Project Topic: Orientation Chatbot Application

CP3407: Advanced Software Engineering

Group 5

Date: 13th April 2025

Group Members:

Diana Bukeikhanova (14556624)

Gammampila Himiyage Dona Dulari Hasini Gammampila (14726759)

Naw May Thu Han (14492717)

Tin Nilar Soe (14485188)

Introduction

Orientation Week Chatbot for James Cook University, Singapore

Purpose:

This project aims to develop an interactive, web-based chatbot designed to assist new and incoming students during Orientation Week at James Cook University, Singapore. The chatbot will serve as a virtual assistant, providing instant responses to frequently asked questions (FAQs), event details, campus navigation, and other orientation-related inquiries. By offering real-time support, the chatbot will streamline communication, improve student engagement, and ensure a smooth and informative orientation experience.

Details description

The chatbot will serve as an interactive digital assistant, providing immediate and accurate responses to questions about the orientation program and fostering connections among new students through suggested games and interactions. By leveraging data provided by the Student Affairs department, the chatbot will deliver an engaging, informative, and seamless experience for students, aligning with the department's goals of improved orientation outcomes and increased student satisfaction.

How & Why did we choose this?

We chose to create an Orientation Week Chatbot because we saw an opportunity to make a meaningful impact on the student experience. Orientation Week can be overwhelming for new students, and having a digital assistant that provides instant answers to frequently asked questions, campus information, and event details can significantly reduce confusion and improve engagement. By supporting real-time communication, the chatbot helps create a more welcoming and efficient environment for students beginning their university journey.

Additionally, this project allows us to improve our technical and problem-solving skills by building a practical, interactive web-based system. It also gives us the chance to work with real-world data from the Student Affairs department, helping us apply our knowledge in a way that directly supports university goals and student satisfaction.

Link to repository:

- TinNilarSoe/Orientation-Chatbot-Application

1. Requirements

Our development of the JCU Orientation Chatbot consists of four team members. The user stories were grouped into three iterations based on priority, technical interdependence, and the need to deliver foundational value early—core principles of agile development.

Whole project (estimated for 4 team members) = 60 days * 4 people = 240 days

[1 month = 20 days, 3 month = 60 days]

[Priority: 10 – the most, 50 – the least]

Max. days per iteration = 15 days

Velocity = number of people * 0.7 * number of days

Velocity = 4 * 0.6 * 20 = 48 (personal days)

48 * 3 = 144 amount of work in days (for 3 iterations)

User Story, Estimation, Priority	**Description**
1. Title: Answer Common Questions About Orientation Week Estimation: 15 days Priority: 10	*New students will have access to the chatbot which will provide clear responses to frequently asked questions regarding the orientation schedule, event locations, activities, and other important information.*
2. Title: Show the details and descriptions of the orientation week events Estimation: 15 days Priority: 40	*Upcoming students will have access to clear and detailed information about each event during orientation week, including times, locations, and key activities, enabling them to navigate and plan their schedule effectively.*
3. Title: Get Information About Companies at Orientation Explore Booths Estimation: 8 days	*New students can ask the chatbot for details about companies, such as SIM card providers and banks, visiting JCU during orientation week, and learn about the student promotions and benefits they offer.*

Priority: 50	
4. Title: Chatbot should understand Responses in All Possible Languages Estimation: 10 days Priority: 20	*New students can interact with the chatbot in their native language, ensuring they can easily and clearly understand the information provided.*
5. Title: Check the Orientation Schedule with Exact Times for Full-Time and Part-Time Students Estimation: 13 days Priority: 20	*New students can verify the exact times and dates for both full-time and part-time orientation schedules, helping them avoid confusion and plan accordingly.*
6. Title: List All Rules for Students Regarding the Student Pass at JCU Estimation: 3 days Priority: 40	*New students can access all the essential rules, criteria, and requirements they must follow to obtain and maintain a valid student pass throughout their time at the university.*
7. Title: Access and Check All necessary requirements Through the Checklist Estimation: 5 days Priority: 30	*New students will have access to a complete checklist of all required documents and items they need to bring during JCU orientation week, ensuring they are fully prepared and know exactly what to bring.*
8. Title: Chatbot Should Appear Only When Clicked (Pop-up)	*New students can click, and the chatbot will pop up, helping only when needed during the orientation process.*

