategic partnerships with universities and schools will also serve as important distribution channels, allowing us to introduce ***SIR*** directly into real educational environments.

10.5 Customer Relationships

We believe that building and maintaining strong relationships with our users is critical to the long-term success of ***SIR***. To achieve this, we offer dedicated technical support through chat and email, ensuring that users can get help when needed. We also collect user feedback to continuously improve the app and adapt it to evolving educational needs. A free trial version is available to attract new users and allow them to experience ***SIR*** before committing to a paid

plan. Through regular updates and enhancements, we aim to build trust and satisfaction among our user base.

10.6 Revenue Streams

SIR follows a flexible revenue model to suit different types of users. We offer monthly and yearly subscription plans (Basic, Plus, and Pro), allowing users to choose the level of service that best meets their needs. A pay-per-exam option is available for occasional users who may not require a full subscription. In addition, a limited free version gives users access to basic features, while licensing deals with educational institutions provide a more scalable source of income. This diversified approach ensures that ***SIR*** remains financially sustainable.

10.7 Key Activities

Our core activities focus on the development, maintenance, and continuous improvement of ***SIR***. These include training the AI model to understand various question types, performing quality assurance on the generated questions, and designing a user-friendly dashboard that educators can interact with easily. We also allocate significant effort to outreach and marketing activities to grow our user base and introduce ***SIR*** to new audiences. Ensuring a smooth and secure user experience is central to everything we do.

10.8 Key Resources

To successfully develop and maintain ***SIR***, we rely on a set of essential resources that support both the technical and strategic aspects of the project. First and foremost, our **development team** plays a central role, consisting of specialists in artificial intelligence, backend infrastructure, and frontend interfaces. Their combined expertise allows us to build a robust and intelligent chatbot that meets the practical needs of educators.

In addition, we utilize **cloud-based hosting and storage services**, which ensure that user data and generated exam content are handled securely, with high availability and scalability. We

also maintain a **structured database of educational curricula and sample questions**, which enables ***SIR*** to generate contextually relevant and pedagogically sound exam content.

Beyond technical assets, we collaborate closely with **educational consultants and subject-matter experts** who help validate the quality and accuracy of the questions produced by ***SIR***. Their input ensures that the AI not only generates grammatically correct content, but also aligns with real academic standards. Furthermore, our use of advanced **UI/UX design tools** allows us to create a seamless and intuitive user interface that educators can easily navigate, regardless of their technical background.

These resources collectively form the foundation of ***SIR***'s functionality, performance, and user satisfaction.

10.9 Key Partners

Strategic partnerships have been a key component in the successful planning and deployment of ***SIR***. We have engaged **university professors and school teachers** early in the development process through pilot testing, which has helped us gather practical feedback and refine the chatbot based on real-world use cases. Their involvement ensures that the platform evolves in alignment with the actual workflow of educators.

In addition to pilot users, we maintain relationships with **educational consultants and academic advisors** who provide us with valuable insights into teaching methodologies, question structuring, and assessment standards. Their guidance has been instrumental in training our AI model to produce meaningful and pedagogically appropriate exam content.

We are also forming partnerships with **online educational platforms and institutions**, which provide not only access to potential users, but also credibility and exposure within the educational technology space. These collaborations open doors for large-scale adoption and institutional licensing, ensuring that ***SIR*** is positioned for growth and long-term impact.

By building strong partnerships across different levels of the education sector, we are reinforcing the relevance, trustworthiness, and effectiveness of ***SIR*** as a next-generation tool for exam generation.

10.10 Cost Structure

Operating and scaling ***SIR*** involves several essential costs. These include expenses for cloud infrastructure and data storage, salaries for the development and support team, and subscriptions to AI tools and APIs. We also allocate budget for marketing campaigns to promote the application and attract users. Finally, documentation, testing, and training materials contribute to the overall cost structure. By carefully managing these expenses, we ensure the financial feasibility of ***SIR***.

10.11 Conclusion

Through the development of our Business Model Canvas, we were able to gain a clear and structured understanding of how ***SIR***, our AI-powered chatbot, can provide meaningful value to the educational sector. By analyzing each component—from our value proposition to our revenue streams—we ensured that our solution is not only technically sound, but also aligned with real market needs and operational sustainability.

This model allowed us to approach the project with a business mindset, helping us consider both user experience and long-term viability. It guided our strategic thinking in identifying the most effective ways to reach our users, maintain strong relationships with them, and build a scalable and cost-effective product.

Ultimately, the Business Model Canvas has played a vital role in transforming ***SIR*** from a technical idea into a complete solution with the potential for real-world application and future growth. It has provided us with a roadmap to move forward confidently, both as developers and as innovators in the field of educational technology.

