TED UNIVERSITY

SENG 492 Senior Design Project 2
MyGen AI Services Super App
Detailed Design Report
15.03.2025

Team Name: DBU

Team Members:

Deniz ÖZCAN, 33577146512, Software Engineering

Betül Ülkü YURT, 11056264926, Software Engineering

Umut ŞAHİN, 11597931646, Software Engineering

Supervisor:

Emin Kuğu

Jury Members:

Tansel Dökeroğlu

Kasım Murat Karakaya

1. Introduction

1.1 Purpose

This document provides a comprehensive and detailed design for the MyGen AI Services Super App. It elaborates on the technical architecture, class structures, object interactions, and workflow patterns that will guide the implementation phase. The purpose is to offer developers and stakeholders a clear understanding of how system components interact to fulfill the requirements specified in the Software Requirements Specification (SRS).

1.2 Overview

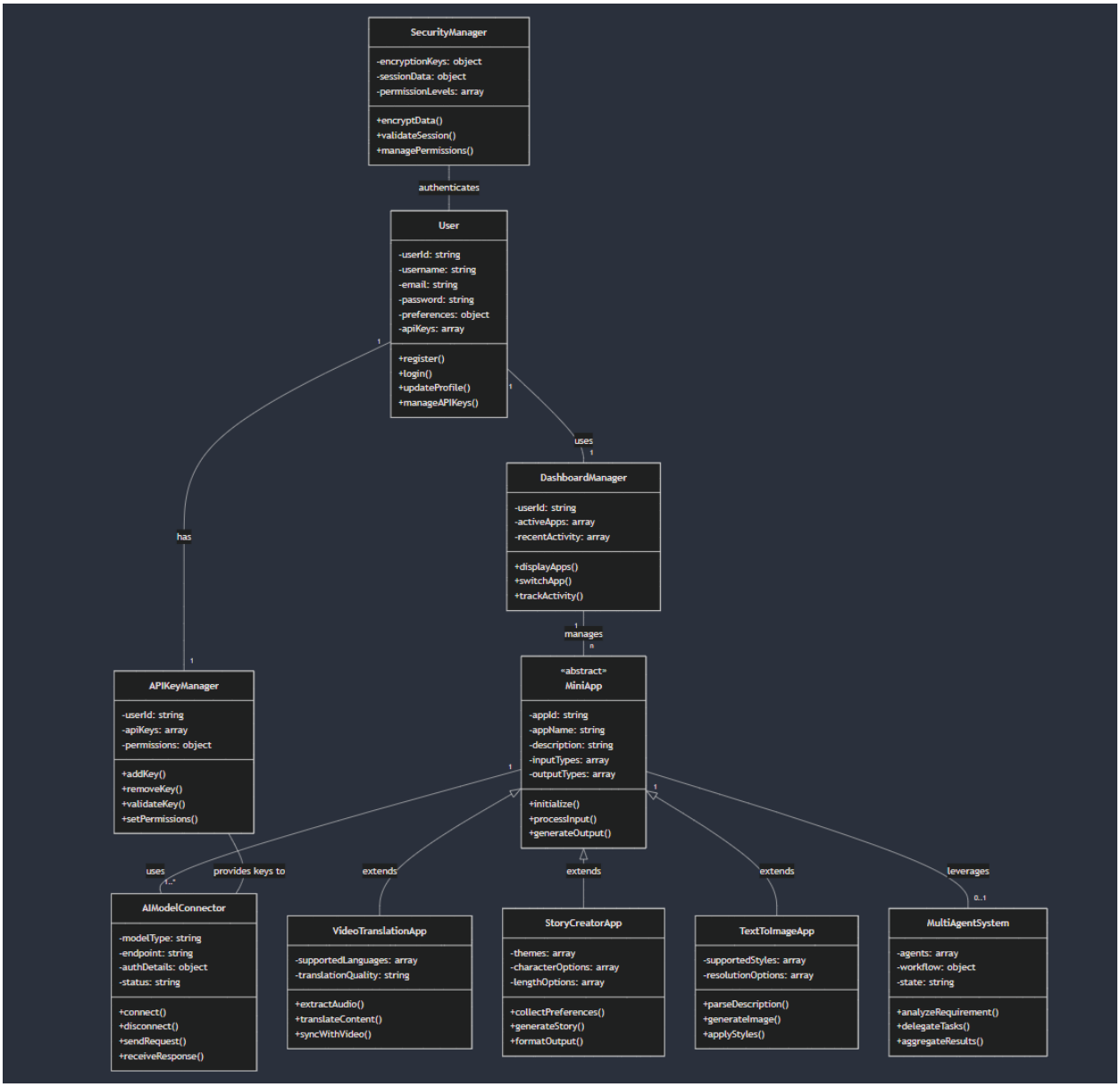
The MyGen AI Services Super App is a unified web and mobile application platform hosting multiple AI-powered mini-apps. The system allows users to leverage various AI models through a single interface, either using preset configurations or their own API keys. The platform emphasizes accessibility, personalization, innovation, and security while providing diverse functionalities such as video language translation, audio document creation, story generation, and more.

Key technologies used include:

- **Backend**: FastAPI with Python
- **Frontend**: React JS
- **AI Integration**: Multiple model support including OpenAI, Gemini, and Llama

2. Class and Object Models

2.1 Class Diagram



Core Classes and Responsibilities:

User Class

- **Attributes**: userId, username, email, password, preferences, apiKeys.
- **Methods**: register(), login(), updateProfile(), manageAPIKeys().
- **Responsibility**: Manages user authentication and profile information.

APIKeyManager Class

- **Attributes**: userId, apiKeys, permissions.
- **Methods**: addKey(), removeKey(), validateKey(), setPermissions().
- **Responsibility**: Handles the secure storage and management of user API keys.

MiniApp (Abstract Class)

- **Attributes**: appId, appName, description, inputTypes, outputTypes.
- **Methods**: initialize(), processInput(), generateOutput().
- **Responsibility**: Defines the common structure for all mini-apps.

