Title of the project

“AI-Powered General Assistant Software”

Submitted By:

Shafin Rahman
ID: 213-16-591
Department of Computing and Information System
Daffodil International University

Supervised By:

Md. Mehedi Hassan
Lecturer (Senior Scale)
Department of Computing and Information System
Daffodil International University

Department of Computing and Information System
Daffodil International University
Dhaka, Bangladesh

Submission Date:14/10/2025

APPROVAL

This Project titled **"AI-Powered General Assistant Software"**, Submitted by Shafin Rahman, ID No: 213-16-591 to the Department of Computing and Information Systems, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Computing & Information Systems and approved as to its style and contents. The presentation has been held on 14-10-2025.

BOARD OF EXAMINERS

Md Sarwar Hossain Mollah — **Chairman**
Associate Professor and Head
Department of Computing & Information Systems
Faculty of Science & Information Technology
Daffodil International University

Md. Nasimul Kader — **Internal Examiner**
Assistant Professor
Department of Computing & Information Systems
Faculty of Science & Information Technology
Daffodil International University

Md. Mehedi Hassan — **Internal Examiner**
Lecturer (Senior Scale)
Department of Computing & Information Systems
Faculty of Science & Information Technology
Daffodil International University

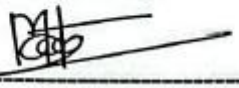
Ahmed Saif Reza — **External Examiner**
Managing Director & Chief Technology Officer
Medico Bio Limited

©Daffodil International University

Declaration

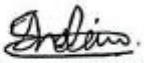
I hereby declare that; this project has been done by me under supervision of **Md. Mehedi Hassan, Lecturer (Senior Scale)**, department of Computing and Information System (CIS) of Daffodil International University. I am also declaring that this project or any part of there has never been submitted anywhere else for the award of any educational degree like, B.Sc., M.Sc., Diploma or other qualifications.

Supervised By



Md. Mehedi Hassan
Lecturer (Senior Scale)
Department of CIS
Daffodil International University

Submitted By



Name: Shafin Rahman
ID: 213-16-591
Department of CIS
Daffodil International University

©Daffodil International University

Acknowledgement

I would like begin with expressing my gratitude to Almighty Allah for giving me this opportunity. Without His guidance, I wouldn't have learn and grow and haven't been able to complete this project.

I would like to show appreciation to my academic supervisor, Md. Mehedi Hassan. His support and instruction have made this project possible. His guidance has truly helped in my professional development. He has always been there to provide the right advice that help me make the best decisions and inspired me to overcome all the challenges.

I would also like to show my heartfelt gratitude to Daffodil International University. My university has provided me the resources that I needed for making this project possible. Special thanks to the Department of Computing and Information System for their continuous support, valuable guidance, and encouragement throughout my study.

©Daffodil International University

Abstract

This software names as AI-Powered General Assistant Software aims to be a stand-alone desktop assistant integrating Python, FastAPI, React, combining with both local (offline) and OpenAI-based models. The system provides a conversational interface that is capable of executing commands, managing files, browsing the web, reading on-screen content, and performing advanced reasoning tasks. Additionally, it has the ability to recognize face with DeepFace(python). The goal is to bridge the gap between AI's accessibility and user autonomy. Unlike commercial assistants (e.g., Siri or Google Assistant), this system operates independently on the user's machine, this ensures privacy and full control over users' local conversation. The software has a very simple chat base frontend and a strong logical backend that support both offline AI (Ollama) and cloud AI (OpenAI API). It maintains sessions as memory with recall ability that hold the user and AI conversation in one page. By simply using command user can generation files (.txt, .docx, .pdf, .pptx) in their local storage. It can act on voice command by calling the wake word(customizable). This assistant software can also be used to organized the storage with in few seconds. It can also send email with any type attachment. By using the python Deepfake, it can organize images by face. This works best on organizing images from a google drive and send them to the person that matches by face and mail them to their personal mail.

©Daffodil International University

Table of Contents

©Daffodil International University

©Daffodil International University

©Daffodil International University

©Daffodil International University

Table of Figure

©Daffodil International University

Table of Table

©Daffodil International University

Chapter 1 – Introduction

©Daffodil International University

1.1 Background

AI has become a necessity in our every day life. We use AI for explanation, summarize, or automate for many of our task. As the AI evolve the need for a personal AI assistant is also shows. AI assistants have evolved from simple chatbots to multi-functional digital partners. This project aims to create a **locally deployable AI assistant** that merges backend power with user-friendly UI design, targeting flexibility, data privacy. While existing assistants depend on constant internet connectivity and centralized data servers, they are unable to operate independently or integrate easily with user-specific workflows that is stored in user's personal device. That ease the gap between an online assistant and the need for a more local and personal hand on the wheel assistant. The system allows complete local execution while also have the room for API extensibility for OpenAI, enabling intelligent processing with or without internet access as need. This software is capable of harnessing any type of local AI model and perform the best depending on the user's local machine's capabilities. Furthermore, the face recognition capability gives this software the ability to handle image data. This opens up more doors as the software can access a drive, compare image with faces and send the only images to the person that match that person face.

©Daffodil International University

Chapter 2 – Initial Study

©Daffodil International University

2.1. Project Proposal

The proposed project, titled AI-Powered General Assistant Software, aims to develop an intelligent desktop assistant capable of performing both conversational and command-based tasks. That Chat system has both text and audio input. It understands our everyday natural language conversation. The assistant can do various automation such as open websites, send emails, read screens, manage files, read face and perform various other utility operations. In this project I used FastAPI, React, and a Python backend that is connected to local (Ollama) and OpenAI(APIs) engines. The software is designed for maximizing our personal productivity and give us a reliable and smooth human–computer interaction.

2.2. Background of the Project

As the day are passed more and more powerful AI tools are seed to be developed. We are relying on this tool more and more. Popular everyday assistants like Siri, Alexa, and ChatGPT have set a new standard for accessibility and support. However, most of them depend heavily on online, they almost always lack proper customization, or fail to be a standalone all in one complete system. In this project my aim was to make the vary bridge of creating a fully offline, customizable, and user-friendly assistant that can perform repetitive everyday operations on our personal system. The system combines the local processing power (better if user have GPU with AI Chip) with AI intelligence for providing automation and privacy all surrounding what user personally prefer.

2.3. Problem Area

Even though AI has shine in every sector in the past few years, there is a limitation that user face and that is their data security, no network usability and for free users a limit on continuing conversation on a single chat. This limitation may look simple but when faced it disconnect the user from accessing Any support from AI, make them insecure about their data and cut them off in the middle of conversation. Furthermore, professionals and students often require tools that can do multiple things beside just chat. So this problem has clearly creates the need for an assistant that is local with controlled data flow and support even there is no network available as well as do many things beside just chat.

2.4. Possible Solution

The possible solution that I proposed is a AI assistant that can run locally using user the processing power of user computer. In this time Ollama is the best for running a verity range of LLM models that are completely free. OpenAI API system is also there in case user needs to do highly complex task that are not processable with their localo machine. FastAPI, React, and Python are use to build the system and SQLite for lightweight data handling. The system offers a dual input as both text and voice interaction that is also capable of understanding many commands like “open GitHub,”

©Daffodil International University

“read screen,” or “send email”. It stores all of the user’s data locally using SQLite by creating sessions. This solved the privacy problem while maintaining the flow of chat sessions. With different modules for web browser automation, screen reading, voice activation, and controlled and secure API key storage, this assistant software provides an all round solution that combine the power of AI intelligence with desktop based functionality.

©Daffodil International University

Chapter 3 – Literature Review

©Daffodil International University

3.1. Discussion on Problem Domain Based on Published Articles

The problem domain of this project is based in various published articles. Intelligent virtual assistants (IVAs) and human–computer interaction (HCI), two articles that I read have been widely discussed in recent literature. According to the research hold by Hoy (2018) the evolution of voice assistants like Siri, Alexa, and Google Assistant has been heavily optimized with the way user from kids to elders interact with them. They use more friendly type of conversation over traditional input methods. However, studies by McTear (2020) and Liao et al. (2021) highlights various challenges that come up with them including there must be an internet connection, user's privacy concerns and lack of task that user expect it to do. All of the today's existing assistants are cloud-based and need an consistent online connection, that limit on integration with local desktop operations. Along with that users feels handcuffed when sees there is few to none customizable option.

3.2. Discussion on Problem Solutions Based on Published Articles

I found that in recent studies have proposed multiple solutions to improve virtual assistant performance. Research by Këpuska and Bohouta (2018) talks the importance of context aware AI that can understand natural language processing with system control. Similarly, work by Zhou et al. (2022) showcase a possible hybrid AI model that integrate cloud based AI that rely on internet with local CPU-GPU computing to balance efficiency and privacy. Other studies I found that propose modular design frameworks that is upgradable as more functionality is planned to include. These modules are specific sectors like email, file management, and screen reading (Zamora, 2019), face recognitions. These findings strongly support the development of an assistants that not only behave like a human but also execute user commands when a certain task has to be done. By learning from this literature, the proposed solution I gave should apply FastAPI for backend structure, React for modern UI, and Ollama/OpenAI models for intelligence that complete the assistant who operates both locally and with options open for cloud APIs if needed.

3.3. Comparison of Three/Four Leading Solutions

a. Best Features

- **Google Assistant**: It has deep integration with the Google ecosystem, strong natural language understanding, and smart home control.
- **Amazon Alexa**: Offers a large third-party skill support and has compatibility with IoT devices.
- **Microsoft Cortana**: Integrates seamlessly with Windows applications and productivity tools.
- **Apple Siri**: Features strong voice recognition and privacy measures within the Apple ecosystem.

©Daffodil International University

b. Limitations

- All major assistants rely heavily on constant internet connectivity and cloud processing their linked company's ecosystem.
- Limited or no offline functionality and minimal local data control.
- Restricted customization(policy) — users cannot easily add or modify capabilities.
- Data privacy concerns as most assistants collect and store personal interactions remotely.
- Poor or no integration with non-ecosystem platforms or desktop automation tasks.

3.4. Recommended Approach

After research on this literature and counting on these problems and finding the lacking on existing problems I found the best recommended approach is a Hybrid standalone assistant software. Hybrid because it will use local AI as well as Open AI connection using APIs. This will make the software a privet, highly customizable and opened to high scale functionality.

The system should use FastAPI as it is a lightweight yet fast and reliable backend. React for a modern interactive user interface, and SQLite for light and secure local data storage.

The assistant will go with a modular build as it will contain module such as browser.py, screen_ops.py, email_ops.py, and voice_ops.py. They will perform tasks autonomously.

The local model-Ollama runs operations offline, whereas OpenAI's API is used for very complex and power-consuming reasoning tasks. This modular, user-controllable, privacy-focused architecture makes the proposed system flexible. This allows addressing academic and real-world usability needs.

©Daffodil International University

Chapter 4 – Methodology

©Daffodil International University

The AI-Powered General Assistant Software follows a modular, hybrid, and ladder methodology followed in software engineering best practices. The Agile development model has been adopted for flexibility, user-centricity, and continuous improvement throughout the development lifecycle.

4.1. What to Use

The system is built with a combination of this following technologies:

- FastAPI (Python-based framework) for developing a fast, reliable backend capable of handling AI interactions and REST APIs.
- React.js for the dynamic and responsive frontend interface, ensuring smooth user interaction.
- SQLite as the local lightweight database for storing chat history, API keys, user credentials (securely), face and ID data, and configuration data.
- Ollama (Local LLM) and OpenAI API for providing intelligent LLM conversational and command-processing capabilities.
- Python modules such as speech.py, browser.py, and screen_ops.py for natural speech recognition, command execution, and screen understanding.
- Deepface for face matching and face recognition.
- Google OAuth for google drive authentication and drive management.