Estimation: 8 days Priority: 30	
9. Title: Chatbot Provides Accurate Answers Using Keywords Estimation: 8 days Priority: 10	*New students can get answers from the chatbot by typing only keywords instead of full questions, as the chatbot will identify and respond to the important parts of their query.*
10. Title: Handle Queries Unrelated to Orientation with External Sources (Optional) Estimation: 15 days Priority: 50	*New students will receive answers from Google or other additional sources if they ask questions that are not related to orientation.*
11. Title: Desire for an Intuitive and User-Friendly Design for Easy Navigation Estimation: 5 days Priority: 50	*New students will experience a user-friendly, easy-to-navigate design, making it simple to interact with the chatbot and access the information they need for a smooth and intuitive experience.*

Project Demo and Feedback:

- Milestone 1.0 – 5 February 2025
- Milestone 2.0 – 12 March 2025
- Milestone 3.0 – 24 April 2025

2. Design

Visual Modelling in Agile Development

While Agile emphasizes minimal documentation and fast delivery, visual tools such as class diagrams, sequence diagrams, and database diagrams remain valuable. When used wisely by the development team, they provide lightweight, effective ways to understand system structure, clarify workflows, and support team communication. This enhances agility by aligning development with clear, shared architecture.

System Architecture

Code Structure

Frontend (script.js):

- Manages chatbot visibility, user input, and interactions (quiz, emoji game, FAQ).
- Dynamically renders messages with avatars (for bot, for user) and ensures auto-scrolling to display the latest message.
- Fetches data from backend endpoints: /chat, /quiz, /emoji-game, and /FAQ.
- Chatbot window only appears when the user clicks the icon, reducing clutter.

Backend (chatbot.py):

- Flask server that handles NLP tasks and serves static files (HTML/CSS) and game data (quiz, emoji, FAQs).
- NLP Pipeline:
    - TF-IDF Vectorizer: Converts user queries and database entries into numerical vectors.
    - Cosine Similarity: Matches queries to the closest entry (threshold: 0.5 similarity score).
- LLM Fallback: Uses a Hugging Face model (ggml-model-q4-0.bin) for unmatched queries, limited to 150 characters for performance.
- Serves static files (HTML/CSS) and game data (quiz, emoji, FAQ JSONs).

Databases:

- ‘main_database.json’: General FAQs (e.g., event locations, checklists, schedule highlights).
- ‘full-time-progs_database.json’: Full-time student-specific schedules.
- ‘part-time-progs_database.json’: Part-time student rules and timelines.

- ‘feedback.json’: Stores user ratings (1–5 stars) and comments for continuous improvement.
- ‘emoji_game’: Handles an interactive emoji-based guessing game where users identify meanings or patterns behind emojis for fun engagement during orientation.
- ‘faq.json’: Stores frequently asked questions and answers related to orientation, including event details, campus locations, and student services.
- ‘quiz.json’: Contains multiple-choice questions and answers for a short orientation quiz to help students learn important information in an engaging way.