AIModelConnector Class

- **Attributes**: modelType, endpoint, authDetails, status.
- **Methods**: connect(), disconnect(), sendRequest(), receiveResponse().
- **Responsibility**: Establishes and manages connections to different AI models.

MultiAgentSystem Class

- **Attributes**: agents, workflow, state.
- **Methods**: analyzeRequirement(), delegateTasks(), aggregateResults().
- **Responsibility**: Orchestrates multiple AI agents to enhance response quality.

VideoTranslationApp Class (extends MiniApp)

- **Attributes**: supportedLanguages, translationQuality.
- **Methods**: extractAudio(), translateContent(), syncWithVideo().
- **Responsibility**: Handles video language translation functionality.

StoryCreatorApp Class (extends MiniApp)

- **Attributes**: themes, characterOptions, lengthOptions.
- **Methods**: collectPreferences(), generateStory(), formatOutput().
- **Responsibility**: Creates personalized bedtime stories.

TextToImageApp Class (extends MiniApp)

- **Attributes**: supportedStyles, resolutionOptions.
- **Methods**: parseDescription(), generateImage(), applyStyles().
- **Responsibility**: Converts text descriptions to images.

DashboardManager Class

- **Attributes**: userId, activeApps, recentActivity.
- **Methods**: displayApps(), switchApp(), trackActivity().
- **Responsibility**: Manages the central user interface for accessing mini-apps.

SecurityManager Class

- **Attributes**: encryptionKeys, sessionData, permissionLevels.
- **Methods**: encryptData(), validateSession(), managePermissions().
- **Responsibility**: Ensures system security and data protection.

2.2 Object Model

The object model illustrates how classes instantiate into objects and interact at runtime:

1. **User Authentication Flow**:
    - A User object creates session credentials.
    - SecurityManager object validates these credentials.
    - DashboardManager object configures the UI based on authenticated user.
2. **API Key Management Flow**:
    - User object requests to add a new API key.
    - APIKeyManager object securely stores and validates the key.
    - AIModelConnector objects use validated keys for service connections.
3. **Mini-App Execution Flow**:
    - DashboardManager object initiates a specific mini-app.
    - Concrete mini-app object (e.g., VideoTranslationApp) processes inputs.
    - MultiAgentSystem object enhances processing with specialized agents.
    - AIModelConnector objects facilitate communication with AI services.
    - Mini-app object returns processed results to the user interface.

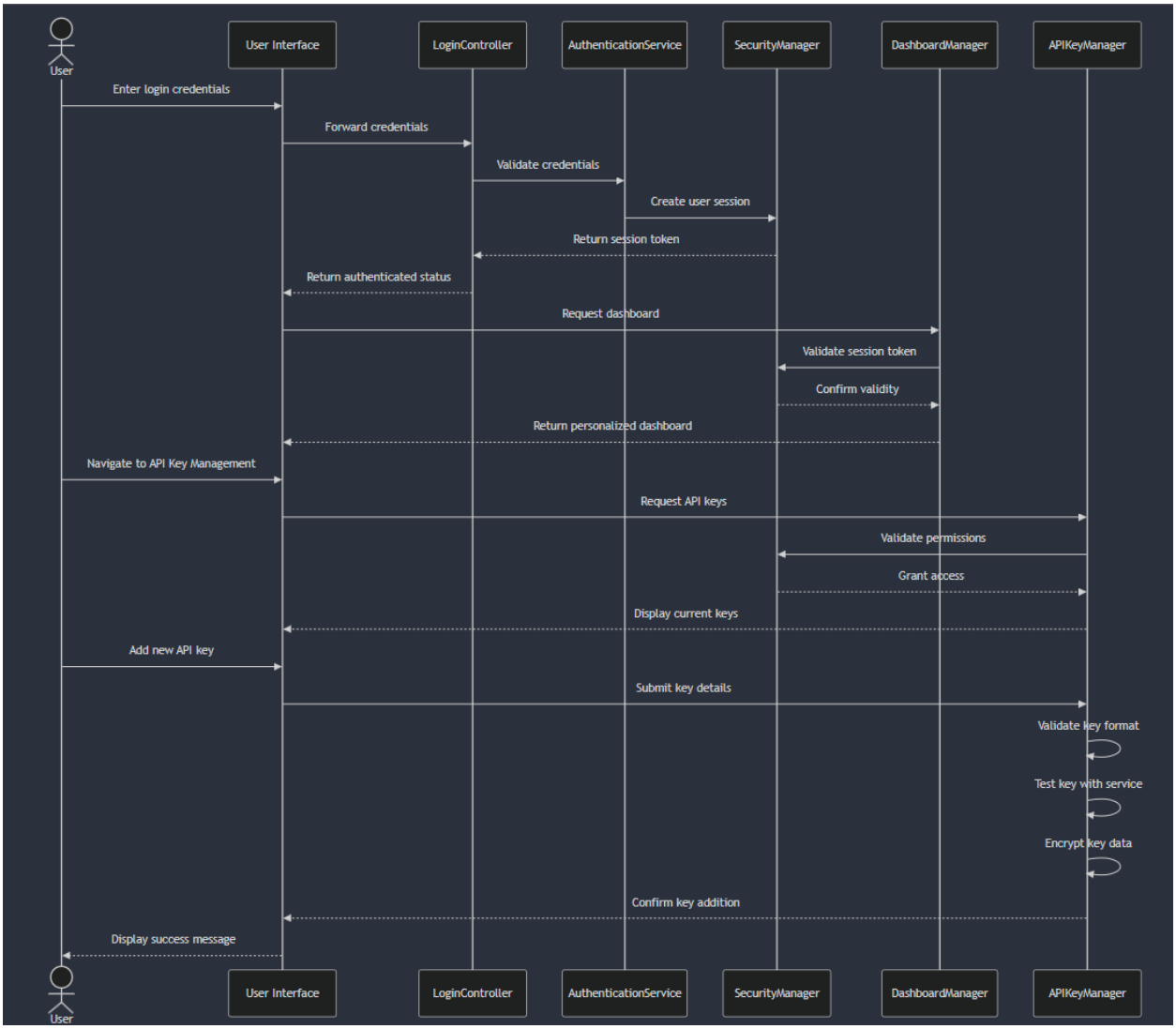
3. Design Sequence Diagrams

3.1 User Authentication and Session Management

Scenario:

A user logs into the platform, gains access to their dashboard, and manages their API keys.