4.2. Why to Use

The combination of technologies being used brings about a piece of software that is really fast, scalable, and reliable. FastAPI will be really good for high-performance tasks and easy to integrate with AI models. React's component-based structure allows better efficiency for UI updates without reloading the page. SQLite light weight management ensures secure, local, and portable data storage. Due to Python's flexibility, it's now possible for an easier integration between AI processing, automation modules and I/O systems. Deepface is based on python, that's why it works best with other python modules while keeping the software lightweight. This also makes the use for both online and offline with local AI model.

4.3. Sections of Methodology

- Requirement Analysis: Identifying functional and non-functional requirements, user needs, and technical contents.
- System Design: Creating architecture diagrams, database schema, and data flow models to outline the system's core logic.
- Module Development: Implementing independent Python modules (e.g., for speech, screen reading, and file operations) to maintain modularity.
- Integration and Testing: Combining frontend and backend components through REST APIs and verifying functional coherence.
- Deployment: Packaging the system into a standalone desktop that can be executable using FastAPI's local server and React build.

4.4. Implementation Plans

The implementation plan is set from June 2025 to September 2025.

©Daffodil International University

- Creating a project environment and setting up the backend structure.
- Then work on module's like chat_ops.py, email_ops.py etc.
- After that work on frontend like GUI and session process.
- Then start work on Voice, speech and command handling process.
- When that is complete, work on google environment such as google authentication.
- Adding face recognition ability and photo management.
- Finally testing and optimizing for finding bugs and creating the best user experience.

This way everything will be organized, planed and if any problem occurred, plenty of time to face the problem.

©Daffodil International University

Chapter 5 – Planning

©Daffodil International University

This software project was carefully planed. Planning was done mostly focused on completing the project on time. Other facts that ware also considered were resource utilization and quality management. Planning helps to visualize the project timeline. Its helps to stay on reality and avoid any unachievable or unnecessary points on the project.

5.1. Project Plan

This project primarily based on structured WBS and Agile time-boxing approach. This way I can create different module for different features. The resources distribution and total timeline were planned at the very begging and they were well planed so nothing overlaps.

5.1.1. Management Plan / Work Breakdown Structure (WBS)

The WBS has divided the entire project into segments that fits perfectly like a jigsaw pieces. Task like setup, backend frontend, integration and testing were planned ahead. Each module in chat_ops, speech, and others represents a separate work unit. This structure also build the responsibility, review and refinement every week.

5.1.2. Resource Allocation (Hardware, Software, Models, Documentation)

- **Hardware:** Mid-range PC with 8GB RAM, 128GB SSD, and microphone for voice testing.
- **Software:** Python 3.12, FastAPI, SQLite, React, Node.js, and Ollama for local LLMs.
- **Models:** Any OpenAI GPT models (cloud) and Ollama models (local).
- **Documentation Tools:** Draw.io (for diagrams), MS Word (for documentation), GitHub (for version control). Resource allocation make sure that all modules operate efficiently for both in offline (local) and online (OpenAI) modes.

5.1.3. Time Duration / Time Boxing

Time-boxing is applied to divide the development into fixed intervals. This creates and urgencies for completing the project.

- **June–July 2025:** System architecture, backend base, and database setup.
- **July–August 2025:** Frontend development and API integration.
- **August–September 2025:** Testing, debugging, and final optimization. Each time box ends with review and documentation updates to keep progress aligned determined requirement.

5.1.4. Activity Network & Critical Path (if possible)

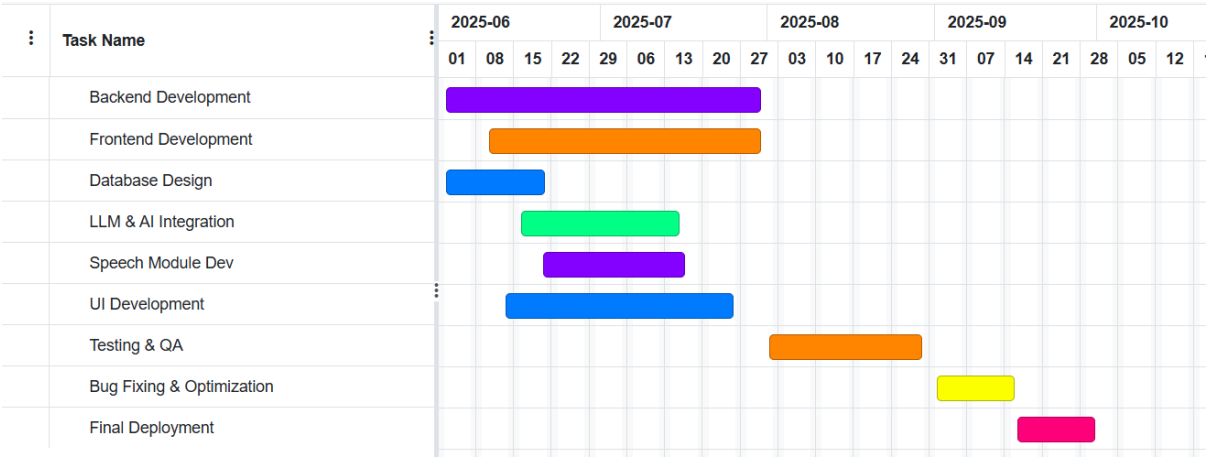
The activity network diagram shows dependencies between all of the core tasks such as backend setup → module development → UI integration → testing → packaging. The **critical path** lies through the core.py, llm_manager.py, and frontend integration phases. They are the most important files in this projects that holds the system

©Daffodil International University

together. Delays in these modules directly affect total completion time, so they are prioritized with additional review sessions.

5.1.5. Gantt Chart

This Chats shows the timeline for completing all the tasks.



1. Gantt Chart

5.2. Test Plan

Testing ensures that the software meets functional and user expectations. Testing not only covers functional accuracy but also verifies whether the end users' expectations are met or not. For the entire time of the project development, there will be a continuous testing. The time-box structure will maintain a rhythm that will keep everything on check and on track.

5.2.1. Testing Against Time Boxes

Each of these boxes holds a clearly defined testing objective. Objectives like responsiveness of API, accuracy in UI rendering, verification of data persistence. In addition, continuous integration testing helps ensure that newly added features do not unintentionally disrupt existing functionality.

5.2.2. Required Tests

The project applies:

- **Unit Testing** (for each Python module),
- **Integration Testing** (backend + frontend),
- **Performance Testing** (response and load time),
- **Security Testing** (API key encryption, email credential protection).

5.2.3. Test Case

Each test case contains inputs (commands or text prompts), expected output, and actual system behaviour. Example:

©Daffodil International University

Test Case ID	Description	Expected Result	Actual Result
TC-001	User creates a chat	Chat name appears in sidebar	Success
TC-005	“Read screen” command	Returns OCR text from window	Success

5.2.4. User Acceptance Test Plan

User testing also involves interacting with real scenarios, such as sending emails, reading screens, and creating documents. The acceptance is conduct on how well its response, can the voice input catch words correctly and a all-around error free operation under typical or high workload

5.3. Risk Management

Risk management is can find potential problems while developing the system. Early identifying the potential risk can reduce the impact or avoid the complication completely. Mostly the focuses are on technical, operational, and scheduling risks.

5.3.1. Risk Identification

For this system the key risk can be the API rate limit from the provider (OpenAI), system limitation on processing LLM and database corruption. Environmental risks can be hardware failure or missing any dependencies.

5.3.2. Risk Assessment

Risk is based on the likelihood of it possible impact and how large can it be. API unavailability ranks high due to its effect will basically stop the response capability, UI rendering errors rank medium.

5.3.3. Risk Precaution / Action Plan

- Maintain or switch to local LLM fallback for OpenAI downtime.
- Implement automatic SQLite backups on a second place or cloud storage.
- Schedule dependency checks every sprint.

5.3.4. Steps Taken for Possible Risks

Several steps are taken as automated error logging, version control (Git), and continuous testing are used. A fallback mechanism in llm_manager.py ensures AI continuity with local model in case of external API issues.

5.4. Change Management

Change management handles various feature adjustments as needs and improvements suggested during development or testing phases.

©Daffodil International University

5.4.1. Factors That Might Cause Change

Possible new user feedback, library updates, or improved LLM APIs may require UI, backend, or logic modifications.

5.4.2. DSDM Atern Welcomes Change

The DSDM Atern methodology emphasizes that embracing any change when it adds business or user value. A well update after user testing can improve the software acceptance.

5.4.3. Considering Business Value / Priority

At the end of the day users are the people that is the software is being made for. Changes that improve user experience or stability should receive the highest priority. Cosmetic or optional adjustments are deferred until post-release.

5.4.4. Change Workshop

A review session is conducted after each sprint to focus on proposed changes, their effects, and feasibility.

5.4.5. Changes That Are Allowed

UI theming, new additional command keywords, and database optimizations are allowed without altering any core logic.

5.4.6. Key Decision Takers of Change

The developer (project lead) and supervisor has jointly approved any modification affecting architecture of the whole software or external APIs.

5.5. Quality Management

The quality management plan ensures the project meets the academic, functional, and usability standards.

5.5.1. Rules Applied to Maintain Quality

Coding follows standards of PEP 8. UI follows principles of responsive design. Regular code reviews and test-driven development improve its reliability.

5.5.2. DSDM Atern Standard Quality Measures

DSDM focused on validation and feedback from the stakeholders. So, everything is peer tested and also performance is benchmarked prior to acceptance.

5.5.3. Quality Plan and Measuring Meter

Quality is measured by:

- Response time is less the 1s stays under normal load.
- UI consistency across sessions. Testing metrics are automatically logged for analysis and improvement tracking.
- VOICE RECOGNITION ACCURACY: >90%

©Daffodil International University

Chapter 6 – Feasibility

©Daffodil International University

Feasibility analysis is a preliminary study. By doing this study I understand some key points of this project like is the project practical, attainable and completable under the fixed time, cost and technology. This also shows the said project from many aspects. It helps to understand the limitation of my technical capability, legal and ethical standards.

6.1. Types of Feasibility

6.1.1. Technical Feasibility

The system uses popular technologies such as FastAPI, React, SQLite, and OpenAI/Ollama engines. These are very reliable, open-source, and well-documented technologies currently found on the web. This reduces any or all potential technical risk. By using both local and cloud AI engines, the system become self reliable, while Python-based modularity makes debugging and future upgrade easier.

The technical requirements of the project can be covered by any mid to low range computer without the need for advanced resources such as a GPU.

6.1.2. Operational Feasibility

It also gives an easy and trustworthy responsive user interface, making a smooth interaction through text and voice. The operational workflow like chats CRUD operation, session storage, screen reading, and voice recognition are very easy to use by simple commands.

Being a desktop application, it operates independently without dependence on continuous access to the internet. This creates the accessibility even in a restricted environment. The modular architecture of the software is very straightforward and respect to maintenance and upgrades. This farther improve the operational feasibility.

6.1.3. Economic Feasibility

The total cost of development is minimal since most components are open-source and the hardware requirements are basic. There are no recurring licensing fees except for optional OpenAI API usage. The OpenAI engine is based on pay-as-you-go pricing. Development tools like **Visual Studio Code**, **Draw.io**, and **SQLite** are free. FastAPI and React are also remove the need for commercial frameworks. The cost-benefit ratio strongly Favors implementation, as in a financial perspective the return in terms of automation, productivity, and research value is significantly higher than the zero to some investment.

