Historical Figure Chatbot – Project Report (Software Engineering 25/26)

Project Report – Group 36

Group Members:
Edison Ford (s5718375**), Hugh Wooll (**s5713423**), James Marsh (**s5719014**), Peter McNealy (**s5713510**)**

PROJECT SUMMARY

This Chatbot will be primarily used as an online assistant chatbot showcasing the life of Shakespeare for students who are primarily studying English literature as well as history (in the Elizabethan era). The chatbot will serve as an educational tool as well as an online exhibit which can be accessed from either within a real-life demonstration, or from a personal device with a key goal of giving the opportunity of interactivity to everyone in or out of a school setting. By allowing users to directly talk to this conversational agent, the system gives a personalised approach of learning to students and those who are curious about him, allowing for them to freely explore Shakespeare's life, literacy works and cultural significance.

We specifically decided Shakespeare as our historical figure since he is an influential staple on global literature, English language and theatre performances. As well as Shakespeare's linguistic influence is well documented, particularly his use of Early Modern English and rhetorical structures (Crystal, 2008). However, his great influence on these aspects of history is at times challenging to understand, especially when spoken in old English dialect. This is why we believe a chatbot is the perfect solution to get people to talk to Shakespeare 1 on 1 to understand the dialect and history through fun and creative interactions. The Shakespeare chatbot will therefore enrich old English literacy as well as give historical awareness. With users being able to prompt questions to Shakespeare, learning about his work/life, they may even be able to get him to write poems and sonnet when asked to. These creative capabilities further encourage the support of different learning styles, allowing for adaptability to all users.

The chatbot's design philosophy is user focused to make sure the interface is accessible and viewable for all ages by complying with the WCAG guidelines standards. During interactions with the chatbot we also want to make sure that it stays true to Shakespeare's historical context and dialogue traits, increasing the authenticity and emersion, making sure all users experience Shakespeare as a historically accurate chatbot. To maintain this historical accuracy, the chatbot's prompts and scripted responses are hardcoded to be told to avoid misinformation and production of fake events and only reference verified sources covering Shakespeare's biography.

The project extends beyond class taught content as this chatbot will also integrate the use of the google Gemini API which will go alongside the JSON scripting engine. This should allow for a real-time generation of creative responses that are not specified in the scripting engine, giving the chatbot more flexibility for answering questions. We also used GitHub (GitHub, Inc., n.d.) as our primary collaboration platform.

Historical Figure Chatbot – Project Report (Software Engineering 25/26)

REQUIREMENTS

Most of these requirements have been gathered thoroughly by brainstorming as a team. We contributed to researching multiple popular chatbots to gain insights and ideas about what kind of features the system should require, with the main references being ChatGPT, Gemini and DeepSeek. Upon comparing the different features of these chatbots we found all 3 had an incredibly similar features and design, and so we decided to take inspiration from all of them and follow with the same features. We decided that having similar features to popular chatbot can give familiarity with navigation and help users grasp the available tools quicker. This was further tested when we conducted questions to friends and family who regularly use chatbots about possible features they would want, and they all listed out similar features including different "chat conversation histories" and being able to "see responses quickly".

Therefore, we listed out all the different features on the various Chabot sites and then made them atomic, making sure each feature is decomposed so that it can be re-worded into multiple requirements (for example: "chat conversation histories" were split into requirements F9, F11, F12, F13, F14). These requirements were then separated into functional / non-functional and grouped into "Shall", "Should" and "May" specifications. We then gave them priority using the MoSCoW method of "Must", "Should", "Could" and "Won't. Finally, each requirement was put into a category of what development area it covers.

After creating the initial requirements, we found that many of the requirements were written in different styles and standards. Therefore, to standardise wording we used ChatGPT to review each requirement, giving us valuable instant feedback. This allowed us to ensure each requirement was atomic and specification appropriate, leading to requirements that lacked clarity such as "The AI response time shall be appropriate" to be more verifiable (NF1 – "The AI response time shall not exceed 5 seconds under normal circumstances").

Over multiple weeks, we would create more requirements after learning new unit materials, for example following WCAG guidelines (NF12) for accessibility focused specifications. This then led to even more accessibility requirements being created, and after review by the team first –then ChatGPT to double check, we completed creating the new requirements for accessibility. This iterative, Agile-inspired process was what we used to fully develop our requirements from initial ideas to final specifications.