Sequence Diagram:



Explanation:

1. User submits login credentials through the UI.
2. LoginController forwards credentials to AuthenticationService.

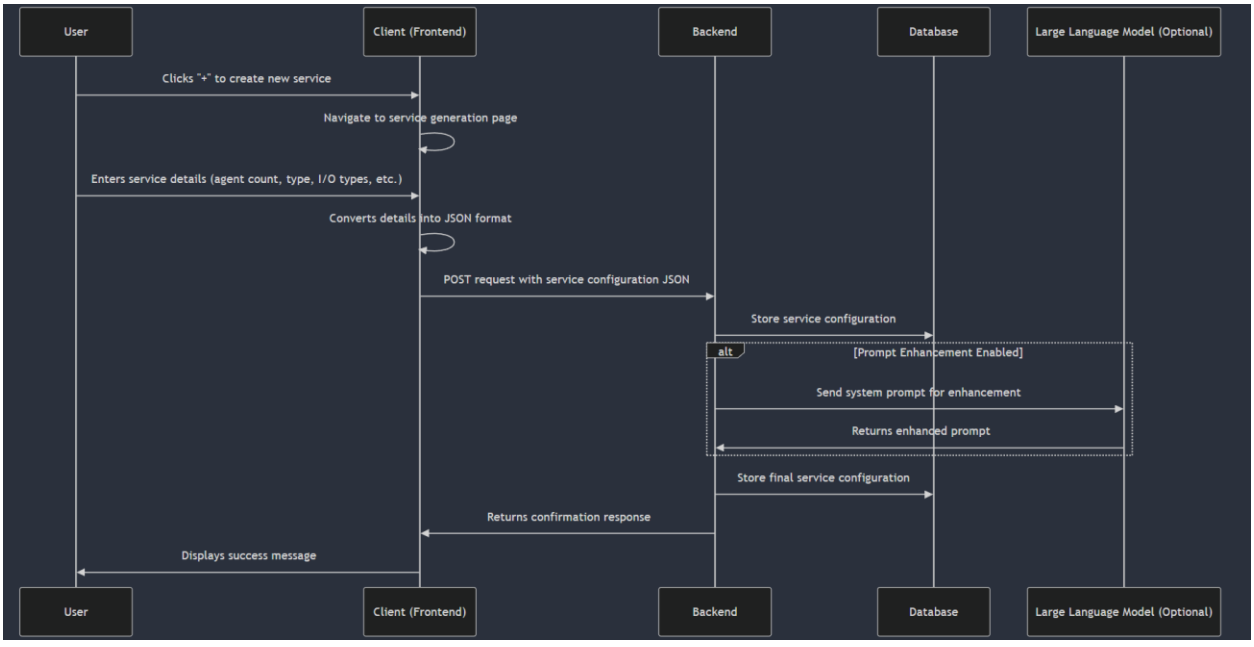
3. AuthenticationService validates credentials against User repository.
4. SecurityManager creates and manages the user session.
5. DashboardManager loads user preferences and recent activity.
6. UI displays dashboard to the user.
7. User navigates to API Key Management.
8. APIKeyManager retrieves and displays current keys.
9. User adds a new API key.
10. APIKeyManager validates and securely stores the key.
11. User receives confirmation of successful key addition.

3.2 Creating a New Mini Service

Scenario:

A user clicks the create new service button to navigate to generate service page and enters mini service details and configurations such as agent count, type of agents, input type, output type etc.

Sequence Diagram:



Explanation:

1. User clicks create new service button to navigate to the service generation page.
2. User enters the desired multi-agent architecture, agent count, input-output types, and other configurations.

3. If desired, user can request a prompt enhancement via checking the Enhance checkbox.
4. The clientside, converts users configurations into backend compatible JSON format.
5. A POST request sent to the respective endpoint with this JSON format body about configurations of the service.
6. The backend creates the service, if prompt enhancement is on the first the system prompt and the service information is sent to the LLM before.
7. The service is stored in the database for later usage.

3.3 Using a Mini Service

Scenario:

A user lists the mini services that are available and clicks one of them. In the service's page users can see information regarding the service such as input types, output types, expected average token usage and average cost (if applicable). User can provide the required fields and hit start to start his processing with this service. Later some time, once the processing is all complete, the output will be visible to the user.

Sequence Diagram:

Explanation:

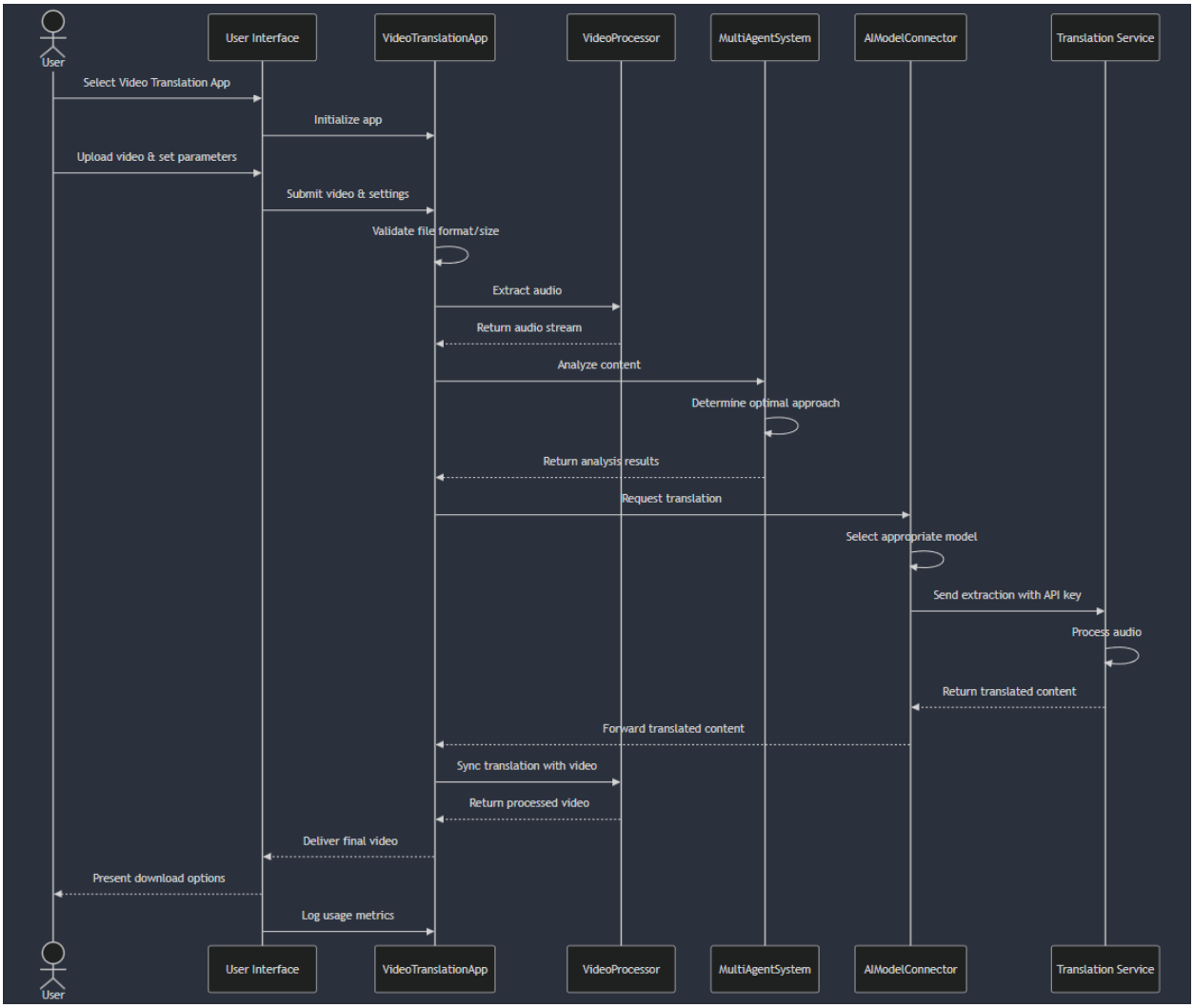
1. User navigates to the service page.
2. User clicks on the desired service and navigates to its page.
3. The service contents are served to the client in a dynamic way from the database.
4. User can view details of the service and provide required fields to use this service.
5. If applicable, user's API key retrieved from the database, and process gets initiated.
6. After processing is done, user can view the output of the process in the My Processes tab.

3.4 Video Language Translation Process

Scenario:

User uploads a video file and requests translation from English to Spanish.

Sequence Diagram:



Explanation:

1. User selects VideoTranslationApp from the dashboard.
2. User uploads a video file and specifies language parameters.
3. VideoTranslationApp validates file format and size.
4. VideoProcessor extracts audio from video.
5. MultiAgentSystem analyzes content for optimal translation approach.
6. AIModelConnector selects appropriate AI model based on content type.

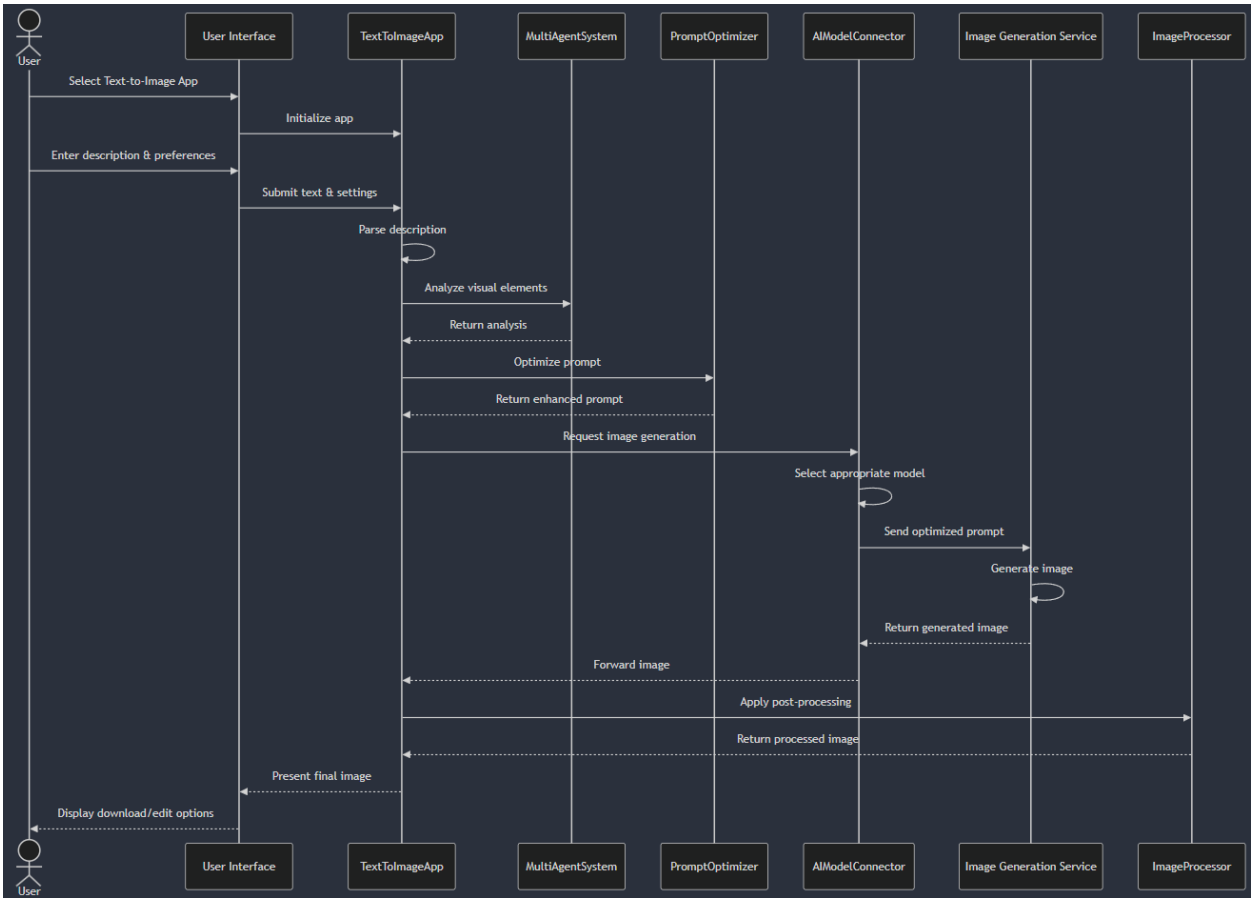
7. AIModelConnector sends extraction to translation service using the user's API key.
8. Translation service processes audio and returns translated content.
9. VideoProcessor synchronizes translated audio with original video.
10. VideoTranslationApp generates final video with translations.
11. UI presents download options to user.
12. System logs usage metrics for user account.