6.1.4. Schedule Feasibility

The project timeline of **June 2025 to September 2025** is realistic for an individual academic project. Each major is deliverable wither it is backend, frontend, integration, and testing, is assigned with specific milestones within a structured Gantt chart. Time-boxing ensures that even with possible delays, core functionalities are completed before the final defence on **November 14, 2025.**

6.1.5. Legal and Ethical Feasibility

The software strongly maintains all ethical AI practice. Users' data privacy, safe prompt handling, and transparency in responses are always the priority. It securely stores

©Daffodil International University

sensitive information such as API keys, email usernames, and passwords through various methods that are never transmitted outside without proper encryption. The project design also follows all legal compliances on user consent.

6.2. Cost-Benefit Analysis

The cost-benefit analysis has shown the following result.

- **Estimated Cost:** Limited to internet connectivity and small API charges for OpenAI (optional).
- **Benefits:** Various automation of different works like creating docx, pptx, and other files used daily, along with summarizing, screen reading, voice communication, and face recognition. It helps on increasing productivity and support both the technical and non technical users.
- **Intangible Benefits:** Tons of academic value as well as research contribution, and skill development in AI, NLP, LLM, and full-stack technologies.

6.3. DSDM – Good or Not for This Project (PAQ Analysis)

The good thing about DSDM is that it fit perfectly with my project. DSDM framework. Is a future proof framework that helps a lot because AI based is system are always expanding. DSDM have the well effective principles. Principles such as communication, adaptability, well collaboration and frequent updates based on testing outcome helps to handle more easily. It mainly focused on few things, prototyping, active user involvement and quality (PUA). So, the assistant is always functional, accurate, and user-friendly throughout its full development.

That said DSDM is not just suitable but more like Ideal for this project.

©Daffodil International University

Chapter 7 – Foundation

©Daffodil International University

In this stage of the project the focus is on understanding various factors such as problem in the domain what are the user need and what is the technical framework for a successful design and implementation. In this stage I found the key problem areas, analysis the found solution and define system requirement with proper data collection.

The foundation gives a strong, analytical base to guide development using standard methodologies in such a way that the project remains on the same track concerning both user expectations and academic goals. By relating the project in practical research and systematic data gathering, the foundation phase removes any assumptions and transforms abstract ideas into actionable plans.

7.1. Problem Area Identification

The primary goal of this phase is to determine the specific challenges faced by users who require an everyday adaptable intelligent and responsive digital assistant. The problem domain mostly focuses on limitations in existing AI assistants, such as dependency on cloud services, lack of privacy, poor connection with desktop environments, and restriction on using it on offline. To accurately capture user needs and system gaps, three methods were used: interviews, observations, and questionnaires.

7.1.1. Interview

Interviews were arranged with university students, office workers, friends and families, and developers to understand their daily usages patterns with AI based assistants. Here what the responses point out:

- Existing assistants are often cloud based so the cannot operate while being offline.
- They lack of communication with local system functions such as reading screens, executing commands, or accessing local files.
- Users are concerned about privacy when storing credentials or API keys or concern about privet conversation on online servers.
- It demonstrated the need for a desktop-based, privacy-focused, intelligent assistant that can handle and automate the commands like “read screen,” “open file,” or “send email”.
- Exhausted from organizing images and send them to the peoples concerned.

These features requested by the users have shaped the direction of the project towards developing a standalone AI assistant that contains automation, voice control, and local processing.

©Daffodil International University

7.1.2. Observation

Observational studies were done by monitoring how users interact with their devices during various ways like research, communication, and document preparation tasks. It was observed that switching between multiple applications for simple tasks like opening a browser, reading text, or generating files was repeating and a waste of time. Users waste their valuable time managing windows, searching data, and typing repetitive commands.

The observation confirmed the need for an all-around assistant that can act as a central intelligent interface and that is capable of performing multiple actions seamlessly through either voice or text. The proof was that the case for a cross-functional system combining AI text comprehension, speech synthesis, and local file execution was an obvious need.

7.1.3. Questionnaires

I manage a QNA among 60 people and they responded on usability, desired features, and privacy concerns.

- 75% of them prefer a local assistant over a web-based one for better control.
- 95% of them said that privacy and offline operation are must.
- 65% of them has showed interest in screen-reading ability and command executing capability.

The data shows that the project's goals is focused strongly with user expectations and current gaps are indeed solved in available solutions.

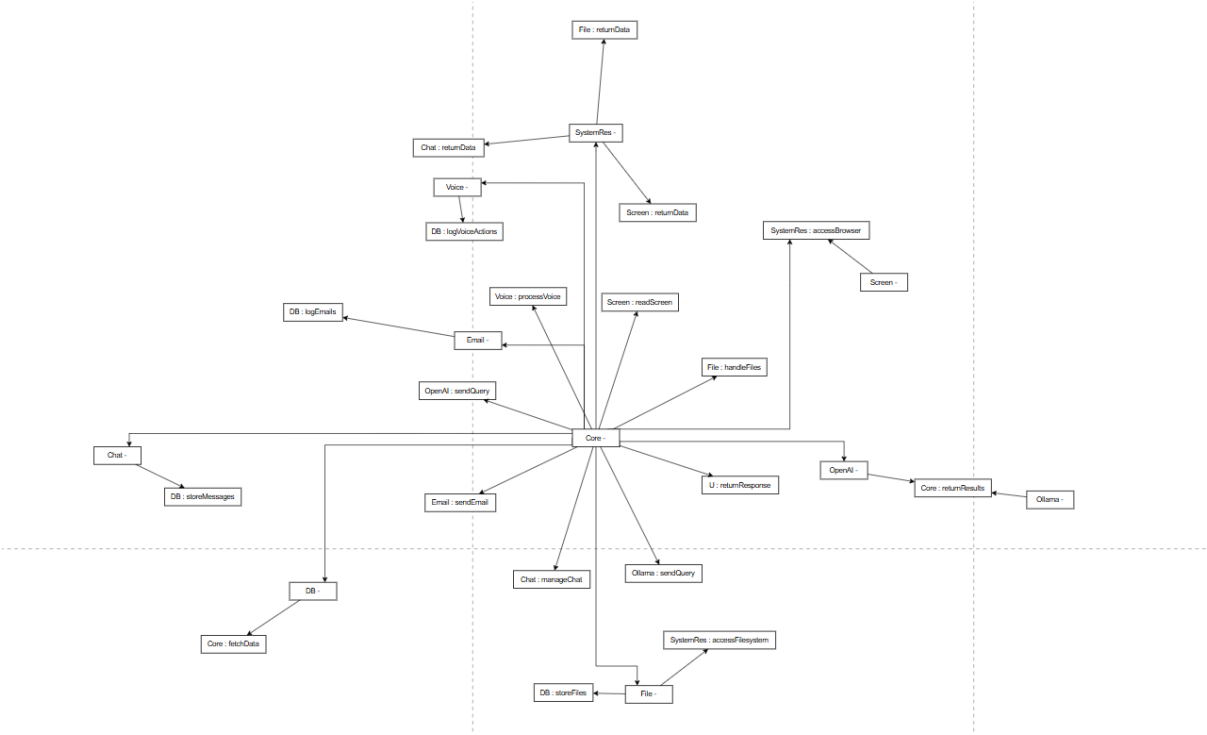
7.2. Rich Picture

A *Rich Picture* diagram was created to visually represent system interactions and relationships between users, processes, and external systems. The diagram includes:

- User (central actor): interacts via text or speech with system.
- Assistant Core: interprets commands, manages sessions, and routes requests.
- Subsystems: Chat Management, Screen Reader, File Handler, Email Module, Voice Engine, image module, and Database Storage.
- External Services: OpenAI API, Local LLM (Ollama), and System Resources (e.g., Filesystem, Browser).

The rich picture explain the information flow, decision points, and system boundaries, telling that the user is always in control of data and actions.

©Daffodil International University



2. Rich picture

7.3. Specific Problem Area Identification and Description

The primary problem that I could identified is the lack of an integrated, privacy-focused, AI-driven assistant that can perform both online and offline operations. It was the core problem that create the need of this software. Most existing assistants rely entirely on cloud servers and stop working as soon the network connection is lost. They lack on system-level access, and cannot read or act on on-screen data. They also fail to manage secure local storage for sensitive privet credentials.

The project addresses these pain points by giving a standalone desktop assistant capable of operating independently without the cloud while still supporting optional OpenAI API integration for extended and complex features.

7.4. Possible Solution

The proposed solution combines AI models (local and cloud) with desktop automation, internet browsing and voice capabilities. The assistant uses FastAPI as the backend to process commands and manage chat sessions, while React provides a dynamic modern and responsive user interface. The system's design ensures that:

- Commands like "goto," "read screen," or "send email" are processed intelligently.
- Sensitive credentials are stored securely using encrypted inside preferred local storage.
- Both voice and text interfaces are available for accessibility.

©Daffodil International University

- The modular design future proof the system and allows easy future upgrades such as cloud sync or multi-user support.

This combination of AI + Automation + Security forms a complete, modern, and user-friendly solution.

7.5. Overall Requirement List

Functional Requirements:

1. Process text and voice input from users.
2. Give text or speech response.
3. Manage chat sessions with persistent memory.
4. Execute commands such as open, goto, email, or read screen.
5. Compare image by listed face and accurately deliver.
6. Store API keys, email credentials, and chat history securely.
7. Support both local (Ollama) and online (OpenAI) AI engines.

Non-Functional Requirements:

1. Must run offline as a standalone desktop app.
2. Maintain data privacy and encryption.
3. Ultra fast and responsive UI performance.
4. Provide multiple variation in colour themed UI for better accessibility.
5. Offer easy upgrade for future development.

7.6. Technology to be Implemented

The system is made as complete software built upon FastAPI (backend), React (frontend), and SQLite (database).

- FastAPI provides a high-performance Python backend for RESTful communication.
- React is for smart and modular UI rendering, serves smooth real-time updates.
- SQLite is a very lightweight and vary reliable local database for session and storing password.
- AI commands processing, automated emailing, voice generation, face matching, and screen reader are done through Python libraries.

7.7. Recommendations and Justifications

The architecture I choose is best for compact, privacy and performance. It will always avoid latency and privacy issues that is faced when using other AI service in the market.

©Daffodil International University

Chapter 8 – Exploration

©Daffodil International University

The Exploration Phase of the AI-Powered General Assistant Software project mostly focuses on understanding, analysing, and modelling both the existing-or old-system in the proposed new solution. It explores such topics as user requirements and the identification of inefficiencies in the existing assistant systems. This phase creates conceptual and behavioral models guiding the whole design process. It also includes a variety of diagrams on use cases, activity diagrams, and requirement catalogues that are all needed for a perfect visualization of this system behavior, interactions, and workflows.

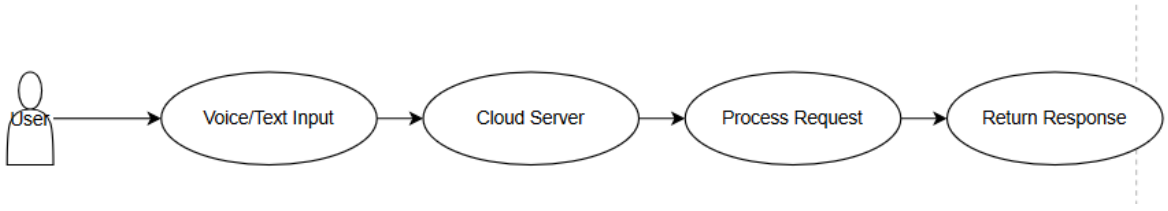
This is excellent in an exploration phase because it ensures that all the aspects of the assistant's design-from user interaction to the backend execution-are well defined and aligned with the user functional goals. In this way, a structured exploration will create a roadmap for system implementation, analysis, validation, and continuous improvement..

