# ”Dream Genie”
## Architecture Document

Authors:
Ben Eli benelashvili@gmail.com
Daniel Bronfman daniel.bronfman2010@gmail.com
Tomer Pardilov tomerpardi@gmail.com

## I. Introduction

This document serves as an architectural guide for the ”Dream Genie” application. It provides an in-depth exploration of the system's architecture, deployment strategies, technologies employed, data storage mechanisms, and dependencies. Our objective is to offer a comprehensive understanding of the inner workings of ”Dream Genie”.

## II. ”What is Dream Genie?”

Dream Genie is an application that aims to allow it's users to experience their dreams in a new light. The users talk about a dream they had recently, the software identifies key scenes in the dream, and creates an artistic representation of each scene for the user.

*Objectives*

Our objectives with this project are:

- Facilitate creativity and stimulate memory among older adults by enabling them to recall and visualize their dreams.
- Possibly improve the overall mood and quality of life of older individuals, particularly for those with conditions such as Alzheimer's disease, by encouraging discussions about dreams.

## III. Architectural Overview

### A. System Architecture

The high-level architecture of the ”Dream Genie” system comprises several components/modules and their interactions.

- Voice/text input to generate dream scenes and associated images.
- GPT-3 is used to split the transcribed text into individual dream scenes based on context and meaning.
- For each dream scene, an image is generated using Stable Diffusion instance hosted by us, using a web-API.
- The resulting dream scenes and associated images can be presented to the user for review or analysis, providing insight into the user's dream and promoting self-reflection.

### B. Deployment Architecture

The software will be deployed using specific server configurations and network topology. Our deployment strategy encompasses a blend of cloud-based and local resources to ensure efficient performance and security.

**Azure-Based Server Deployment**

Our primary server infrastructure will be hosted on Microsoft Azure. We will employ Docker Compose to manage three distinct containers within this environment:

1) **Gunicorn Container:** This container will serve as the application server, responsible for handling incoming requests and managing the core application logic.
2) **Redis Container:** Redis will play a pivotal role in our architecture by serving as the storage backend for the Flask-Session library. It will securely store user data, identified by session IDs, to ensure a seamless user experience.
3) **Nginx Container:** Nginx will be employed for load balancing, following the recommendations provided by Gunicorn. This setup ensures high availability and optimal distribution of incoming requests.[1]

**Interactions and Communication**

Our server will communicate with clients through Flask-SSE (Server-Sent Events), facilitating real-time updates. This mechanism allows the server to publish events to which clients can subscribe, creating a dynamic and responsive user experience. We may further explore this approach in the ”Decisions” section, delving into potential challenges and solutions related to WebSocket implementations.

**Local Server with GPU Support**

For tasks requiring substantial computational power, such as the Stable Diffusion (SD) module, we have deployed servers locally. These servers are equipped with GTX 1080Ti GPUs, accessible through BIU DSI. Like our Azure-based deployment, these servers also operate within Docker containers to streamline management and scalability.

**Secure External Access**

To provide secure external access to our servers, we have implemented Ngrok tunnels. These tunnels allow authorized users to access our services remotely while maintaining robust security measures. Additionally, we have deployed the Whisper endpoint/service in a Docker container, leveraging Flask as the handler for this component. We have chosen the faster-Whisper implementation due to its exceptional speed. Access to this service is also facilitated via Ngrok tunnels, with an added layer of security provided by HTTP authentication.

Our deployment architecture balances cloud-based flexibility with localized computational power, ensuring both scalability and security in serving our users and clients effectively.

### *C. Technologies Used*

Key technologies, frameworks, and tools employed in the architecture include:

- Python
- Flask
- Gunicorn as a WSGI
- Docker
- GPT-3 (text-davinci-03) by OpenAI
- Stable Diffusion
- Whisper by OpenAI
- Bootstrap