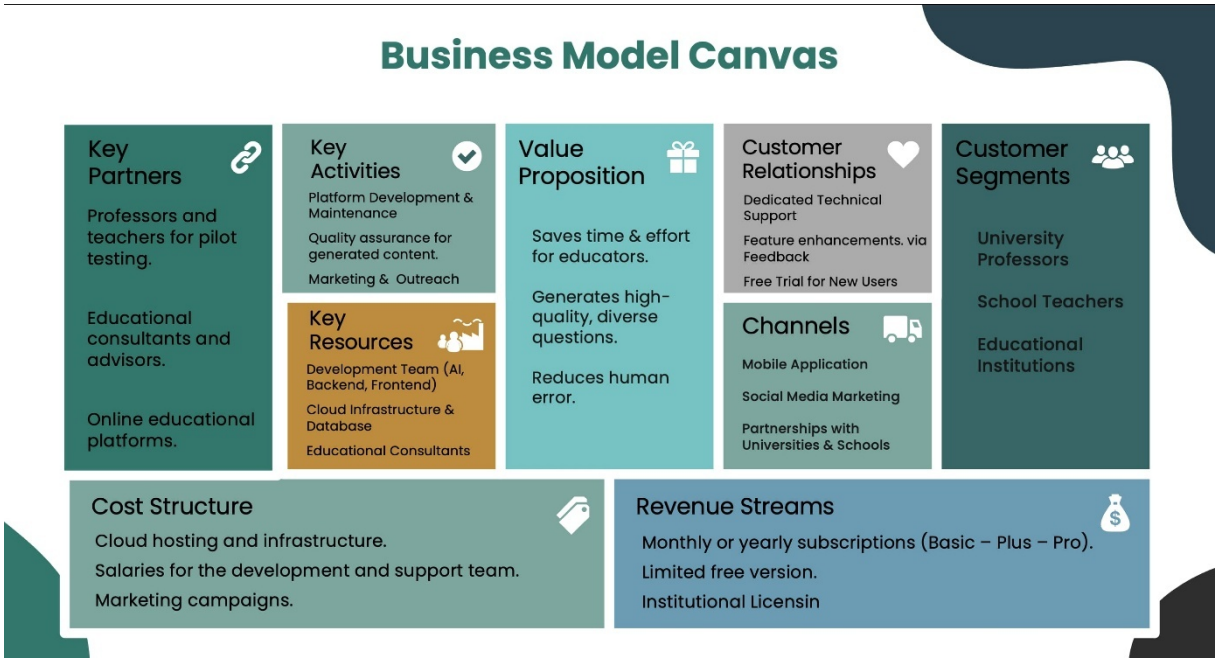


Figure 45: Business Model Canvas for SIR Application

REFERENCES

REFERENCES

[1] IBM, "What Are AI Agents?" https://www.ibm.com/think/topics/ai-agents

[2] Amazon Web Services, "What are AI Agents? - Agents in Artificial Intelligence Explained." https://aws.amazon.com/what-is/ai-agents/

[3] IBM, "What is RAG (Retrieval Augmented Generation)?" https://www.ibm.com/think/topics/retrieval-augmented-generation

[4] Amazon Web Services, "What is RAG? - Retrieval-Augmented Generation AI Explained." https://aws.amazon.com/what-is/retrieval-augmented-generation/

[5] GeeksforGeeks, "What is Retrieval-Augmented Generation (RAG) ?" https://www.geeksforgeeks.org/what-is-retrieval-augmented-generation-rag/

[6] Weaviate, "What is Agentic RAG."

https://weaviate.io/blog/what-is-agentic-rag

[7] n8n, "Powerful Workflow Automation Software & Tools." https://n8n.io/

[8] Hostinger, "What is n8n - revolutionize workflow automation." https://www.hostinger.com/tutorials/what-is-n8n

[9] PydanticAI, "PydanticAI." https://ai.pydantic.dev/

[10] ProjectPro, "How to Build an AI Agent with Pydantic AI: A Beginner's Guide." https://www.projectpro.io/article/pydantic-ai/1088

[11] Laravel, "Laravel Documentation." https://laravel.com/docs

[12] Django Software Foundation, "The web framework for perfectionists with deadlines." https://www.djangoproject.com

[13] Express, "Express - Node.js web application framework."

https://expressjs.com

[14] Node.js, "Node.js Official Documentation."

https://nodejs.org/en

[15] React Native, "React Native Main Documentation."

reactnative.dev

[16] React Native, "React Native Render Pipeline Architecture."

https://reactnative.dev/architecture/render-pipeline

[17] React Native, "React Native Cross-Platform Implementation Architecture." https://reactnative.dev/architecture/xplat-implementation

[18] Expo, "Expo Main Documentation." https://expo.dev

[19] TypeScript, "TypeScript Official Documentation." https://www.typescriptlang.org

[20] React Navigation, "Stack Navigator Documentation." https://reactnavigation.org/docs/stack-navigator/

[21] Zustand, "Zustand Demo and Documentation." https://zustand-demo.pmnd.rs

[22] NPM, "React Query Package Documentation." https://www.npmjs.com/package/@tanstack/react-query

[23] Axios, "Axios Introduction Documentation." https://axios-http.com/docs/intro

[24] Figma, "Figma Help Center." https://help.figma.com/hc/en-us

[25] Adobe, "Adobe XD UXP Developer Documentation."

https://developer.adobe.com/xd/uxp

[26] Adobe, "Photoshop User Guide." https://helpx.adobe.com/photoshop/user-guide.html.

[27] Adobe, "Illustrator User Guide." https://helpx.adobe.com/illustrator/user-guide.html.

[28] Hugging Face, "Hugging Face Documentation." https://huggingface.co/docs.