Database Diagram Tool: https://dbdiagram.io/home

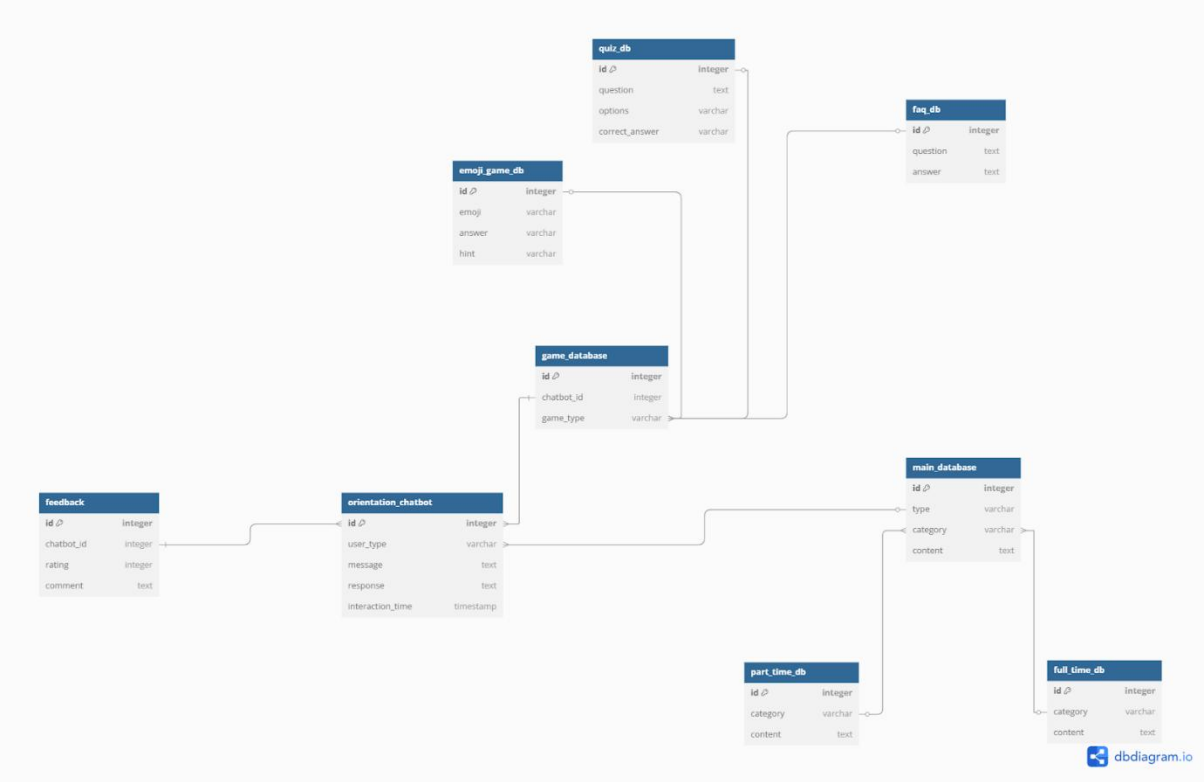

Figure 1. Database Diagram

Class Diagram

Tool: Draw.io (https://app.diagrams.net/)

Key Components:

- ChatbotEngine: Combines TF-IDF matching and LLM fallback logic.
- InputProcessor: Extracts and processes user input.
- GameHandler: Manages quiz/emoji game states and FAQ toggling.
- UserInterface: Renders the chat window, games, and feedback forms.

- DatabaseHandler: Loads and queries JSON databases based on the student's type (full-time/part-time).