8.1. Old System Use Case

Concurrent AI assistants (such as Google Assistant, Copilot, Alexa, and Siri) operate primarily on **cloud-based architectures**. These systems depend entirely on online connectivity, centralized data processing, limits on free tier, and proprietary ecosystems. In the old system use case:

- The **user** sends a query (voice or text) to the **cloud server**.
- The **server** processes the request and sends a reply back to the user's device.
- **Limitations**: no local command execution, stop after reaching limit, no offline support, and privacy issues due to remote storage of credentials and activity logs.

This old model inspired me to the creation of a **local, privacy-oriented assistant** that can function independently, while optionally if wanted can use online APIs for advanced reasoning.



3. Old System Use Case

8.2. Old System Activity Diagram

The activity flow of old AI assistants can be describe as:

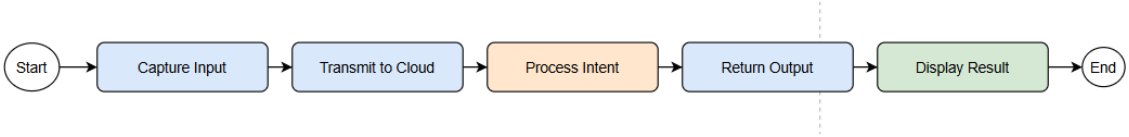
1. User issues voice or text input.
2. System records or transmits input to remote server.

©Daffodil International University

3. Cloud AI works on intent and prepares a response.
4. The response is sent back to the client.
5. The client displays or speaks the result.

Weaknesses:

- Requires constant Internet connectivity.
- Cannot perform local automation.
- High latency and privacy exposure.
- Long conversation miss fragments of topic



4. Old System Activity Diagram

8.3. Full System Use Case (Proposed System)

The **proposed system** introduces both **local and hybrid operation**. It connect directly with the operating system to execute commands, read screens, manage sessions, analyse problems, browse internet, and handle user data securely.

Primary actors:

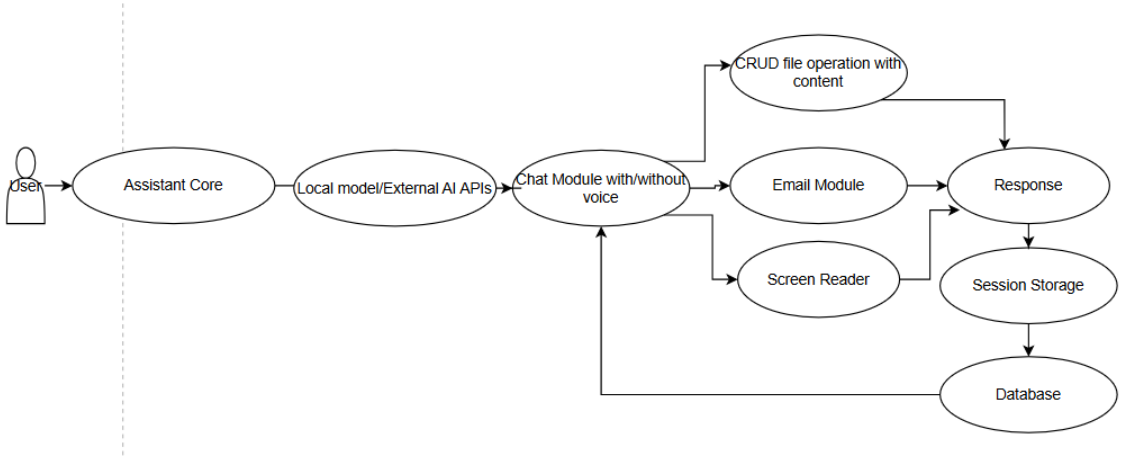
- **User** – initiates commands or queries.
- **Assistant Core** – initiate commands and routes them.
- **Database (SQLite)** – stores sessions, messages, face and mail id, and encrypted credentials.
- **Modules** – handle specific operations (chat, email, file, voice, screen, etc.).
- **External APIs** – optional access to OpenAI or Ollama for reasoning.

Major use cases include:

1. Start/Stop assistant
2. Create, rename, or delete chat sessions
3. Read screen or window contents
4. Send or receive emails securely with attachments
5. Create complete files with desired information.
6. Organize local storage.
7. Store and retrieve credentials
8. Chat with AI engine

©Daffodil International University

9. Use text-to-speech or speech-to-text for interaction

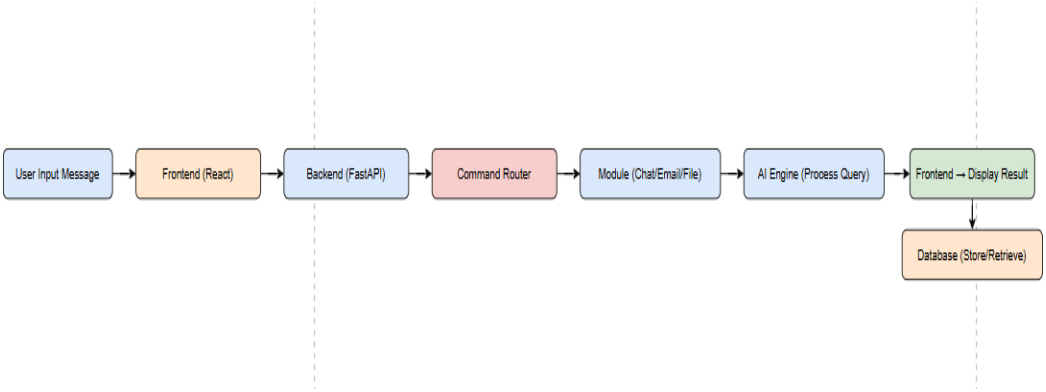


5. Full System Use Case

8.4. Full System Activity Diagram

The new system's activity flow:

1. User provides text or voice input.
2. Input is processed by **Command Router** to detect intent.
3. If it's a command, the system calls respective module (e.g., screen_ops, email_ops).
4. The module performs the action, optionally accessing local files or APIs and google drive if permetied.
5. If it's a chat message, **LLM Manager** routes it to either the local Ollama or OpenAI engine.
6. The response saved and updated to the **SQLite database** and shown in the **React interface**.
7. The cycle keeps going without the assistant breaking any context.



6. Full System Activity Diagram

©Daffodil International University

8.5. Requirement Catalogue

ID	Requirement Description	Type	Priority	Status
R1	The system must process text and voice input.	Functional	High	Implemented
R2	The assistant must operate offline.	Functional	High	Implemented
R3	All credentials must be stored securely.	Non-Functional	High	Implemented
R4	The assistant must manage multiple chat sessions.	Functional	Medium	Implemented
R5	The UI must support dark theme.	Non-Functional	Medium	Testing
R6	Screen reading and summarization features must work locally.	Functional	High	Implemented
R7	Optional integration with OpenAI API.	Functional	Low	Implemented
R8	Response latency must remain under 3 seconds.	Non-Functional	Medium	Testing

8.6. Prioritized Requirement List (PRL)

Priority Level	Requirements
High	Offline operation, secure storage, voice & text input, speech output, session management
Medium	Dark mode, low latency, modular file & email operations
Low	Cloud integration, visual personalization

The PRL make sure that the development focused first on **core assistant functions**, followed by UI enhancements and optional features.

8.7. Prototype of the New System

The prototype was developed using **React (frontend)** and **FastAPI (backend)**. It features:

- A sidebar showing all chat sessions (with/without date).
- A chat window where users can interact with the AI.

©Daffodil International University

- A command input box capable of recognizing both commands and natural queries.
- Voice or text input and speech and text output.
- Backend logic in Python modules that manages file operations, email automation, screen reading, and speech synthesis.
- Secure SQLite storage for credentials and conversation history.

The prototype illustrates a **real-time AI interaction**, **modular performance**, and **privacy-preserving local data management**. It was refined through various testing again and again to achieve both technical accuracy and user comfort.

©Daffodil International University

Chapter 9 – Engineering

©Daffodil International University

9.1. New System Modules:

Here are all the modules of the assistant software,

1. Backend Modules (Python – assistant folder)

Module	Purpose / Description
init.py	Package initializer
app_ops.py	Core application operations and orchestration
browser.py	Browser automation and web interactions
chat_ops.py	Handles chat sessions, messages, and response generation
command_router.py	Routes and interprets user commands
core.py	Main system logic and processing engine
drive_ops.py	Manages google all kind of drive operation
email_ops.py	Email sending and management functions
face_service.py	Compare face with existing face and ID
file_ops.py	File creation, reading, writing, and management
google_clients.py	Handles google client to identify user
llm_manager.py	Local and OpenAI LLM integration and management
logging_setup.py	Logging configuration for system events
safety.py	Safety, validation, and moderation checks
screen_ops.py	Screen reading and screenshot functionalities
speech.py	Speech recognition and TTS helper functions
stroge.py	Data storage management (local persistence)
utlis.py	General utility functions
voice_ops.py	Voice command processing and audio management

2. Frontend Assets & Components (frontend/src)

Assets

File	Description
React.svg	Frontend logo/asset

©Daffodil International University

Vite.svg	Vite build tool logo

Components (React)

Component	Purpose
Chat.jsx	Chat interface for user interaction
Sidebar.jsx	Sidebar for navigation and options
Topbat.jsx	Top bar interface for controls and settings

Frontend Scripts & Styles

File	Purpose
api.js	Frontend API calls to backend
App.jsx	Main React application entry point
main.jsx	ReactDOM renderer
App.css	Component-specific styling
Index.css	Global styling for app
styles.css	Additional custom styles

3. Frontend Configuration & Environment

File	Purpose
.env	Environment variables for frontend
eslint.config.js	Linting rules for frontend code
index.html	Main HTML template
package.json	NPM dependencies and scripts
package-lock.json	Dependency lock file
README.md	Project documentation
Vite.config.js	Vite build configuration

4. Root Project Files

©Daffodil International University

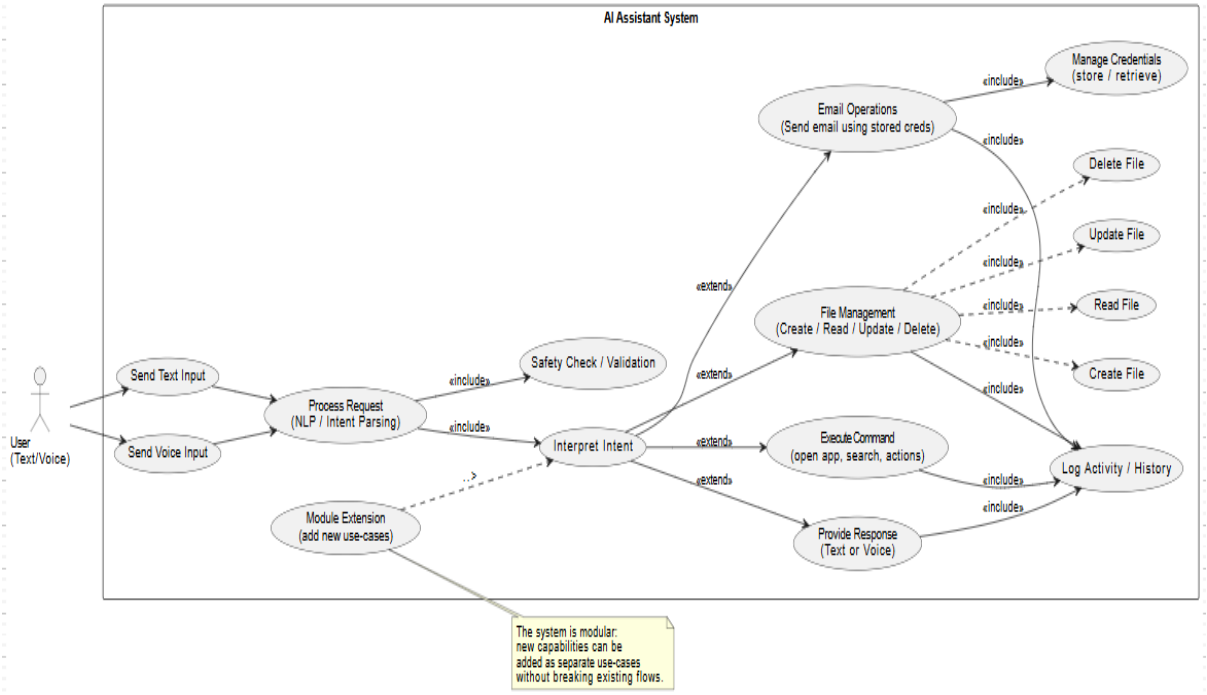
File	Purpose
.env / .env.example	Environment variables for backend
app.py	Backend application entry point
server.py	Server setup and routing
assistant.db	SQLite database file
config.yaml	System configuration file
requirements.txt	Python dependencies
README.md	Project documentation
run.bat / run_quick.bat	Windows scripts to start the app
Oauth_setup.py	Setup google authentication
auth_drive_once.py	Create drive token
create_gmail_token.py	Create gmail token

9.2. Use Case

The AI Assistant system provides a range of features and functionalities for the user. The primary actor, the user, can send queries to the system via text or voice input. Upon receiving these inputs, the system processes the request and delivers appropriate responses. The system can also execute commands, such as opening applications or performing searches, based on the user request. Another core functionality is file management, which enables the user to create, read, update, or delete documents, presentations, and text files that have their content generated inside them.