3.5 Text-to-Image Generation Process

Scenario:

User writes a prompt like, “A mountain landscape at sunset, with an orange-purple sky and a lake reflecting the colors.”, then AI model analyzes the keywords (mountain, sunset, orange-purple sky, lake, reflections), generates an image matching the description.

Sequence Diagram:



Explanation:

1. User selects TextToImageApp from dashboard.
2. User enters description and selects style preferences.
3. TextToImageApp parses and refines the description.
4. MultiAgentSystem analyzes description for visual elements and style requirements.
5. PromptOptimizer enhances description for better image generation.
6. AIModelConnector selects appropriate image generation model.
7. AIModelConnector sends optimized prompt to AI service.
8. AI service generates image based on description.
9. ImageProcessor applies post-processing effects if requested.
10. TextToImageApp presents generated image to user.
11. User downloads or edits the image.

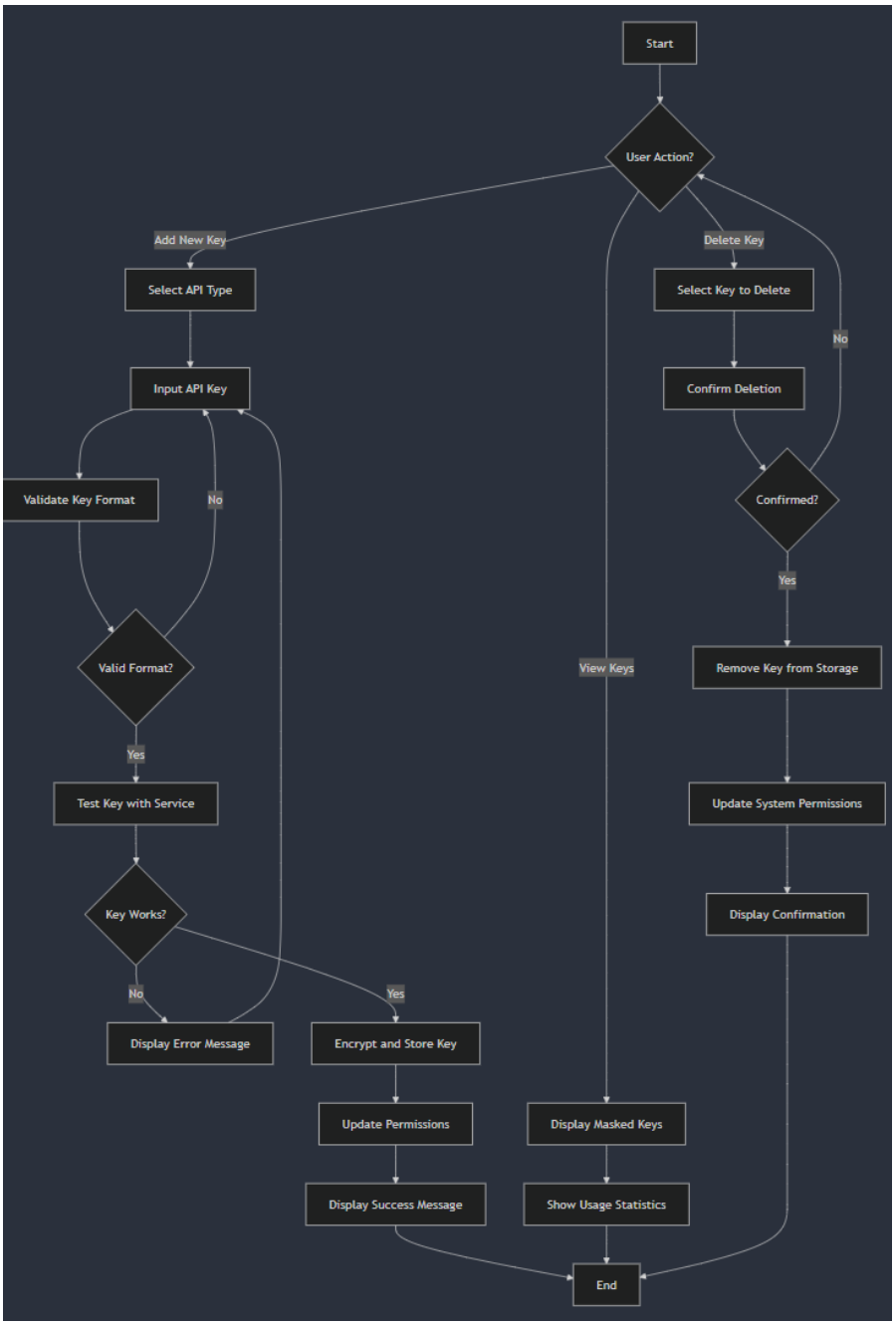
4. Activity Diagrams

4.1 API Key Management Workflow

Process Description:

This workflow details how users manage their API keys within the system.