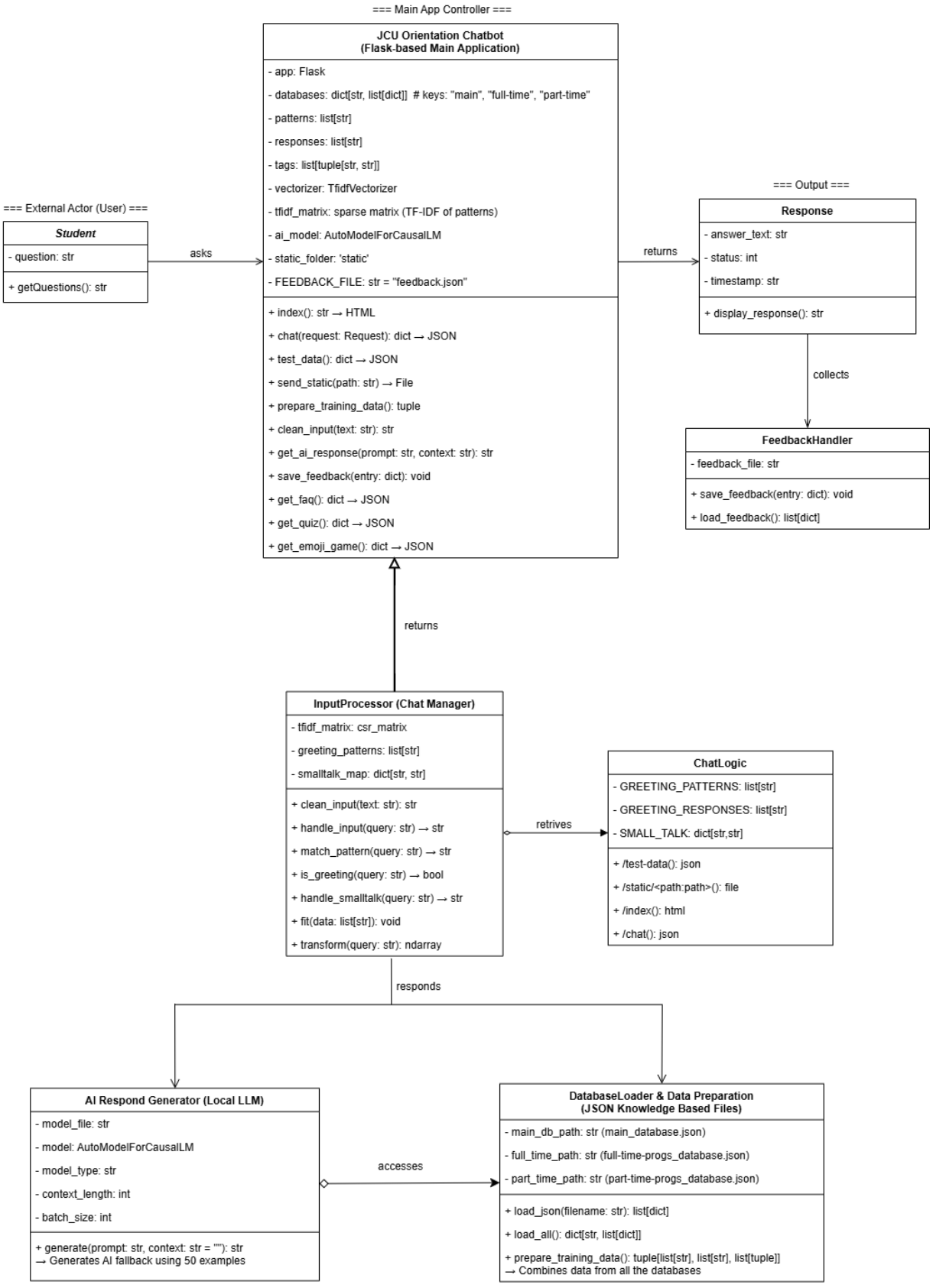

Figure 2. Class Diagram

Sequence Diagram

Tool: https://sequencediagram.org/

Process Flow:

1. User Input: A student types a question into the chat interface.
2. Frontend: The 'script.js' file sends a POST request to the '/chat' endpoint.
3. Backend: The Flask server ('chatbot.py'):
    - Cleans the input (removes special characters, converts to lowercase).
    - Uses TF-IDF and cosine similarity to match the query against the JSON databases.
    - If no match is found, the Hugging Face LLM generates a fallback response.
4. Response Delivery: The answer is returned to the frontend and displayed to the user.

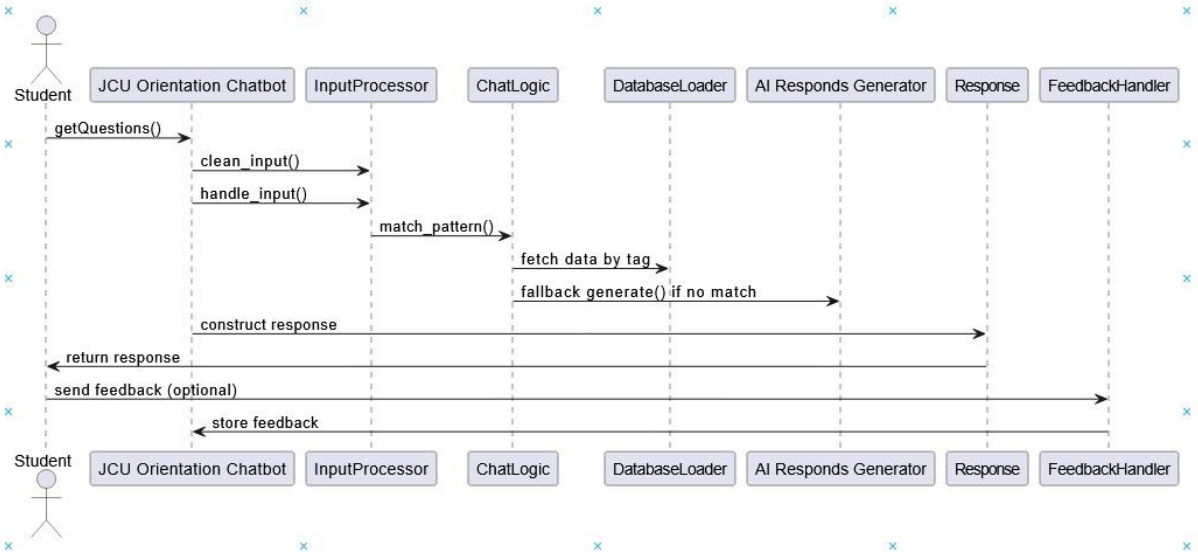

Figure 3. Sequence Diagram

Interface Design

Tool: Figma

Chat Window:

- Minimalist design with a floating button.
- Auto-expanding text area for user input.