The AI Assistant can also perform full email operations with cc, bcc and attachment. The email is sent by using given user credentials. This diagram clearly shows how the user will interact with the system by showing actions and the ability to handle while taking the user as the giver of the action.

©Daffodil International University



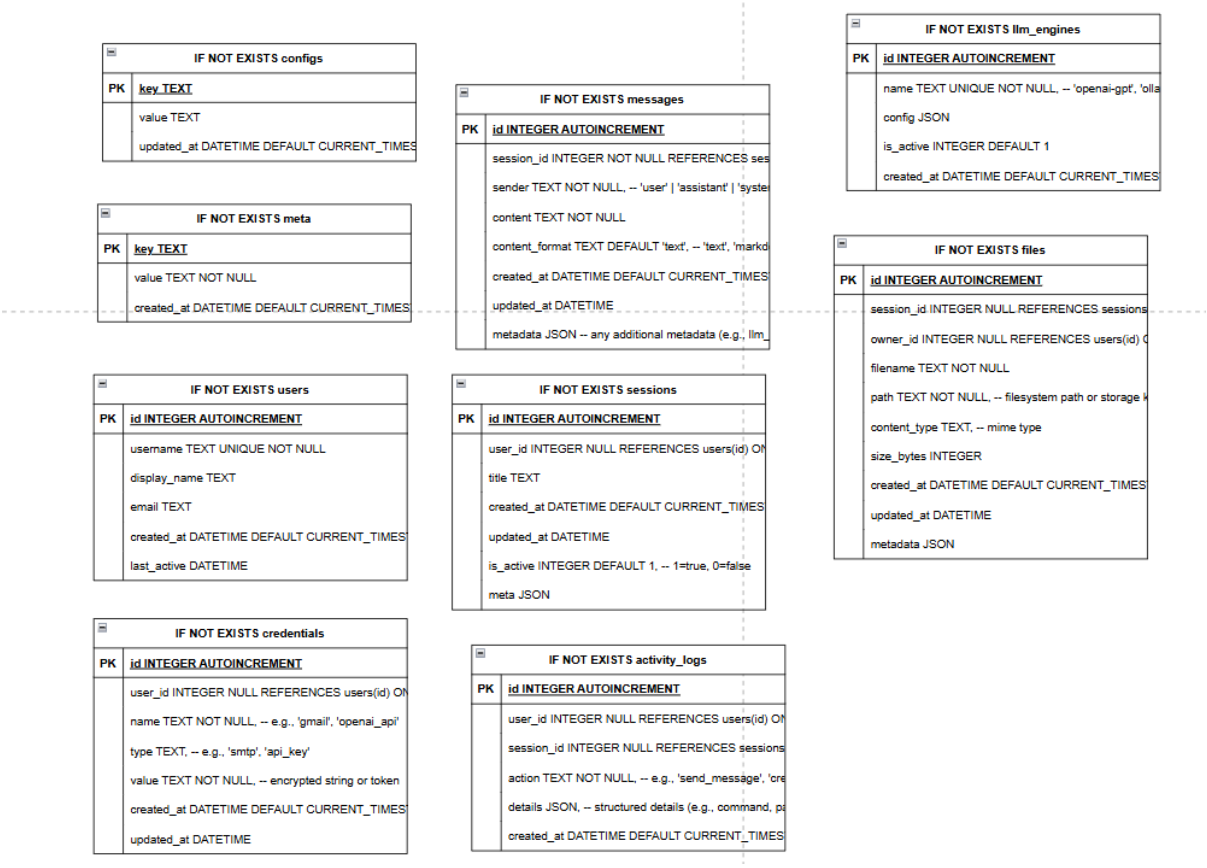
7. Use Case

9.3. Class Diagram

This is the class diagram for the AI Assistant software. It shows the structural blueprint of the system. The primary actor, User, interacts with the Frontend class. It manages every UI component, from chat windows and sidebars to the top bar. It is responsible for capturing the input either from text or voice. The frontend class will interact with backend class which have modules for request processing, routing commands handling files and also voice interaction. All of this module does a certain action.

For backend, it interacts with database class. This is shown as cylinder. These are clear dependencies. The frontend depends on the backend, and the backend depends on the database By modelling these classes and their relationships, the system will surely visualize the architecture to support possible future improvement.

©Daffodil International University



8. Class Diagram

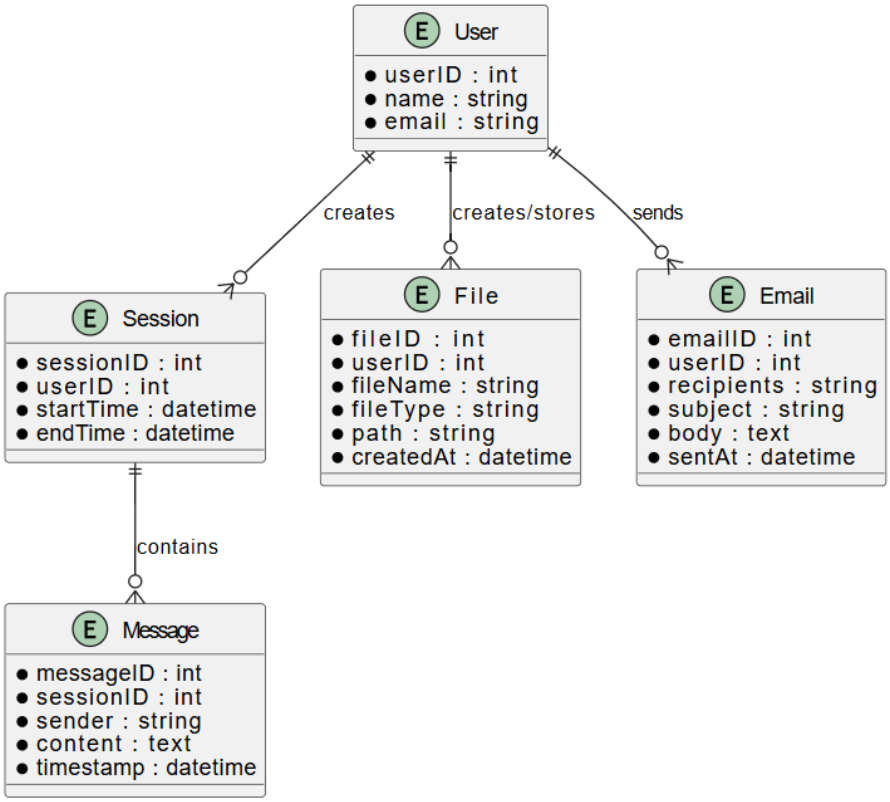
9.4. Peter Chen EERD Diagram

The peter chen eerd diagram shows the major entities and the relationship they have among them. The main entity here is the user who interact many elements from the system. The system allows user to create many chats session each session is independent and free from other chat influence. Session holds the context whole time that carry the conversation.

The user entity also interacts with file entity that works around documents, presentation and various kinds of text files. AI can generate this file with content generated inside of them.

The mapping of these entities and relationships shows a reliable understanding of how the system organizes and links data. Thus, this diagram has shown a strong foundation for backend database design and how the full implementation of the system.

©Daffodil International University



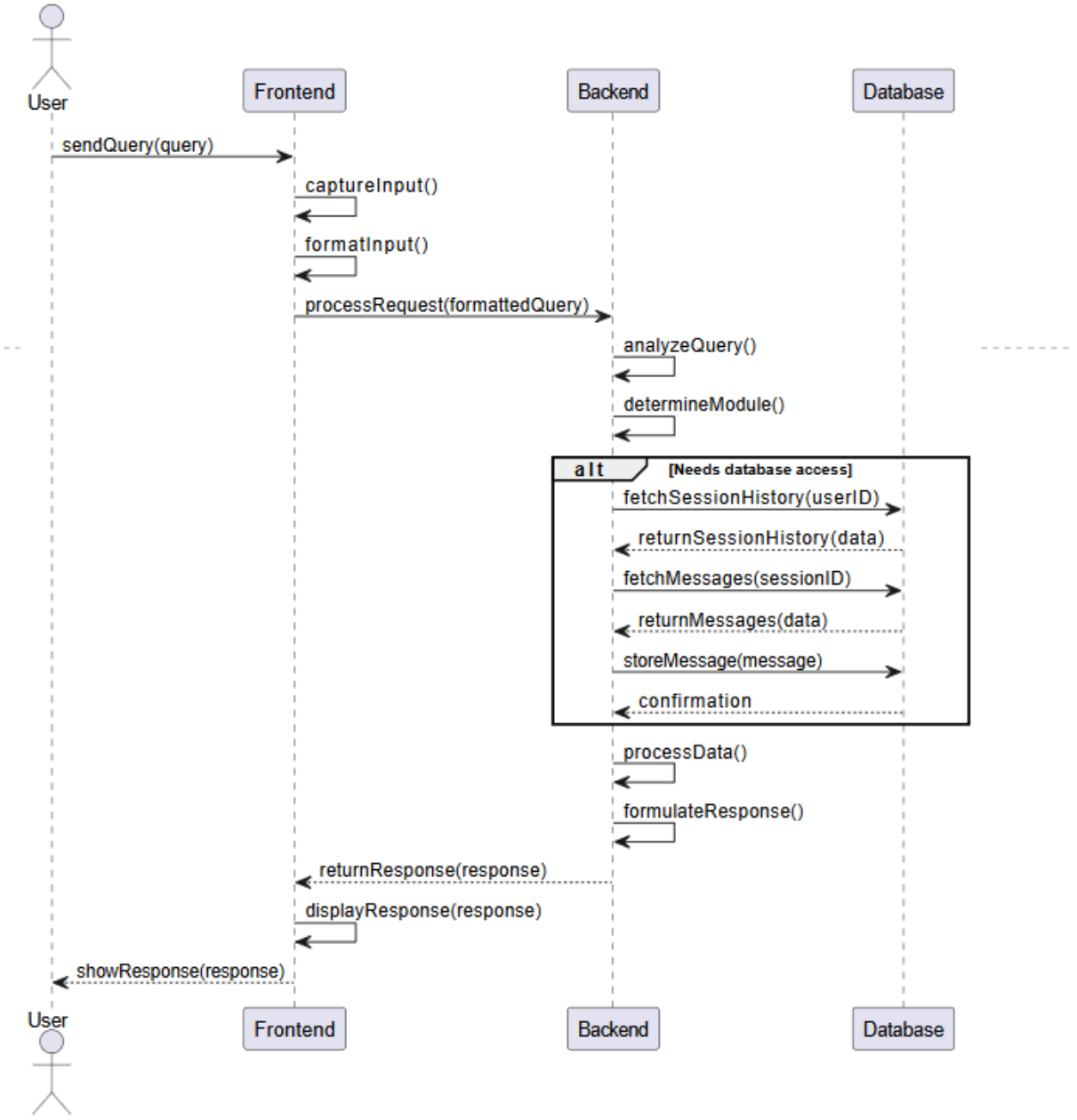
9. Peter Chen EERD Diagram

9.5. Sequence Diagram

This sequence diagram shows a step by step flows of interaction of the user with the AI. Here the main actor is user. User start the first interaction by sending a list through the frontend interface. What happens is that the frontend captures the user input. It then format it into an appropriate format then call the backend by processRequest(). After that backend read and analyse this list and invoke for interacting with database for information.