Activity Diagram:



Explanation:

1. User logs in and navigates to API Key Management section.
2. User chooses to add, view, or delete API keys.
3. For adding:
    - User selects API type (OpenAI, Gemini, etc.).
    - User inputs API key.

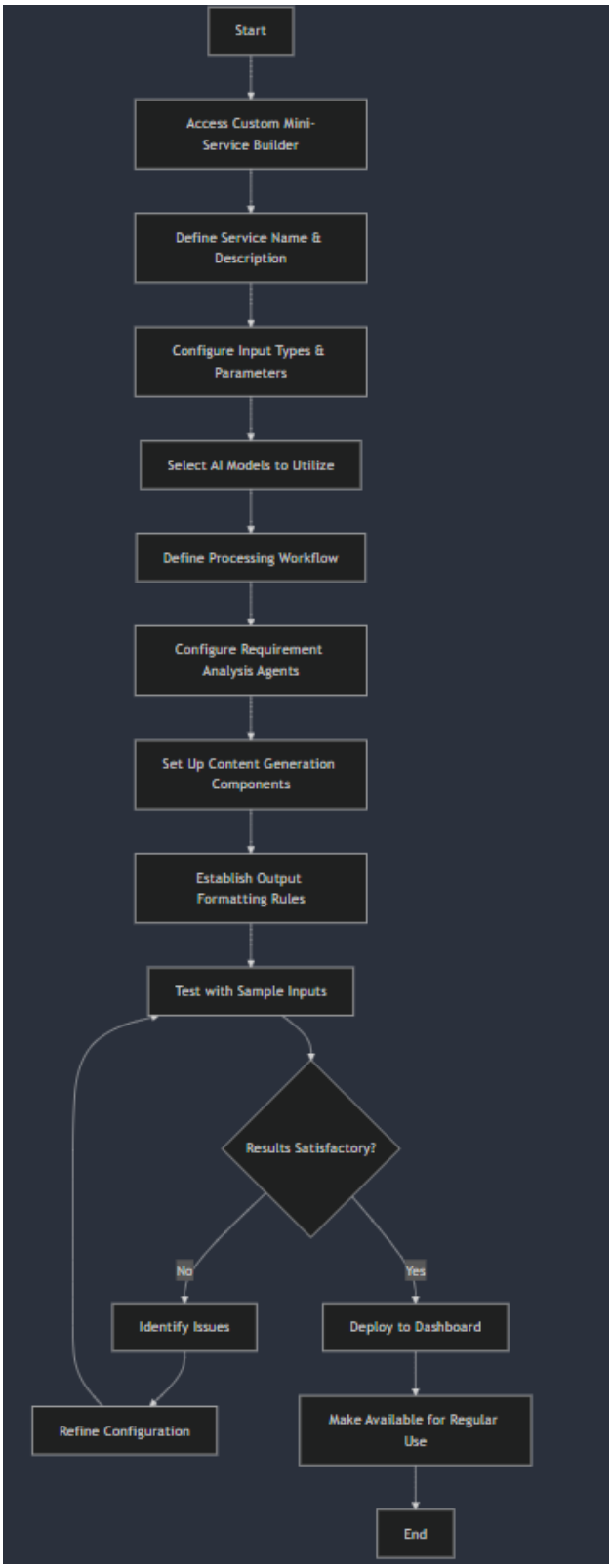
    - System validates key format.
    - System tests key with minimal API call.
    - System encrypts and stores valid key.
4. For viewing:
    - System displays partially masked keys.
    - User can see usage statistics.
5. For deletion:
    - User confirms deletion intent.
    - System removes the key from storage.
    - System updates permissions accordingly.

4.2 Custom Mini-Service Creation Workflow

Process Description:

This workflow shows how advanced users can create custom mini-services within the platform.

Activity Diagram:



Explanation:

1. User accesses the Custom Mini-Service Builder.
2. User defines the service name and description.
3. User selects the starting input type (text, image, audio, etc.), this determines which agents can be used at the beginning of the workflow.
4. User adds agents to the workflow in a sequential order, each agent processes the output from the previous agent.
5. User defines the processing workflow:
    - Configure requirement analysis agents.
    - Set up content generation components.
    - Establish output formatting rules.
6. User tests the service with sample inputs.
7. User refines configuration based on test results.
8. User deploys the custom service to their dashboard.
9. System makes service available for regular use.