[1]https://docs.gunicorn.org/en/stable/deploy.html

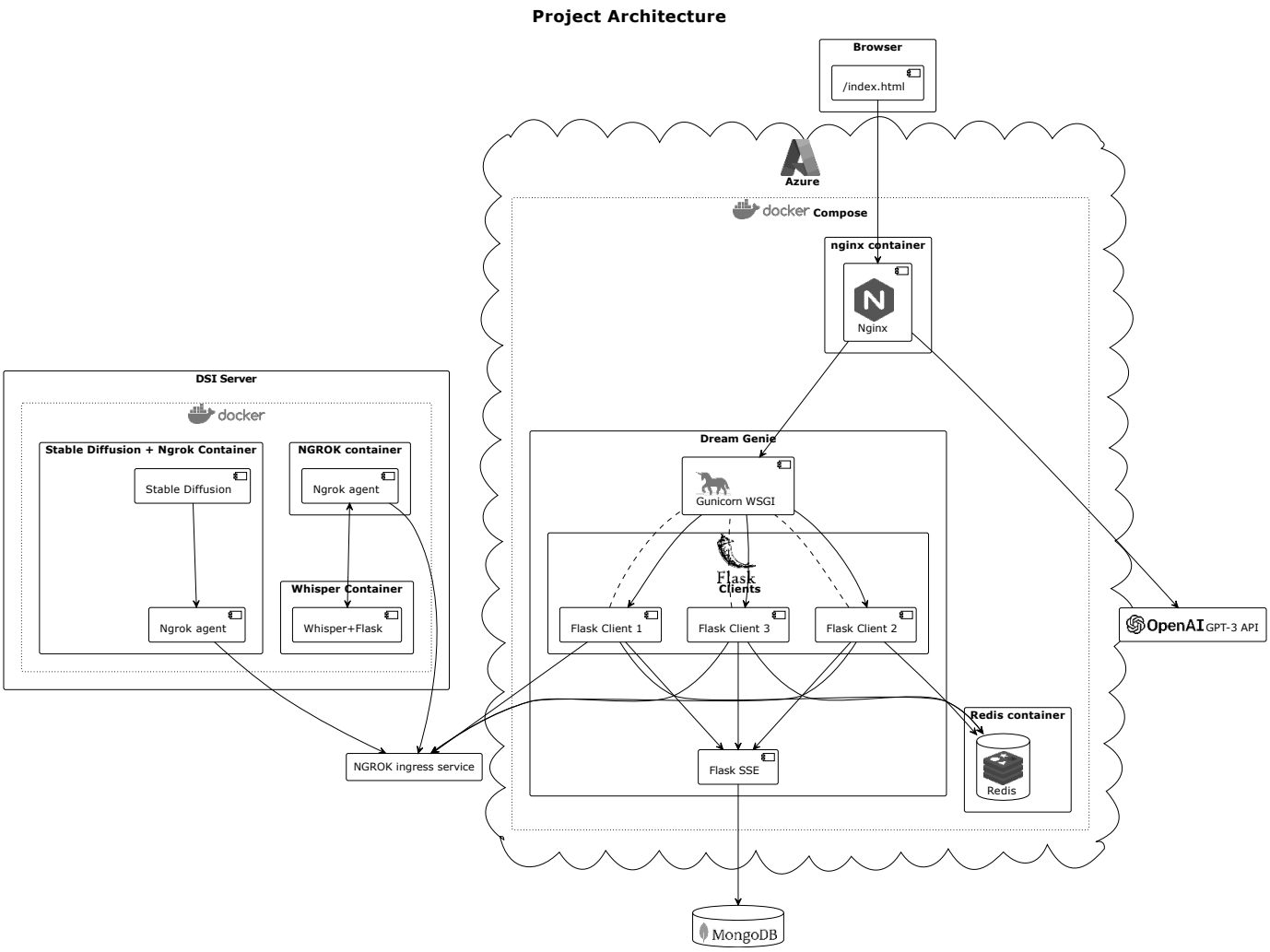


Fig. 1. Deployment Architecture

## IV. System Components

### A. User Interaction local GUI (app_gui.py)

The Dream Genie application features a user interface that allows voice input and displays prompts for validation, as well as progress indicators. It includes code for getting voice input from the user and transcribing the audio using Whisper, an open-source speech recognition model. Whisper is self-hosted and run as a service using a Flask server, with the medium model size selected for optimal transcription speed and accuracy ratio.

### B. Flask Web application (app.py)

This module is a Flask web application that provides a user interface for a Dream Genie service. The service allows users to upload an audio file, which is then transcribed into text using a separate whisper service. The transcribed text is then used to generate a dream scene, which is displayed to the user along with the original audio file. The user can then rate the image, scene, and overall experience, and provide feedback. The module contains several routes for handling user requests, including processing the audio file, displaying the feedback form, and submitting feedback to a MongoDB database. The module also uses eventlet to spawn a separate thread for sending the audio data to the whisper service, in order to avoid blocking the main thread.

### C. User Interaction Web GUI

The User Interaction Web GUI of the Dream Genie web application consists of several HTML pages, with the logic implemented in Javascript, with event listeners:

- **Gallery Page (gallery.html):** This central page displays dream images generated in real-time through a Bootstrap carousel. Users can interrupt image generation with a ”Go Back” button and provide feedback via a ”Give Feedback” button after all images are generated.
- **Home Page (index.html):** The Home Page allows users to input dreams, record audio, and initiate processing, with a loading spinner. JavaScript manages audio and data submission.
- **Feedback Page (feedback.html):** The Feedback Page captures user feedback, including age, gender, AI familiarity, ratings, and comments, submitted to the ”/submit_feedback” endpoint via POST.