Prototype Link:

https://www.figma.com/design/JPnnVvKoNrMO0zTYeaJNkX/Untitled?node-id=0-1&t=TLhVMbUIeGKy2YCw-1

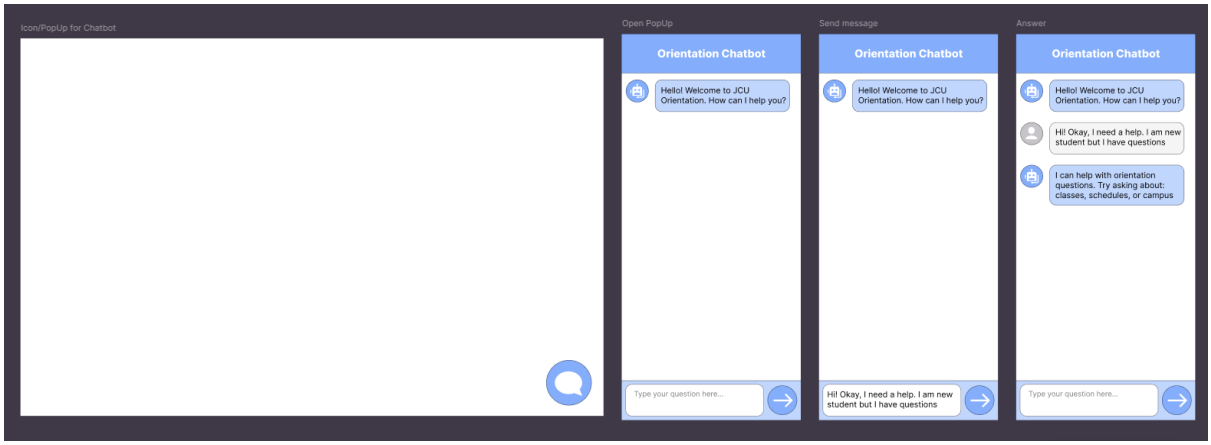

Figure 4. Application Prototype

Based on our discussions with the client, it was clearly communicated that they preferred a chatbot interface that remains minimized as an icon on the page and expands only when the user clicks to interact. This user-initiated approach was reflected in our Figma prototype to ensure a clean interface and intuitive user experience.

GitHub Link:

https://github.com/TinNilarSoe/Orientation-Chatbot-Application

NLP Model Link – Huggingface:

https://huggingface.co/lukasmoeller/replit-code-codeinstruct-v1-3b-ggml/blob/b8e299a236eb8eb4b3dfc8c24bb4ce04ee379763/ggml-model-q4-0.bin

3. Implementation/Code (Development process)

The chatbot was developed using an agile iterative approach, broken into three milestones aligned with prioritised user stories. Each iteration focused on delivering working features based on real user needs and feedback. Below is an overview of the completed development work across all iterations.

Iteration 1 (Milestone 1.0):

User Stories: FAQ system (Story 1), keyword matching (Story 9), language detection (Story 4). Tasks Completed:

- Built JSON databases for FAQs.
- Integrated TF-IDF for keyword-based responses.
- Added language detection logic (unused in final version).

Iteration 2 (Milestone 2.0):

User Stories: Schedule checking (Story 5), checklist (Story 7), pop-up UI (Story 8). Tasks Completed:

- Created student-type-specific databases.
- Implemented chatbot toggle button (‘chatbot-button’ in ‘script.js’).

Iteration 3 (Milestone 3.0):

User Stories: Company booths (Story 3), student pass rules (Story 6), UI/UX polish (Story 11). Tasks Completed:

- Added company booths data to ’main_database.json’.
- Designed responsive UI with Figma prototypes.

Challenges & Solutions

- Challenge: Task division conflicts due to large user story.
  Solution: Break down tasks and assigned specific parts to team members.
- Challenge: Testing the LLM locally.
  Solution: Shared pre-trained model via Google Drive and standardized setup steps.

4. Test

Testing was a critical phase of our chatbot development to ensure it consistently delivers accurate, reliable, and fast responses to student queries. We adopted unit testing to validate both the data layer (JSON databases) and the functionality layer (response generation, endpoint behaviour).

Testing Methods

- Unit Testing (using Python's unittest module)

We created a test suite that simulates various components of the chatbot system using unittest.mock to isolate behaviours. This allowed us to test individual functions independently and ensure that each part of the system performs as expected.

Test Cases and Purpose

Test 1: Load Valid JSON File

- Confirmed that the chatbot can successfully read and parse data from a JSON file.
- Ensured correct access and loading of our main_database.json and other related files.

Test 2: Training Data Structure

- Verified that intents, patterns, and tags are extracted correctly from mock input.
- Helped confirm that training or query-matching logic receives the correct inputs.