Functional Requirements

ID	Priority	Category	Requirement	Status
F1	Must	User Interaction	The system shall have a message bar for users to send a message the chat	Complete
F2	Must	User Interaction	The system shall provide a "Send" button that allows users to submit their messages	Complete
F3	Must	User Interaction	The system shall display the user's message in the chat window in chronological order	Complete
F4	Must	User Interaction	The system shall allow for the user to create an account	Complete
F5	Musts	User Interaction	The system shall allow for the user to log in and log out	Complete
F6	Must	User Interaction	The system shall generate an AI response automatically after the user sends a new message	Complete
F7	Must	User Interaction	The system shall display the AI's responses in a clear and readable format	Complete

F8	Must	User Interaction	The system shall include a visual theme inspired by Shakespearean-era design elements	Complete
F9	Must	User Interaction	The system shall maintain a conversation history, allowing users to scroll through past interactions	Complete
F10	Must	User Interaction	The system shall process modern or informal user language without errors	Complete
F11	Must	User Interaction	The system shall provide an add chat function to start a new conversation	Complete
F12	Must	User Interaction	The system shall allow users to open an existing chat history and display all messages associated with that conversation	Complete
F13	Must	User Interaction	The system shall provide a rename function to existing chats, allowing for naming conventions	Complete
F14	Must	User Interaction	The system shall provide a delete chat function so that users can clear conversation histories	Incomplete
F15	Must	User Interaction	The system shall include an introductory message from “William Shakespeare” upon starting a new conversation	Complete
F16	Must	User Interaction	The system shall offer a help section explaining all system features and how to use them	Complete
F17	Must	User Interaction	The system shall store and retrieve user-specific chat histories based on their login credentials	Complete
F18	Should	Security	The system should save each user message and corresponding chatbot response to the database	Complete
F19	Should	User Interaction	The system should prevent users from sending messages faster than a configured limit	Complete
F20	Should	User Interaction	The system should have persistent chat sessions, even when the user refreshes or re-opens the browser	Complete
F21	Could	User Interaction	The system may provide a retry option for responses if a message delivery fails (Persistent requests)	Complete
F22	Must	AI Behaviour	The system shall ensure that the AI’s responses are consistent in the style and vocabulary of William Shakespeare	Complete
F23	Could	AI Behaviour	The system may provide a “learn more” page, giving historical context or definitions of Shakespearean terms	Incomplete
F24	Could	AI Behaviour	The system may offer a plain English mode as an alternative to the Shakespearean style	Incomplete
F25	Could	AI Behaviour	The system may offer additional persona modes inspired by Shakespearean characters	Incomplete
F26	Won’t	AI Behaviour	The system may offer a “verbosity” setting to adjust AI response length	Incomplete
F27	Won’t	AI Behaviour	The system may allow users to add a token limit for the ai to reduce API cost.	Incomplete
F28	Could	Accessibility	The system may provide audio output for AI responses in a Shakespearean-like voice	Incomplete
F29	Won’t	Accessibility	The system may integrate speech-to-text for a hands free-interaction	Incomplete

F30	Won’t	Accessibility	The system may allow customization of the chat interface (fonts, themes, background) for options of contrast	Incomplete
F31	Won’t	Accessibility	The system may allow users to change the language of all texts and responses	Incomplete

Table 1. List of functional requirements, displaying priority accessiblity, description and status of completion

Non-Functional Requirements

ID	**Priority**	**Type**	**Requirement**	**Status**
NF1	Must	Performance	The AI response time shall not exceed 5 seconds under normal circumstances	Complete
NF2	Must	Security	The system shall ensure all user data is handled securely and privately	Complete
NF3	Must	Security	The system shall hash all user passwords before storing them in a database, in accordance with GDPR security principles	Complete
NF4	Should	Security	The system should provide a high-level description of how user data is stored and protected	Complete
NF5	Should	Security	The system should encrypt data in transit using HTTPS	Complete
NF6	Should	Security	The system should invalidate user sessions after 10 minutes of inactivity	Incomplete
NF7	Must	Accessibility	The system shall be available 99% of the time, excluding maintenance, assuming the user has a working internet connection	Complete
NF8	Must	Accessibility	The system shall be accessible from a mobile device	Complete
NF9	Must	Accessibility	The system shall offer a dynamic layout that adapts to different screen sizes (up to 200% zoom)	Complete
NF10	Must	Accessibility	The system shall be accessible via a standard web browser without additional plugins	Complete
NF11	Must	Accessibility	The system shall be compliant with the WCAG 2.2 to an industrial standard (AA)	Complete
NF12	Should	Accessibility	The system should provide clear error messages when issues occur	Complete

