TASK 3 REPORT

AI-Based High-Level to Low-Level Architecture Conversion Tool

1. Introduction

The objective of Task-3 was to design and implement an AI-based automation tool that converts high-level business requirements into detailed low-level technical specifications.
In real-world software engineering, this translation process is typically manual, time-consuming, and highly dependent on experienced architects. The goal of this task was to simplify and partially automate this process using artificial intelligence while preserving architectural correctness and clarity.

2. Objective

The primary objectives of Task-3 are:

- Analyze high-level business requirements written in natural language
- Decompose requirements into logical system modules
- Generate database schemas and data models
- Design REST API endpoints
- Produce structured pseudocode describing system behavior
- Output a complete low-level technical specification document

3. System Overview

The system implements an automated architecture generation pipeline that transforms a single business requirement into a structured technical design document.

Pipeline Flow:
High-Level Requirement → Requirement Analysis → Module Identification → Schema Design → API Design → Pseudocode Generation → Technical Specification Document

The output is a developer-ready low-level design document in Markdown format.

4. Architecture Design

The system is organized into modular components, each responsible for a specific transformation step.

4.1 Requirement Analyzer

This component accepts a high-level business requirement as input and performs:

- Input validation
- Text normalization
- Structural preparation for downstream processing

At the current stage, the requirement input is hardcoded to demonstrate deterministic system behavior. The architecture supports future extension for dynamic user input via CLI, web UI, or API.

4.2 Module Generator

This component breaks the analyzed requirement into logical system modules such as:

- User Management
- Core Business Logic
- Payment or Transaction Handling
- Tracking and Notification Systems

The output ensures clear separation of concerns and forms the backbone of the system architecture.

4.3 Schema Generator

Based on identified modules, this component generates normalized database schemas including:

- Entity definitions
- Primary keys
- Foreign key relationships
- Transaction-oriented tables

The schemas follow industry-standard relational database design practices.

4.4 API Generator

This module generates RESTful API endpoints for each system module.
Each endpoint clearly defines:

- HTTP method
- Resource path
- Functional responsibility

The APIs are designed to support scalable, service-oriented architectures and microservice compatibility.

4.5 Pseudocode Generator

The pseudocode generator produces detailed control flow logic describing:

- User authentication
- Order lifecycle management
- Payment retries and failure handling
- Asynchronous delivery tracking
- Event-driven notifications

This enables developers to clearly understand system behavior before implementation.

5. Technology Selection and Justification

5.1 Use of AI (Google Gemini)

Google Gemini is used to interpret natural language requirements and generate structured technical outputs.

Technical reasons for choosing Gemini:

- Strong natural language understanding
- Ability to infer architectural patterns from abstract descriptions
- Consistent structured output generation
- Rapid prototyping capability

The AI acts as a reasoning engine, not just a text generator.

5.2 Alternative Approaches Considered

Other possible approaches include:

- Manual system design by architects
- Rule-based requirement parsers
- UML modeling tools
- Static architecture templates
- Requirement-to-design mapping via enterprise tools

However, these approaches:

- Require high manual effort
- Lack adaptability to varied requirements
- Are not suitable for automation at scale

5.3 Why AI-Driven Automation Was Chosen

AI-based automation was selected because:

- Business requirements are often ambiguous and unstructured
- AI can reason contextually rather than relying on rigid rules
- The system can scale to multiple domains without rewriting logic
- It reduces time from requirement to technical design significantly

This approach aligns with modern software engineering workflows and AI-assisted development practices.

6. Output Format

The final output is generated as a Markdown document containing:

- Business requirement summary
- Module breakdown
- Database schemas
- REST API definitions
- Detailed pseudocode

Markdown was chosen because it is:

- Human-readable
- Developer-friendly
- Easily convertible to PDF or documentation platforms

7. Expected Outcome

The system successfully converts high-level business requirements into a comprehensive low-level technical specification with minimal manual intervention.
The generated document can be directly used by developers, architects, or technical reviewers as a foundation for system implementation.