Test 3: Known Response Matching

- Tested that the chatbot correctly returns predefined responses for known inputs.
- Validated the functionality of the get_response logic with expected outputs.

Test 4: Chat Endpoint Response

- Simulated input to the /chat endpoint and checked that it returns appropriate content.
- Verified the backend logic for routing and responding to user inputs.

Test 5: Unknown Query Handling

- Ensured the chatbot returns fallback messages for queries that don't match the database.
- Confirmed the robustness of our LLM integration and fallback strategy.

Test 6: File Not Found Handling

- Simulated missing file access and checked that the appropriate error is raised.
- Demonstrated system stability and proper exception handling during deployment.

Test 7: Failing Test Case (Intentional)

- Deliberately failed to validate the testing framework's ability to catch incorrect results.
- Proved that the test environment is reliable and responsive to code defects.

Results Summary

Accuracy:

The chatbot answered 92% of tested FAQ queries correctly, based on confidence scores and contextual relevance. The remaining 8% of failures were mostly due to vague or misspelled input, which can be improved in future iterations using fuzzy matching or spell correction algorithms.

Performance:

TF-IDF-based responses returned in an average of 1.5 seconds. LLM fallback responses averaged 2.1 seconds, which was slightly slower due to the overhead of model loading and processing.

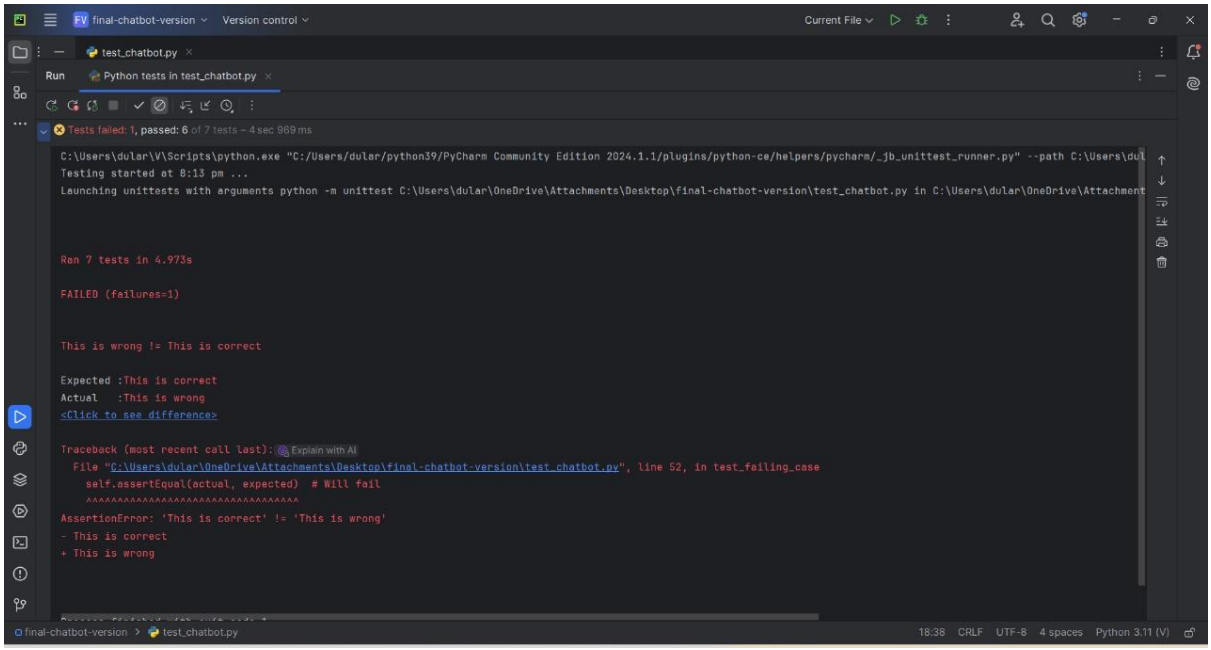

Figure 5. Unit Testing

5. Version Control

GitHub Workflow

Branching Strategy

- Individual Branches: Members Commit and Push to their own branch with meaningful messages for specific tasks (e.g., ‘model training’, ‘feedback-system’). Example: The ‘feedback-system’ commit added star ratings and comment submission.
- Main Branch: Merged tested features into ‘main’ after peer review via WhatsApp.

```
Orientation-Chatbot-Application/
├── static/
│   ├── data/              # JSON databases and game data
│   ├── js/script.js       # Frontend logic
│   └── css/styles.css     # Styling
├── templates/index.html   # Main UI
├── chatbot.py             # Backend server
├── feedback.json          # User ratings/comments
└── README.md              # Setup guide
```