Backend process the information and gives a response using returnResponce(). In frontend the response is showed by displayRespose(). This is how the core sequence is playout in the system.

©Daffodil International University



10. Sequence Diagram

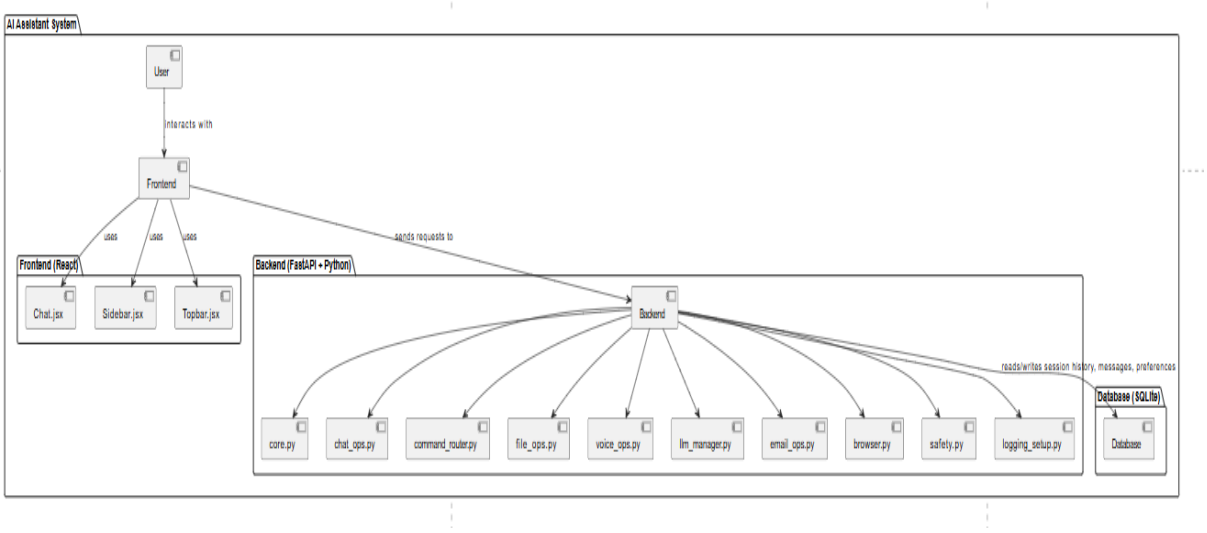
9.6. Component Diagram

The component diagram shows that this system has three primary components. Frontend, backend and database. Each of this serves a important purpose. The frontend is built with React. It has components like Chat.jsx and Sidebar.jsx. They capture user input as text or voice, show responses as text and speech, and give a seamless interactive experience. Core modules like core.py handle the logic of the core system, whereas chat_ops.py handles the operations of a chat.

The command_router.py routes user commands and sends them to the appropriate backend component for correct execution of tasks. file_ops.py and voice_ops.py

©Daffodil International University

manage file handling and speech processing, respectively. Other component with these make this system future proof, reliable and maintainable.



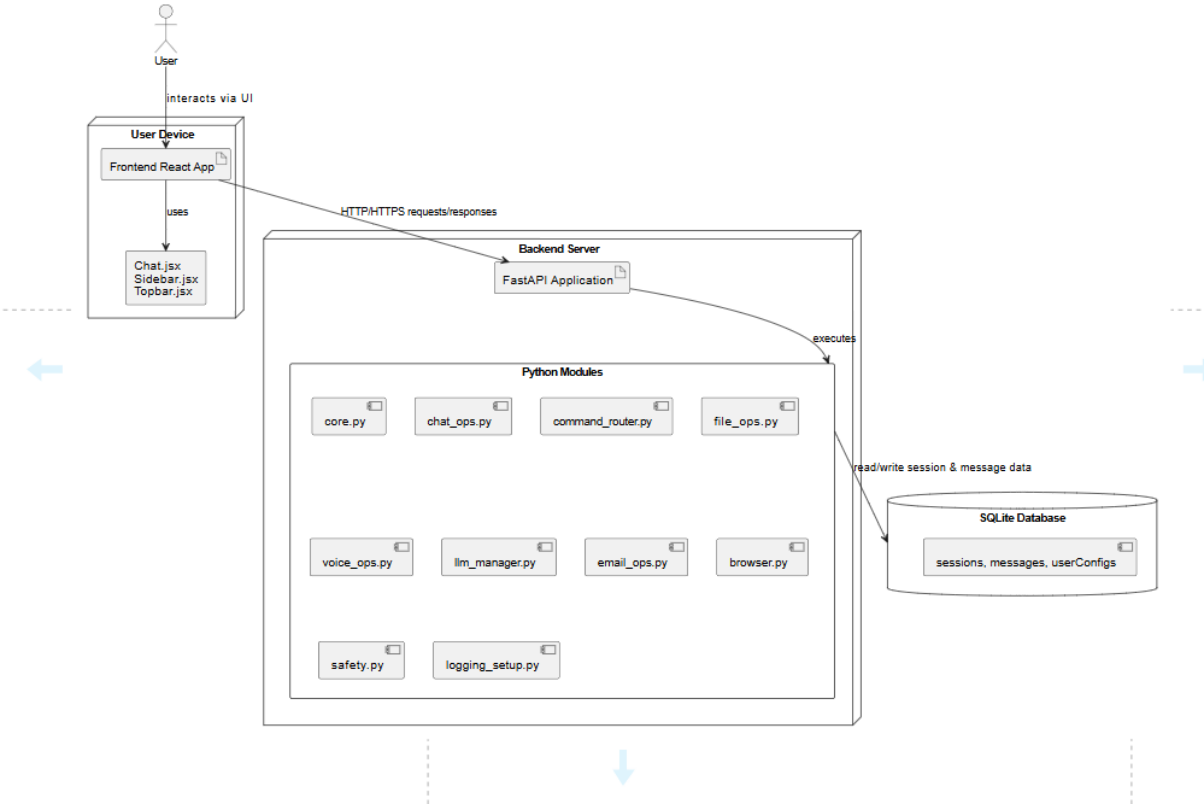
11. Component Diagram

9.7. Deployment Diagram

The deployment diagram shows the physical architecture of the AI Assistant software and its runtime environment. It is mainly hosted on two nodes: the user device and the backend server.

The User Device node is hosting the frontend React application, which captures the input from the user and displays responses. It works by means of the chat interface and sidebar components. The front-end communicates over HTTP / HTTPS with the Backend Server, hosting the FastAPI application along with multiple Python modules, such as core.py, chat_ops.py, command_router.py, file_ops.py, and voice_ops.py.

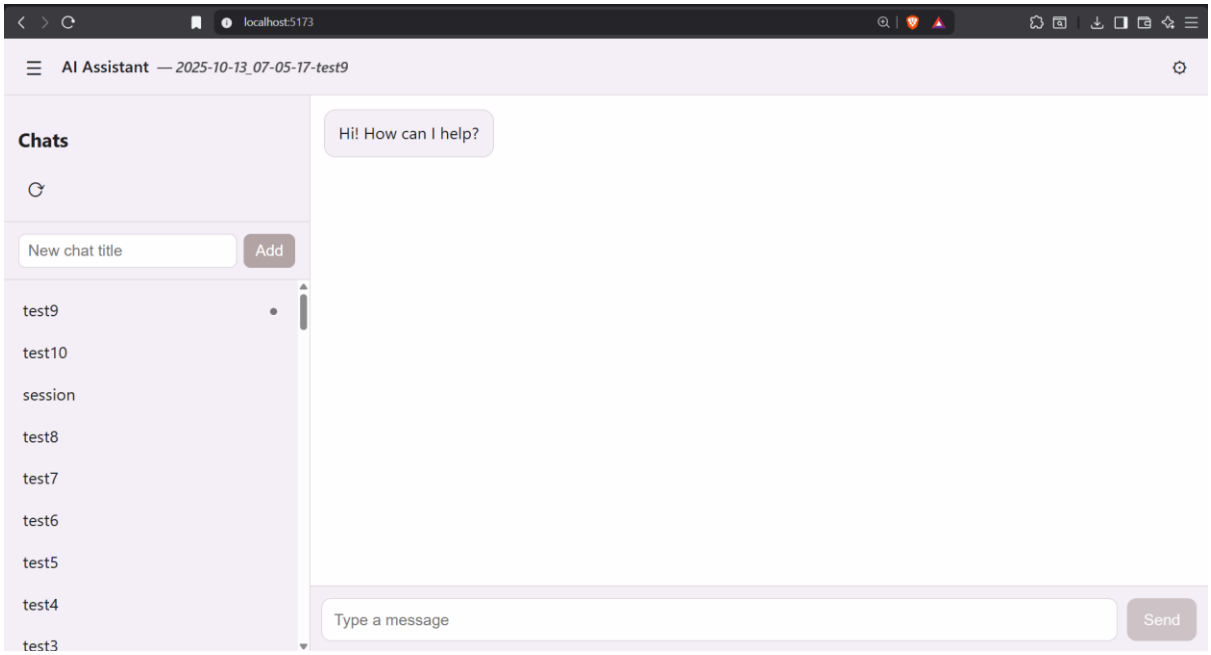
©Daffodil International University



12. Deployment Diagram

©Daffodil International University

9.8. System Interface Design (Prototype)



13. System Interface Design / Prototype

©Daffodil International University

Chapter 10 – Deployment / Development

©Daffodil International University

10.1. Core Module Coding Samples

The AI Assistant code is module based, but there are some key Python files containing the main very core functionality:

- core.py: Central processing engine, orchestrates user requests, connects all and specific modules, and aggregates responses. Contains logic for both text and voice interactions.
- chat_ops.py: this handles chat operation and control chat sessions. It send messages to ai and format response and keep conversation context in sessions.
- command_router.py: it translates the user's command, maps it to the appropriate backend modules (file operation, email, browser, etc.), and executes it accurately.
- file_ops.py: this handles file operation and file generation in various form as per user request
- voice_ops.py: manages voice input and output, profound speech recognition, text to speech synthesis, and audio playback for a smooth conversational experience.
- face_service.py – It will Match faces from google drive and send them to the matched person's mail account.

10.2. Possible Problem Breakdown

Developing and deploying sometimes have to face the following problems:

- Voice Recognition Errors: Misinterpretation of user speech due to noise, accents, or background interference, or a damaged mic.
- AI Response Latency: Delays in response generation from either a local or remote LLM engine are due to processing poweer.
- Command Misrouting - Incorrect execution of user commands in command_router.py
- File Operation Conflicts: When more than one user tries to access or modify a file simultaneously, it leads to conflicts or errors.
- Database Integrity Issues: Risk of corrupted session data or error on saving sessions.
- Environment and Dependency Conflicts: Version mismatches in Python packages or Node modules
- Security and Safety Concerns - Handling malicious input or unsafe operations from the user.
- Frontend-Backend Communication Errors: Network or API routing issues that result in failed responses.