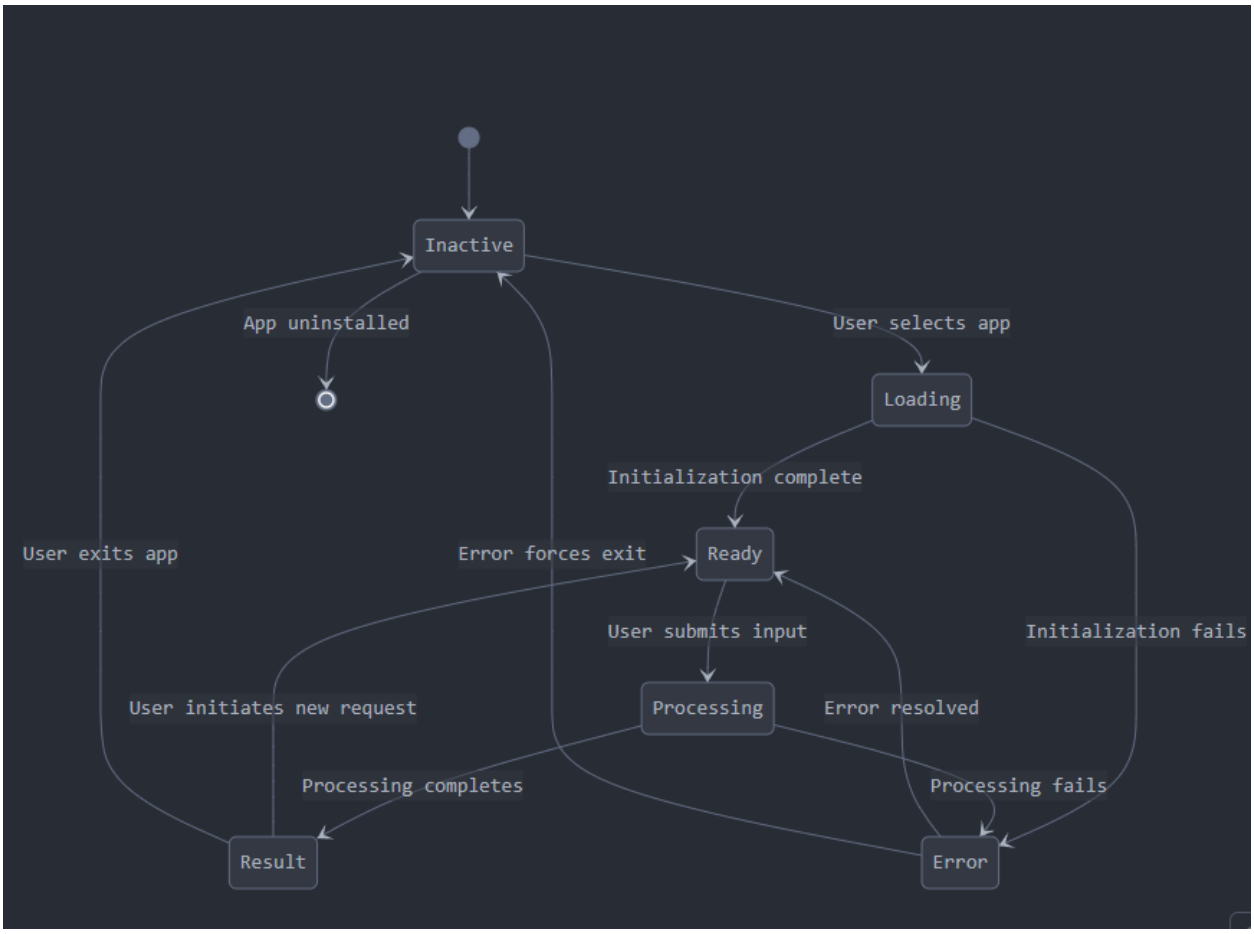
5. State Charts

5.1 Mini-App Lifecycle States

Description:

This state chart illustrates the different states a mini-app can be in and the transitions between these states.

State Chart Diagram:



Explanation:

- **Inactive**: Initial state when app is installed but not in use
    - Transitions to Loading when user selects the app
- **Loading**: Resources and configurations being prepared
    - Transitions to Ready when initialization complete
    - Transitions to Error if initialization fails

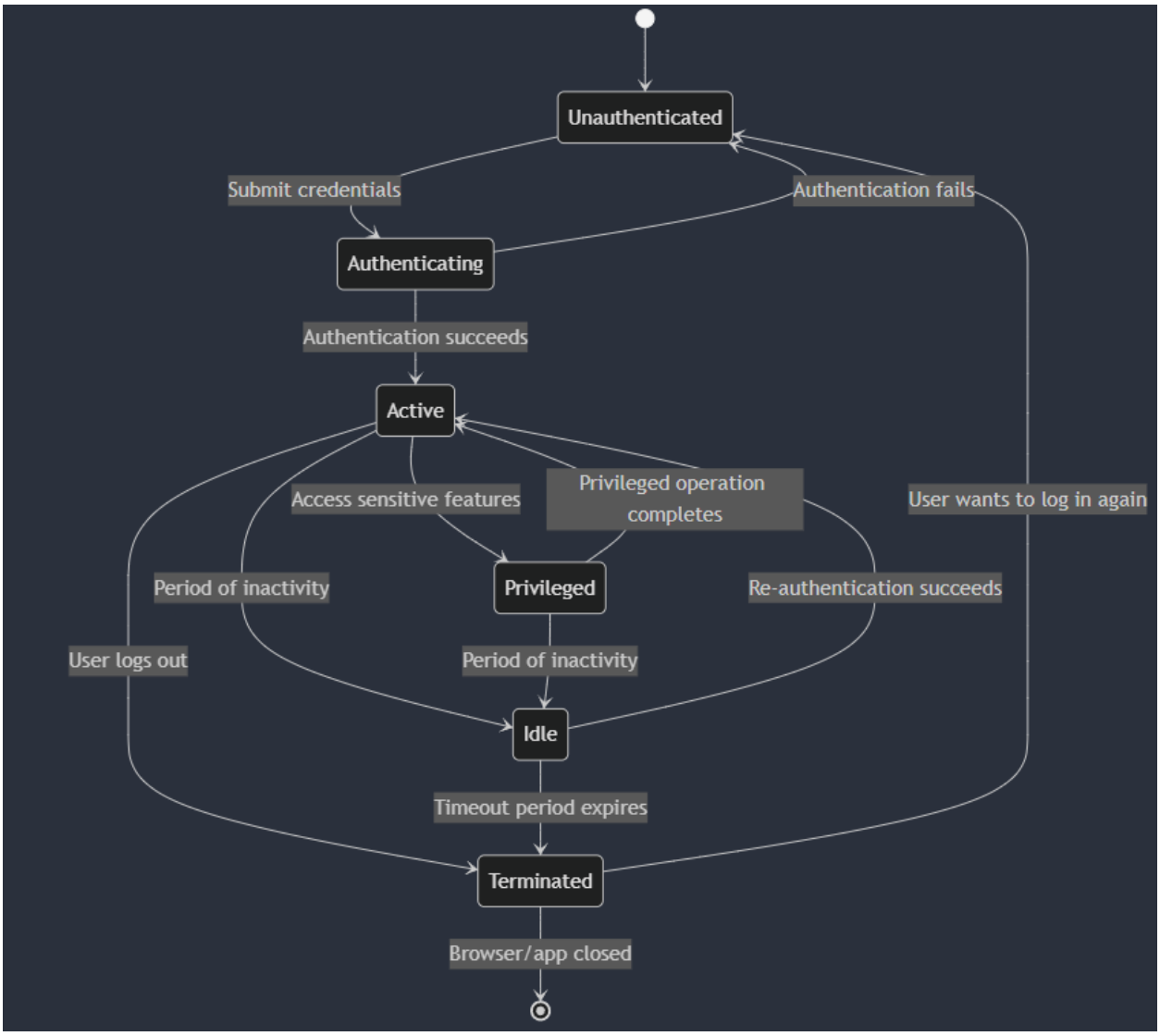
- **Ready**: App is prepared to accept user input
    - Transitions to Processing when user submits input
- **Processing**: App is actively working on user request
    - Transitions to Result when processing completes
    - Transitions to Error if processing fails
- **Result**: Output is available to the user
    - Transitions to Ready when user initiates new request
    - Transitions to Inactive when user exits the app
- **Error**: App encountered a problem
    - Transitions to Ready if error can be resolved
    - Transitions to Inactive if error forces exit