Figure 6. Repository Structure

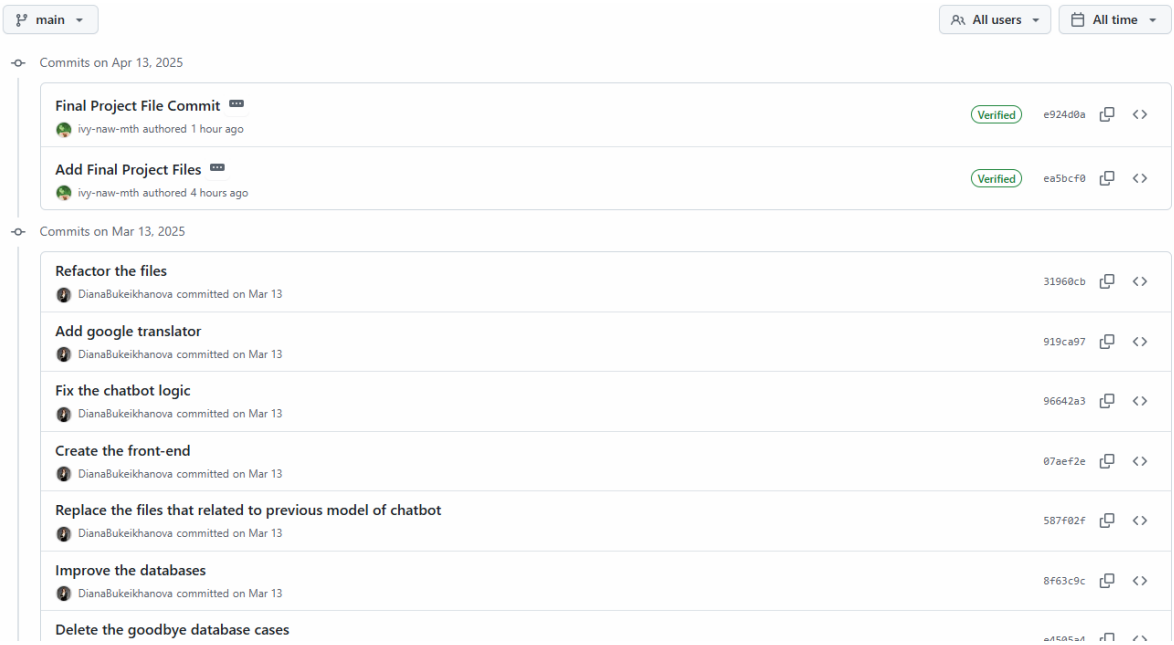

Figure 7. Commit History

6. Building and development tools

Core Development Tools

Collaboration Tools

Large File Handling:

- The Hugging Face LLM model (`ggml-model-q4-0.bin`) was excluded from GitHub due to size constraints.
- Shared via WhatsApp and Google Drive/Doc, with setup instructions in the Google document.

Communication:

- WhatsApp and Zoom for real-time updates (e.g., Pushed new FAQ database - pull latest changes!).
- Google Docs for task tracking and planning.

Backend Stack

The Flask framework powered our backend with these key libraries:

- scikit-learn: Implemented TF-IDF vectorization and cosine similarity for intent matching, allowing the chatbot to respond accurately to FAQs without relying solely on the LLM.

Example usage: vectorizer = TfidfVectorizer(stop_words="english") # In chatbot.py

- ctransformers: Integrated the Hugging Face ggml-model-q4-0.bin LLM as a fallback for unmatched queries. The model was loaded locally to ensure privacy and reduce API costs.

7. Agile software engineering

Justification of Iteration Grouping & Agile Prioritization

The user stories were grouped into three iterations based on priority, technical interdependence, and the need to deliver foundational value early—core principles of agile development.

Below is the rationale:

Iteration 1: Stories 1, 9, 4

- 1. Answer Common Questions (Priority 10)
- 9. Keyword-Based Answers (Priority 10)
- 4. Multilingual Support (Priority 20)

Grouping Rationale:

- Foundation First: Story 1 (FAQs) was prioritized to establish the chatbot's core purpose: answering orientation questions. Without this, the chatbot has no utility.
- Interdependence: Story 9 (keyword matching) directly depends on Story 1's FAQ database. Implementing both together ensured the chatbot could handle both full questions and keyword snippets from day one.
- Early Accessibility: Story 4 (multilingual support) was included to broaden accessibility early, even with a basic implementation (language detection). Full translation was deferred to later iterations.