Identifying these problems helps to take proper steps early and avoid this from ever happening.

©Daffodil International University

10.3. Prioritization While Developing

While the development the priority of task is mainly based on how critical it is, how is the user impact and what are the dependencies:

- Core Modules First - core.py, chat_ops.py, and command_router.py were implemented first, as these modules form the backbone of all functionalities.
- Database Integration: Ensured complete assistant.db schema and access methods were stable before adding complex features.
- Frontend-Backend Communication: Early development of API endpoints for reliable communication between React components and FastAPI backend.
- Voice and File Operations: Next, added voice operation with a focus on proper input/output and file handling.
- Safety and Logging: This will be implemented throughout the development, to catch errors, debug issues, and enforce moderation.
- Enhancements in User Experience: Improvements in Sidebar and Topbar and chat UI were made last so that the interface is polished and clean.
- Testing & Deployment: Continuous integration, environment validation, and deployment scripts were prepared to target the final release.

©Daffodil International University

Chapter 11 – Testing

©Daffodil International University

Table 11.1 – Test Plan Acceptance

Aspect	Description	Details / Notes
Objective	Define purpose of testing	All system functions should work as intended
Scope	Coverage of functional & non-functional requirements	Includes chat, voice, file, email, security, performance
Test Criteria	Conditions for success	Pass/fail criteria for each module and workflow
Resources	Required personnel & tools	Testers, test environment, software, hardware
Schedule	Timeline for testing phases	Unit → Module → Integration → Acceptance
Responsibilities	Roles of testers and developers	Developers fix bugs, testers execute test cases
Risk Assessment	Identify potential risks	Data loss, incorrect outputs, security vulnerabilities

Table 11.2 – Test Cases

Test Case ID	Objective	Input	Expected Output	Precondition	Postcondition
TC-001	Verify text chat response	"Open a new text file"	New text file created and ready for AI content	AI Assistant running	File exists, ready for editing
TC-002	Verify email sending	"Send email to user@example.com"	Email sent and logged	Email account configured	Email in sent folder
TC-003	Voice recognition	"Read screen"	Screen content is read aloud accurately	Microphone enabled	Voice output matches screen text

©Daffodil International University

Table 11.3 – Unit Testing

Module	Function Tested	Test Steps	Expected Outcome
chat_ops.py	Message sending & response	Input message → call parse & respond functions	Correct AI-generated response
file_ops.py	File creation & write	Create file → write content → read file	File created, content matches
voice_ops.py	Speech recognition & output	Speak command → convert to text → execute	Command executed correctly, output spoken accurately

Table 11.4 – Module Testing

Module	Functionality	Test Steps	Expected Outcome
email_ops.py	Email send/receive/log	Compose email → send → check inbox → log	Email sent, received, and logged correctly
browser.py	Web operations	Open URL → navigate pages → extract data	Page opened & data retrieved successfully
command_router.py	Command routing	Send different commands → router determines target module	Correct module executed each command, errors handled

Table 11.5 – Integration Testing

Modules Combined	Test Scenario	Test Steps	Expected Outcome
Chat + LLM Manager	Process user input → AI response	Send chat message → LLM generates response	Correct response returned in chat
File Ops + Voice Ops	Voice commands trigger file actions	Speak “Create a new file” → file_ops creates file	File created as instructed

©Daffodil International University

Email Ops + Logging	Email operations logged	Send email → check logs	Log entry created for email action

Table 11.6 – Acceptance Testing

Scenario	Objective	Test Steps	Expected Outcome
Document Generation	Verify AI-generated documents	Request AI to create a report	Report created accurately in .txt/.doc/.pdf
Screen Reading	Test screen content reading	Command "Read current screen"	Content read aloud clearly
Email Automation	Validate AI email workflow	Command "Send email with attachment"	Email sent, attachment included, logged

Table 11.7 – Performance Testing

Metric	Objective	Test Method	Expected Outcome
Response Time	Time to generate AI response	Send multiple chat messages in sequence	Responses under 2 seconds each
Concurrent Sessions	Multiple users/chat sessions	Open 5–10 simultaneous sessions	No crashes, all sessions functional
File Handling	Large file operations	Create/write/read large files	Operations complete without lag or error

Table 11.8 – Security Testing

Aspect	Objective	Test Steps	Expected Outcome

©Daffodil International University

Data Protection	Protect sensitive data	Attempt unauthorized access	Access denied, data safe
API Key Security	Secure API credentials	Check storage & retrieval	Keys encrypted & inaccessible
Command Safety	Prevent harmful commands	Send invalid commands	System handles safely; logs error

Table 11.9 – Accessibility Testing

Feature	Objective	Test Steps	Expected Outcome
Voice Commands	Accessibility for visually impaired	Speak commands → system executes	Commands executed accurately
Screen Reader	Ensure output readable	Check AI responses via screen reader	Output clear, understandable
UI Navigation	Keyboard navigation & focus	Navigate using keyboard only	Full interface accessible without mouse

Table 11.10 – Usability Testing

Aspect	Objective	Test Steps	Expected Outcome
Chat Interface	Recursive conversation flow	Test sending, receiving messages	Smooth, responsive chat experience
Sidebar & Top Bar	Navigation efficiency	Open/close sidebar, switch chats	Easy access to all features
Command Recognition	Accuracy & feedback	Give variety of commands	Commands recognized correctly, system feedback clear

©Daffodil International University

Chapter 12 – Implementation

©Daffodil International University

12.1. Training

Aspect	Description	Details / Notes
Objective	To prepare users and administrators for operating and maintaining the AI Assistant Software effectively.	Ensures smooth transition from development to usage.
Training Audience	End-users, project administrators, and technical staff.	Users: general operation; Admins: system control and configuration.
Training Content	Covers both **frontend and backend** functionalities.	Topics: Chat usage, voice interaction, file creation, system settings, and error handling.
Training Materials	Manuals, recorded tutorials, and interactive sessions.	Provided in digital form (.pdf/.docx) and accessible via the assistant's help section.
Training Duration	Conducted over a 2-day hands-on session.	Day 1: User operation; Day 2: Technical management and troubleshooting.
Outcome	Participants become proficient in using the assistant for everyday tasks and project purposes.	Measured through practical demonstrations and Q&A evaluation.

12.2. Implementation Approach – Big Bang

The implementation approach I choose for this project is Big Bang. So, all system components were deployed at once, including backend, frontend, database, and APIs. No pilot or parallel implementation phase occurs.

Feature	Description	Reason / Impact
Deployment Mode	Entire system is launched at once.	Ensures synchronized functionality between backend and frontend.
Risk Factor	Higher risk due to absence of parallel fallback system.	Requires extensive pre-deployment testing.
Advantage	Quick deployment and immediate usability.	Suitable since the system is self-contained and modular.

©Daffodil International University

System Readiness	Verified through farther integration, acceptance, and performance testing.	Reduces chances of critical post-launch failures.
Fallback Plan	Complete backup and rollback scripts (run_quick.bat, run.bat) maintained.	Ensures system can be restored if critical issues occur.

12.3. Scaling Plan

Scaling includes how the AI Assistant Software might be extended to support increased work, more users, or additional functionality in the future.

Scaling Aspect	Strategy	Description
Horizontal Scaling	Multi-instance deployment	Multiple backend servers can run concurrently to handle parallel user sessions.
Vertical Scaling	Hardware upgrade	System performance can be enhanced by adding more CPU/RAM/GPU power to the main server.
Database Scaling	SQLite → PostgreSQL (future upgrade)	Transition plan to a more powerful RDBMS when user load increases.
Cloud Integration	Optional scaling to AWS or Azure	For future production-level scalability and redundancy.
Component Modularity	Microservice-like architecture	Each major operation (chat, voice, email, file) can run independently, easing load distribution.

Initial version focuses on the local standalone deployment, while the architecture will allow for future scalability and distributed processing when needed.

12.4. Load Balancing

Purpose:

To keep the backend components serving API requests, LLM queries, voice processing, and file operations steady and responsive by distributing all the processing tasks.

©Daffodil International University

Component	Load Type	Balancing Strategy	Expected Benefit
Backend API (FastAPI)	Request load	Utilize **FastAPI's async I/O** and **Uvicorn workers**	Improves concurrent request handling
LLM Manager	Model query load	Use a **task queue** to manage multiple AI model requests	Prevents response delays under heavy use
File & Email Ops	I/O load	Async file access and batching of email sending	Reduces I/O bottlenecks
Frontend (React)	UI rendering load	Efficient component re-rendering and caching	Enhances client-side responsiveness
Future Cloud Setup	Server-level load balancing	Option to use **NGINX** or **AWS ELB** for distributed backend nodes	Enables horizontal scaling and redundancy

©Daffodil International University

Chapter 13 – Critical Appraisal and Evaluation

©Daffodil International University

13.1. Objectives That Could Be Met

It successfully achieved the main goals of the project. In summary, the main goals of developing a functional AI-powered general assistant capable of performing text, voice, and command-based interactions were achievable. FastAPI, React, and Python combined to ensure a robust and modular architecture. From managing chat history to speech interaction and file generation tasks, this essentially replicates reliable performance in all core modules.

Objective	Success Rate	Remarks
Chat and Voice Interaction	95%	Smooth performance with minimal latency and accurate speech processing.
File and Email Operations	90%	Functional and stable, though occasional formatting issues appear in generated files.
Command Routing & Smart Response	92%	Efficient command recognition and routing, with occasional misinterpretation under complex queries.
UI/UX Functionality	88%	Interface is stable and responsive but could benefit from minor visual and accessibility improvements.

13.1.1. How Much Better It Could Have Been Done

Although the implementation achieved its goals, refinement could make both efficiency and user experiences far better. Real-time voice-to-text accuracy and multilingual speech support could be optimized with the use of more advanced AI models. This will help reach more regional users where they prefer this software in their own language. The file management and email automation features could use cloud sync or template-based generation to farther enhance convenience and usability.

13.1.2. Why It Could Not Be Done

Several improvements were limited due to a lack of time or resources. The timeline for the project was focused on the development of a working prototype, not the scaling of features. Furthermore, several APIs needed for higher functionality-features like multilingual speech and screen reading via OCR-required configuration and paid-for subscriptions that fell beyond the remit of a student project. The processing power was also an issue since AI models are quite power-hungry.

©Daffodil International University

13.1.3. Objectives That Have Been Missed

Mainly a few secondary objectives, in particular those concerning cloud-based scaling and mobile synchronization, have not been fully achieved. Although the architecture supports such extensions, they have remained undone given the limitations of deployment and testing time in the initial development phase.

Missed Objective	Reason	Future Consideration
Cloud deployment integration	Limited hosting resources	Integrate AWS or Azure
Mobile synchronization	Out of current development scope	Develop React Native or Flutter companion app
Real-time multi-user support	Complex session management	Add WebSocket-based concurrent communication

13.1.4. Why These Objectives Were Missed

These were less emphasized to avoid instability, making it a complete desktop-based system. Advanced scalability or mobile features would have required adding new frameworks, a bunch of new modules with their distinct features, testing environments, and integration time beyond the academic timeline. The focus remained on the core assistant functions performed flawlessly before expanding to extended domains.

13.1.5. What Could Have Been Done to Complete Those Objectives

This would have made extensions easier if early cloud service setup and modular scaling APIs were integrated. Further, testing could have been more efficient with collaborative development through containerization-Docker-and environment standardization being done earlier, thus reducing the delay in setups. This may allow for the integration of the functionality of cloud-based and mobile before project completion.