These interfaces offer a seamless and user-friendly experience for interacting with the Dream Genie system.

### D. Image Generation (send_prompt.py)

The Dream Genie project uses Stable Diffusion for image generation, with a prompt sender that creates a JSON payload with all the necessary parameters for image generation and sends it to Stable Diffusion via the API. The automatic1111 webui for Stable Diffusion implements an easy-to-use API for image generation and supports multiple user-created scripts. The webui allows users to save preferred styles for future prompts and offers the flexibility to select which Stable Diffusion checkpoint to use for optimal image quality. The automatic1111 webui is in active development, with new features and updates being added regularly, and uses ngrok tunneling to expose local ports.

### E. Dream Analyzer (gpt_call.py)

The Dream Genie project uses OpenAI’s GPT3 language model to segment users dreams into scenes, with a script handling the call. The script allows for testing the dream separation on a corpus of sourced dreams and loading manually separated dreams as examples to improve the quality of separation. It builds a textual prompt to GPT3 based on the input, adding necessary commands and samples, and allows us to choose which GPT3 model to use (davinci or curie). The script separates the output into a list of scenes and returns it.

### F. Miscellaneous Utilities (utils.py)

This component provides utilities to obtain public URLs for services and handles ngrok tunneling for secure connections.

# V. Data Flow

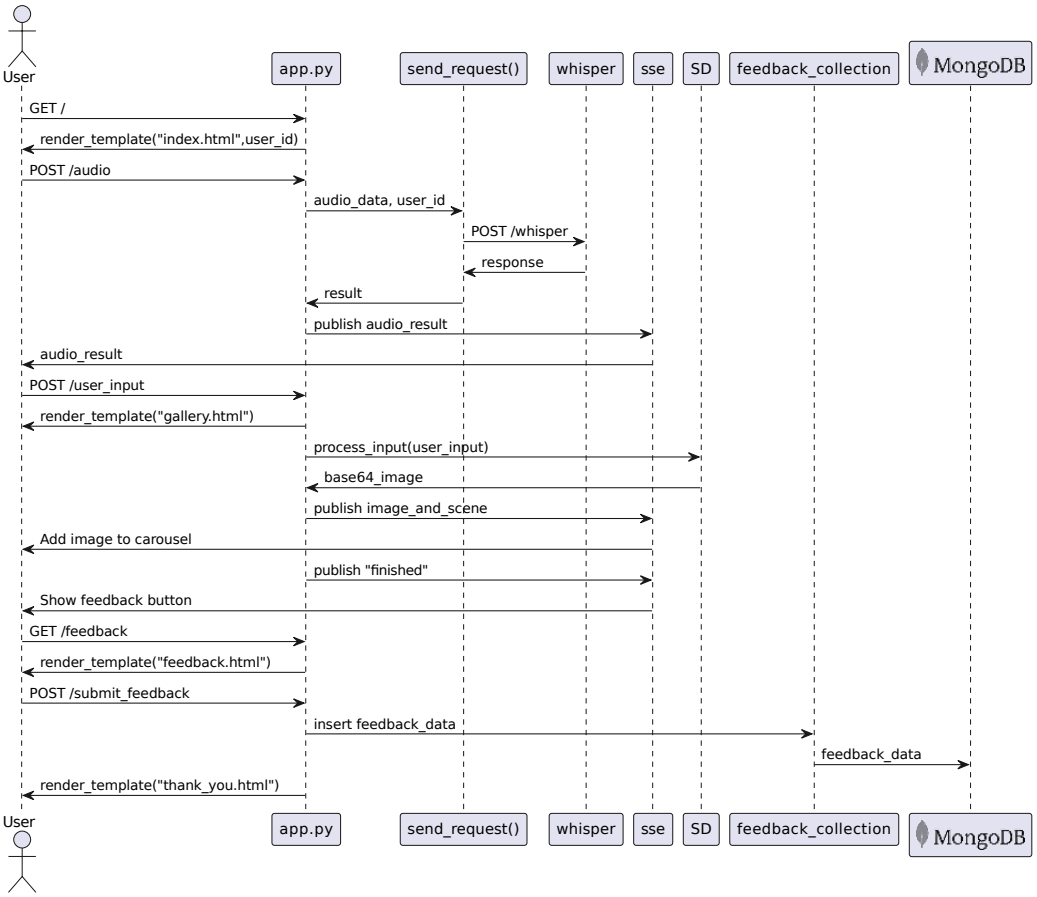


Fig. 2. Flow Chart of User interaction with the Web UI

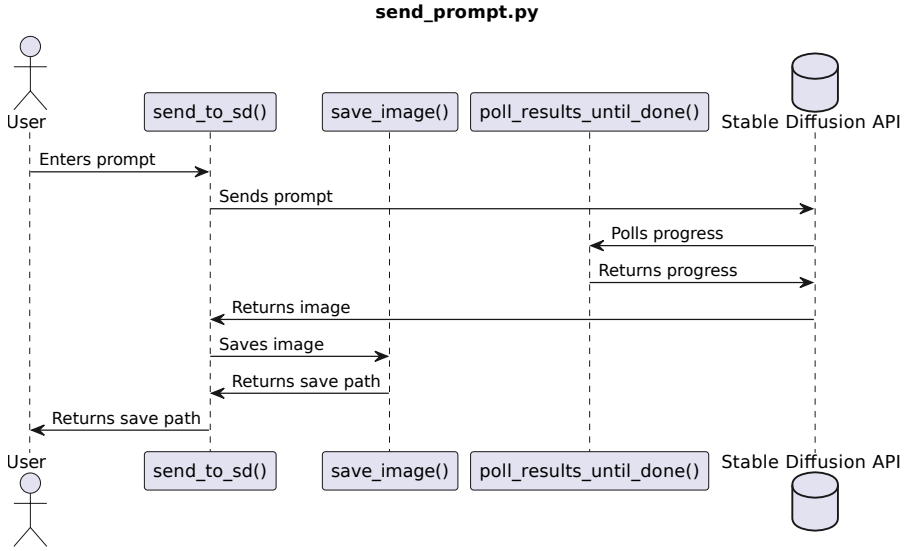


Fig. 3. Flow Chart of send_prompt.py

## VI. Data Storage - MongoDB

Data within the "Dream Genie" system is stored and managed using the MongoDB NoSQL database. We planned to preform an experiment using "Prolific" in order to collect user feedback about the experience of using Dream Genie, and the following information is intended for that purpose.

### A. Feedback Collection Schema

The MongoDB feedback_collection schema includes the following fields:

- **User ID**: A unique identifier for each user, automatically generated by the system.
- **Image Rating**: User's rating of how closely the generated images match the dream scenes on a scale of 1 to 5
- **Scene Rating**: User's rating of how well the generated scenes represent their dreams on a scale of 1 to 5
- **Experience Rating**: User's overall rating of their experience with Dream Genie on a scale of 1 to 5
- **Age (Optional)**: User's age, provided optionally, typically within a realistic range
- **Gender**: User's gender