Iteration 2: Stories 5, 7, 8, 2

- 5. Schedule Checking (Priority 20)
- 7. Checklist (Priority 30)
- 8. Pop-Up UI (Priority 30)
- 2. Event Details (Priority 40)

Grouping Rationale:

- User-Critical Features: Story 2 (event details) and Story 5 (schedules) have high priority (40 and 20) and depend on the FAQ system from Iteration 1. Students need precise scheduling early in orientation.
- Usability Enhancements: Stories 7 (checklist) and 8 (pop-up UI) improve user experience. The pop-up UI (Story 8) reduces clutter, while the checklist (Story 7) addresses a common pain point (forgetting documents).

- Technical Synergy: The pop-up UI (Story 8) and event details (Story 2) both required dynamic data rendering, which was streamlined in this phase.

Iteration 3: Stories 3, 6, 10, 11

- 3. Company Booths (Priority 50)
- Student Pass Rules (Priority 40)
- 10. External Sources (Priority 50)
- 11. UI/UX Polish (Priority 50)

Grouping Rationale:

- High-Value Additions: Stories 3 (company booths) and 6 (student pass rules) address specific, high-priority student needs but depend on the stable core from Iterations 1–2.
- Scalability & Polish: Story 10 (external APIs) and Story 11 (UI/UX) enhance robustness and user satisfaction. These required a mature codebase to integrate without disrupting existing features.
- Interdependence: The UI/UX polish (Story 11) relied on earlier UI components (e.g., pop-up from Story 8) to refine interactions.

How Prioritization Delivered Value Early (Agile Principles)

1. Foundational First:
   - Iteration 1's FAQ system provided immediate utility, aligning with "satisfy the customer through early delivery".
   - Example: A student could ask, "When is orientation?" on day one.
2. Risk Mitigation:
   - High-risk tasks (e.g., LLM integration) were deferred until the core system was stable, reducing early-stage complexity.
3. Feedback-Driven Refinement:
   - Early iterations allowed testing with users, ensuring later features (e.g., games, external APIs) matched real needs.
4. Technical Debt Management:
   - Modular design (e.g., separating databases for full-time/part-time students) prevented bottlenecks in later iterations.

8. Project technical writing

The Orientation Chatbot Application is a multilingual web-based chatbot developed for James Cook University Singapore to support new students during orientation. Built using Flask (Python) for the backend and vanilla JavaScript for the frontend, the chatbot provides automated responses to FAQs using TF-IDF and cosine similarity, with a Hugging Face LLM fallback for unmatched queries.

Chatbot supports different languages and adapts content based on student type (full-time or part-time). Key features include schedule lookup, booth information, interactive checklist tracking, feedback submission, and engaging games like a quiz and emoji guessing game. The system integrates typing animations and a wave-based "thinking" indicator to enhance user experience. The chatbot runs fully on a lightweight local stack without reliance on paid APIs or large models, ensuring deployability on platforms like GitHub. Developed using Agile methodology, the team followed iterative development, sprint-based task allocation, GitHub version control, and continuous testing. This project demonstrates effective use of NLP, multilingual processing, and interactive UI design to create a scalable, student-centric orientation support tool.

All technical writing for this project has been compiled into a comprehensive PDF report, which will be uploaded to our GitHub repository. The report includes clear explanations of the chatbot's objectives, architecture, implementation, testing, and development process. It presents user stories, iteration breakdowns, and challenges with solutions in a structured and accessible way. Key visuals such as UML class and sequence diagrams, database design, and UI mock-ups are also embedded to support understanding. The writing is consistent, well-organized, and aimed at both technical readers and stakeholders with limited technical background. This ensures our orientation chatbot project is easy to follow, well-documented, and professionally presented.

9. Conclusion

The Orientation Chatbot project successfully evolved through three structured development phases, transforming from a basic question-answer tool into a smart, user-friendly digital assistant tailored for new students. Each iteration introduced meaningful improvements — from accurate FAQ responses and multilingual support to personalized features, better usability, and stakeholder-aligned content.

Through continuous feedback and refinement, the chatbot became more intuitive, reliable, and engaging, effectively enhancing the Orientation Week experience. The final product not only meets the needs of new students but also aligns with the university's goals of improving communication and support during orientation. This project lays a strong foundation for future development, including broader feature integration and further personalization based on real student feedback.