13.1.6. How Better Are the Features of the Solution

How much better are the features of the solution? The current system provides significant improvements over conventional desktop assistants through hybrid integration of text, voice, and command functionalities. A modular back-end with a responsive React front-end maintains smoother operations with reduced latency. The smart routing between commands and chat offers much better accuracy compared to traditional fixed-intent assistants.

©Daffodil International University

13.1.7. Features That Could Not Be Touched

Features that Couldn't Be Touched Some features that were planned, like the auto-reading of screen content through OCR and deep integrations with cloud drives-Google Drive and OneDrive-are not implemented at the moment. These require additional third-party APIs and authentication layers, which, due to security and time constraints, were not prioritized for this version.

13.1.8. Why These Features Could Not Be Touched

Why These Features Could Not Be Touched This meant that the OCR-based and cloud-based extensions required a lot of testing, permissions management, and API verification. The implementation of these extensions now in the project timeline was risking system stability. And also, secure storage of the third-party credentials required an advanced encryption layer, which would prolong the project beyond the scheduled defence deadline. So, they were abundant.

13.1.9. What Could Be Done to Touch Those Features

Future work can adopt dedicated API modules for OCR and cloud integration, using libraries like **Tesseract OCR**. Secure key management (via encrypted vaults) will make these features do-able without compromising data integrity or security concerns.

13.2. Objectives Totally Not Met / Touched

Some lower-priority objectives, particularly extended documentation automation and multi-language model support, were not reached. These were considered enhancements rather than core points for this phase. The current version focuses on local AI integration and user interaction, which fulfilled the primary system purpose perfectly.

Unmet Objective	Reason	Possible Improvement
Multi-language support	Model limitation	Add multilingual speech model in next version
Advanced analytics dashboard	Not critical for prototype	Introduce dashboard in next update for performance tracking

13.2.1. Why They Could Not Be Touched

The limited duration of the project and the emphasis on stability over expansion restricted this advanced or aesthetic modules. Some plans are require more time and financial investment to execute.

©Daffodil International University

13.2.2. What Could Have Been Done

If there were more time the missing component could have been touched. More code analysis could have reduce the system size.

But saying that, this project has met its primary objectives quite perfectly. The system is stable, well refined and serves it purpose successfully. Additional time and money investment can improve can continue this project's software development life cycle.

©Daffodil International University

Chapter 14 – Lessons Learned

©Daffodil International University

14.1. Pre-Project – Review – Closing

The possible scope and requirement of this software were well reviewed before starting the development. This pre-project analysis helped in recognizing what technologies whould be better and what will be the best. The overall lifecycle showed how much boost can planning and continuous progress tracking gives to a project.

14.2. What Have I Learned

In the process of this project, I have learned to build a complete software. I learned that field like AI have a race ongoing and this software development has allowed me to participate on this race.

14.3. What Problems I Have Faced

Several problems I have faced during development of this project. Combining local AI and with OpenAI API was quite difficult to execute. Also, the mic was working on the first computer but was disrupted on the second.

14.4. What Solutions Occurred

If I had more time, more time on debugging and testing could have been done. More testing in voice command to improve accuracy even when there is noice around the mic.

©Daffodil International University

Chapter 15 – Conclusion

©Daffodil International University

15.1. Summary of the Project

The AI-Powered General Assistant Software is project design and develop as a intelligent system. This software can chat with the user as a friend, carry conversation and help out completing repetitive task. It supports commands that carry out operation like file generation, organization, face recognition. It runs completely while being offline and can use OpenAI API for complex tasks.

15.2. Goal of the Project

The idea was also to eliminate dependency on third-party cloud-only assistants by offering an offline-compatible model through Ollama integration. Another objective was to make sure of responsiveness and cleanliness in the UI, which would support both typing and speech interaction. The system should be able to generate various file formats from AI-generated content, such as .txt, .pdf, .ppt, and .docx, on command. The other goal involved secure handling of credentials and user data so that privacy and reliability are guaranteed. Ultimately, the project should show how AI can serve as a personal digital partner for academic, professional, and general computing needs.

15.3. Success of the Project

The project had great success in integrating various technologies into one functional system. The backend and frontend work effortlessly. The database works great and protect the data through expression. The software has successfully generated files and successfully carry out operations. The face recognition was also a great feature. Now event photos are deliverable to specific people that matches the face.

15.4. What I Have Done in the Documentation (Stages / Activities / Plans)

This project documented every phase in the SDLC, from vary start to the vary end. Problem identification and requirement analysis by various diagram proved a huge help. In every stage of this documentation clarify the project field and framework. Each chapter contributed to keeping the development process.

15.5. Value of the Project

Value of the Project This project is of both academic and practical importance. In academic field the project brings a next step to AI evolution. Practically, this is a very good model solves a most problematic points of todays Artificial intelligence, having it offline and works with desktop environment.

15.6. My Experience

My experience from this project is both challenging and rewarding. As I worked on full stack development, both backend and frontend with database handling. I got to learn

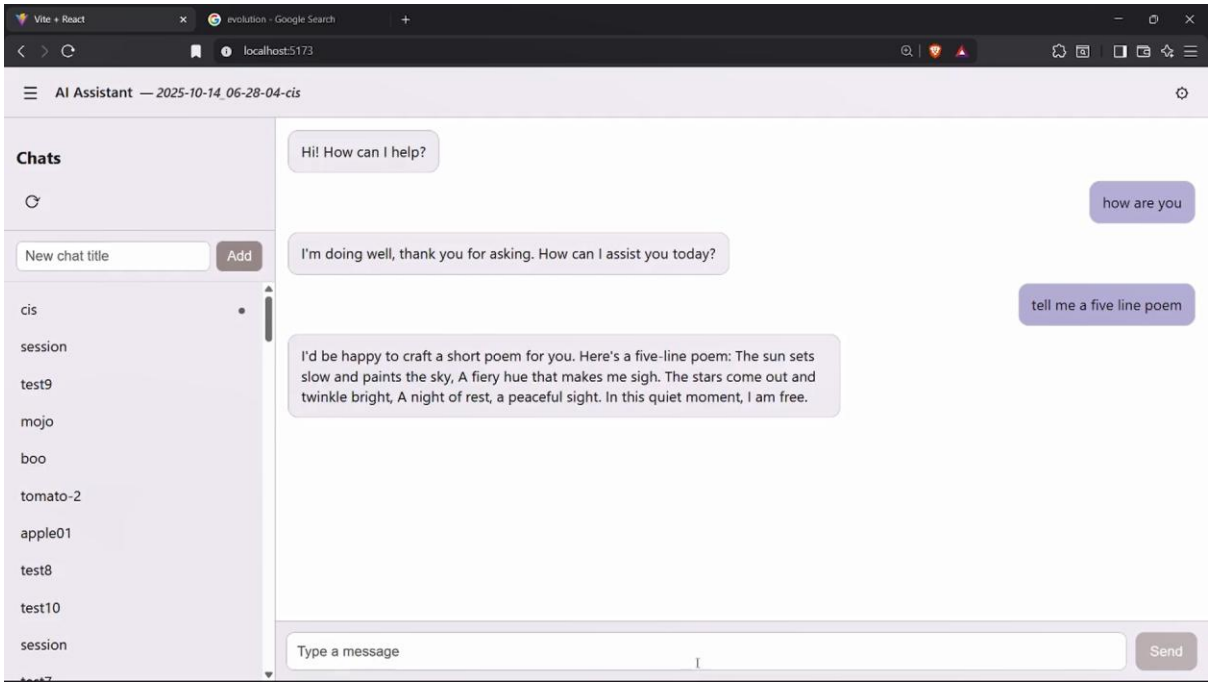
©Daffodil International University

how to deal with real-time interactions. I learned how to handle AI response and how to use it for specific reasons. I learned a lot about data security as well as data privacy. It helps me to think critically and how to apply appropriate problem solving techniques. Most importantly, it increases my appreciation for how AI can be used in building tools that make life so much easier for humans and further improve our digital experience. Overall, this has been an invaluable step toward growing into a software engineer and an AI developer.

©Daffodil International University

Appendices

A: Sample Inputs/Outputs



14. Sample Inputs/Outputs

©Daffodil International University

Reference

1. Brown, T. B., et al. (2020). *Language Models are Few-Shot Learners*. NeurIPS.
2. OpenAI. (2023). *GPT-4 Technical Report*.
3. Touvron, H., et al. (2023). *LLaMA: Open and Efficient Foundation Language Models*. Meta AI.
4. Raffel, C., et al. (2020). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. JMLR.
5. Zhang, X., et al. (2023). *A Survey on Large Language Models: Applications, Challenges, and Future Directions*.
6. Xu, J., et al. (2024). *AutoGen: Enabling Next-Gen Conversational AI with Multi-Agent Collaboration*. Microsoft Research.
7. Nakano, R., et al. (2021). *WebGPT: Browser-Assisted Question Answering with Human Feedback*. OpenAI.
8. Qin, Y., et al. (2023). *Toolformer: Language Models Can Teach Themselves to Use Tools*. Meta AI.
9. Shridhar, M., et al. (2023). *LLMs as OS-Level Agents: An Overview*.
10. Radford, A., et al. (2023). *Robust Speech Recognition via Large-Scale Weak Supervision (Whisper)*. OpenAI.
11. OpenAI. (2024). *Safety Guidelines for Interactive AI Assistants*.
12. Huang, J., et al. (2022). *A Survey on Multimodal Interaction and Speech-Driven Interfaces*.
13. Ramírez, S. (2021). *Building Modern Python APIs with FastAPI*. O'Reilly.
14. Kluyver, T., et al. (2020). *Jupyter and FastAPI for Interactive Python Services*.
15. Abramov, D., & Clark, B. (2023). *The React Handbook*. Meta.
16. Lerner, S. (2021). *Production-Grade React Applications*. O'Reilly.
17. Zettlemoyer, L., et al. (2023). *Seeing and Doing: Integrating Vision Models into Intelligent Agents*.
18. Google Research. (2022). *Screen Parsing for Assistive Technologies*.
19. Anthropic. (2023). *Constitutional AI: Harmlessness from AI Feedback*.

©Daffodil International University

213-16-591

ORIGINALITY REPORT

14%	12%	1%	11%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to University of Greenwich Student Paper	5%
2	dspace.daffodilvarsity.edu.bd:8080 Internet Source	4%
3	Submitted to Daffodil International University Student Paper	3%
4	arxiv.org Internet Source	<1%
5	Submitted to Georgia Institute of Technology Main Campus Student Paper	<1%
6	Submitted to City University Student Paper	<1%
7	Submitted to Arkansas State University, Jonesboro Student Paper	<1%
8	Submitted to Liverpool John Moores University Student Paper	<1%
9	ouci.dntb.gov.ua Internet Source	<1%
10	Ivana Vichentijevikj, Kostadin Mishev, Monika Simjanoska Misheva. "Prompt-to-Pill: Multi-Agent Drug Discovery and Clinical Simulation Pipeline", Cold Spring Harbor Laboratory, 2025	<1%

©Daffodil International University

	Publication	
11	img.ly Internet Source	<1 %
12	ojs.aaai.org Internet Source	<1 %
13	penerbit.uthm.edu.my Internet Source	<1 %
14	"Construction Applications of Virtual Reality, Volume 3", Springer Science and Business Media LLC, 2025 Publication	<1 %
15	Tay, Siok Wah. "A Two-Level Communication-Based Access Control Framework for the Internet of Things", The University of Manchester (United Kingdom), 2024 Publication	<1 %
16	erepository.uonbi.ac.ke Internet Source	<1 %
17	github.com Internet Source	<1 %
18	ojsamik.amikmitragama.ac.id Internet Source	<1 %
19	publisher.uthm.edu.my Internet Source	<1 %

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

©Daffodil International University