- **Familiarity**: User's self-reported familiarity with AI image generation, chosen from options like "No experience," "Somewhat knowledgeable," or "Well-versed."
- **Comments**: An open-text field where users can provide additional feedback or comments.

These fields are used to organize user feedback for system analysis and improvement.

## VII. Dependencies

The "Dream Genie" system relies on several external services, libraries, and APIs to function effectively. These dependencies are managed through the following installation procedures:

### A. System-Level Dependencies

*Note: We used a Docker container with those dependencies satisfied*

- **A UNIX working environment**
- **Python 3.9+**
- **PortAudio19**: PortAudio is an audio library that "Dream Genie" utilizes for audio input and output. To ensure compatibility and functionality, the system installs the 'portaudio19-dev' package.
- **Redis Server**: Redis is employed for various system functions, such as caching and background tasks. The system ensures its availability by installing the 'redis-server' package.

### B. Python Dependencies

- **Additional Python Packages**: Various Python packages are required to run different components of "Dream Genie." These packages, listed in the project's 'requirements.txt' files, are installed using the following commands:
  - `pip install -r DreamGenie/requirements.txt` ensures that the necessary packages for the core functionality of "Dream Genie" are installed.
  - `pip install -r DreamGenie/WebGui/requirements.txt` installs the specific requirements for the web server component of the system.

These dependencies are systematically managed to guarantee the proper operation of "Dream Genie" and to support its various functionalities.

# VIII. Appendices

## A. Appendix A: Dream Genie Project Repository

The Dream Genie project repository contains the complete source code and documentation for the Dream Genie application, including its architectural details. You can explore the repository and access project-related resources by visiting:

https://github.com/BDT2023/DreamGenie

## B. Appendix B: Faster-Whisper Repository

The Faster-Whisper repository contains the implementation of the Whisper speech recognition model optimized for speed. You can find the repository and its source code at the following link:

https://github.com/guillaumekln/faster-whisper

## C. Appendix C: Stable Diffusion (automatic1111) Repository

The Stable Diffusion (automatic1111) repository houses the implementation of Stable Diffusion, a method used for image generation in the Dream Genie project. You can access the repository and its codebase here:

https://github.com/AUTOMATIC1111/stable-diffusion-webui