5.2 User Session State Chart

Description:

This state chart shows the possible states of a user's session within the system.

State Chart Diagram:



Explanation:

- **Unauthenticated**: Initial state before login.
  - Transitions to Authenticating when credentials submitted.
- **Authenticating**: System verifying user identity.
  - Transitions to Active if authentication succeeds.
  - Transitions to Unauthenticated if authentication fails.
- **Active**: User is logged in and can access features.
  - Transitions to Idle after period of inactivity.
  - Transitions to Privileged when accessing sensitive features.
  - Transitions to Terminated when user logs out.
- **Idle**: Session exists but requires re-authentication.

    - Transitions to Active after successful re-authentication.
    - Transitions to Terminated after timeout period.
- **Privileged**: Elevated access for sensitive operations.
    - Transitions to Active when privileged operation completes.
    - Transitions to Idle after period of inactivity.
- **Terminated**: Session has ended.
    - Transitions to Unauthenticated if user wants to log in again.

6. Database Design

Users Table: This table stores essential user information, including authentication credentials and email uniqueness enforcement.

id:	Unique identifier for each user.
username:	Stores the username (must be unique and not null).
hashed_password:	Securely stores passwords using SHA-256 encryption.
email:	Ensures unique identification for user communication and login purposes.

API Keys Table: This table maintains API keys associated with different AI providers.

id:	Unique identifier for each API key entry.
api_key	Stores the encrypted API key used for external service integration.
user_id:	Links the API key to a specific user.
provide:	Defines the AI provider (e.g., "gemini", "open-ai").

Process Table: Tracks user interactions with mini-services, including token consumption.

id:	Unique process ID.
mini_service_id:	References the mini-service being executed.
user_id:	Identifies the user initiating the process.
total_tokens:	JSON field storing total tokens consumed per process.

Mini Services Table: Stores metadata and execution data for mini-services, defining their workflows and usage statistics.

id:	Unique identifier for the mini-service.
name:	Service name.
description:	Brief details of the service.
workflow:	JSON structure defining process flow with interconnected nodes.
input_type:	Specifies the type of input accepted (e.g., "image", "text").
output_type:	Defines expected output type.
created_at:	Timestamp of service creation.
owner_id:	Links the service to its creator.
average_token_usage:	JSON field storing statistical token consumption per usage.
run_time:	Tracks the number of times the service has been executed.

Agents Table: Holds configuration details for AI-based agents interacting with the system.

id:	Unique identifier for the agent.
name:	Agent’s name.
system_instruction:	Defines the agent's system-level instructions.
agent_type:	Specifies the agent type (e.g., "gemini", "text2speech").
config:	JSON field storing additional configurations like temperature.
input_type:	Defines input format.
output_type:	Specifies the expected output format.
owner_id:	Links the agent to its creator.
created_at:	Timestamp of agent creation.

Logs Table: Maintains system logs for monitoring and debugging purposes.

id:	Unique log entry identifier.
ip_address:	Stores the IP address associated with the log entry.
type:	Categorizes logs as Info (0), Warning (1), or Error (2).
description:	Detailed log message.

Relationships and Data Integrity

- The **users** table is referenced by the **api_keys**, **process**, **mini_services**, and **agents**tables, ensuring that only registered users can create and manage these entities.
- The **process** table links **mini_services** and **users**, tracking service execution per user.
- **Mini_services** store workflow configurations as JSON, providing flexibility in defining execution logic.
- **Agents** store AI-related configurations, linking them to specific users.
- **Logs** store event tracking data, ensuring system monitoring and debugging capabilities.

7. Summary

This detailed design document outlines the architectural framework for the MyGen AI Services Super App. The design follows SOLID principles and embraces object-oriented programming to ensure modularity, scalability, and maintainability.

Key design choices include:

- Abstract MiniApp class allowing easy extension for new services
- Multi-agent system architecture for enhanced AI capabilities
- Secure API key management for user privacy and control
- Modular component design for independent development and testing

Future considerations and possible improvements:

- Implementation of a plugin architecture to allow third-party developers to create mini-apps
- Enhanced caching mechanisms to improve performance for frequently used services
- Expansion of supported AI models as new options become available
- Development of advanced analytics to help users optimize their AI usage and costs
- Implementation of federated learning capabilities for improved privacy and personalization

The design presented in this document provides a robust foundation for implementing the requirements specified in the SRS while maintaining flexibility for future enhancements.

8. References

Bit.ai. (n.d.). Software design document template. Bit.ai. Retrieved March 09, 2025, from https://bit.ai

Lucidchart. (n.d.). Software design document template. Lucidchart. Retrieved March 09, 2025, from https://www.lucidchart.com/pages/templates/software-design-document

Atlassian. (n.d.). Software design document: Tips & best practices. Atlassian. Retrieved March 10, 2025, from https://www.atlassian.com/work-management/knowledge-sharing/documentation/software-design-document

IEEE Computer Society. (2009). IEEE standard for information technology—Systems design—Software design descriptions (IEEE Standard No. 1016-2009). IEEE.

Wikipedia contributors. (n.d.). Software design description. In Wikipedia, The Free Encyclopedia. Retrieved March 10, 2025, from https://en.wikipedia.org/wiki/Software_design_description