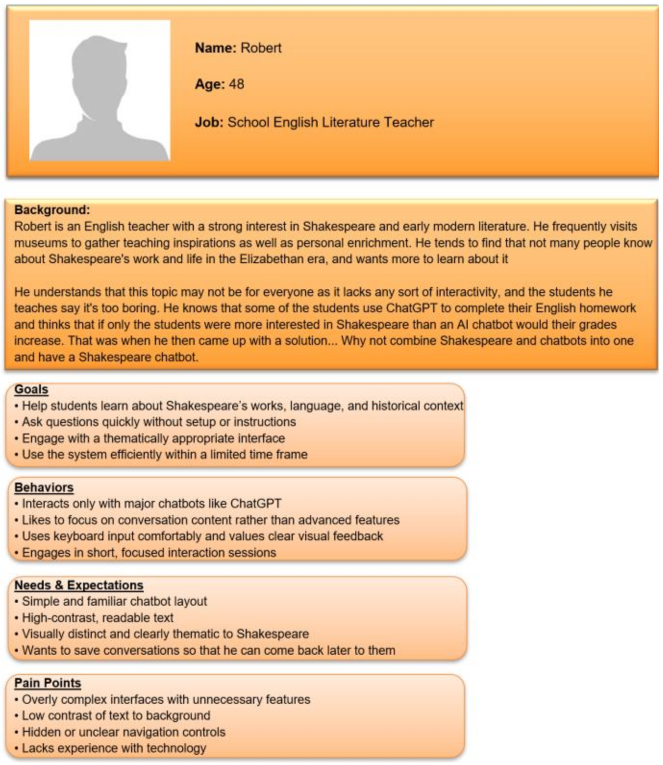
Table 2. List of non-functional requirements, displaying priority accessiblity, description and status of completion

Historical Figure Chatbot – Project Report (Software Engineering 25/26)

DESIGN

From a user experience perspective our overall design choice is to opt for a similar foundation of other major AI chatbot websites since these websites are what most students are familiar with and use on a regular basis. These websites include ChatGPT, Gemini and DeepSeek. All these websites upon login have similar features and design choices. As all of these are highly used websites it is logical that users who have previously used these chatbots will be able to transfer their existing knowledge onto our designed chatbot, allowing for seamless and intuitive navigation due to a familiar interaction model, this is also in ordinance with the persona (Figure 1).

From a technical perspective our design will have a frontend interface, built using the modern web framework react, chosen for its component-based architecture using the virtual DOM, allowing for dynamic updates to the user interface. A backend API layer which serves as a middleman for the frontend and AI model. The AI model will be designed so that the backend communicates with a selected model, giving us more possibilities for features in the future. And a JSON scripting engine used to detect inputted keywords as well as a JSON file for user data storage to keep chat histories of certain users.



Name: Robert

Age: 48

Job: School English Literature Teacher

Background:
Robert is an English teacher with a strong interest in Shakespeare and early modern literature. He frequently visits museums to gather teaching inspirations as well as personal enrichment. He tends to find that not many people know about Shakespeare's work and life in the Elizabethan era, and wants more to learn about it

He understands that this topic may not be for everyone as it lacks any sort of interactivity, and the students he teaches say it's too boring. He knows that some of the students use ChatGPT to complete their English homework and thinks that if only the students were more interested in Shakespeare than an AI chatbot would their grades increase. That was when he then came up with a solution... Why not combine Shakespeare and chatbots into one and have a Shakespeare chatbot.

Goals
- Help students learn about Shakespeare's works, language, and historical context
- Ask questions quickly without setup or instructions
- Engage with a thematically appropriate interface
- Use the system efficiently within a limited time frame

Behaviors
- Interacts only with major chatbots like ChatGPT
- Likes to focus on conversation content rather than advanced features
- Uses keyboard input comfortably and values clear visual feedback
- Engages in short, focused interaction sessions

Needs & Expectations
- Simple and familiar chatbot layout
- High-contrast, readable text
- Visually distinct and clearly thematic to Shakespeare
- Wants to save conversations so that he can come back later to them

Pain Points
- Overly complex interfaces with unnecessary features
- Low contrast of text to background
- Hidden or unclear navigation controls
- Lacks experience with technology

Figure 1. Full Persona of an English Literature teacher who would like an AI chatbot of Shakespeare

System Design (Technical / Architecture)

The frontend provides a simplistic user interface. This is a purposeful decision seeing as it is used to tutor individuals studying Shakespeare and his poems and plays. We have also implemented a check to ensure the server is running before rendering anything on the client side to prevent errors to do with the server influencing the frontend upon sending messages to the chatbot or trying to login. The messaging systems purposely requires logging in and/or signing up so that the user can get the full usage out of the application. We handled large amounts of logic like scripts and predetermined responses on the server's architecture. The client architecture has logic as well have has been kept limited to reduce strain on the user's system and reduce wait times. The frontend is responsible for managing user interaction, maintaining session states and when required sending and receiving data.

The backend receives requests form the frontend, validates inputs, applies logic and generates appropriate responses. It also manages communication with the database for storing and retrieving data. The backend exposes a set of defined endpoints that the frontend can interact with. The backend carries scripting logic with predefined response, for any out of bounds responses we use a request to Gemini with one system prompt, which is very in depth and then followed with a clear showing of user message and then backend receives said request for dynamic responding to user inputs.

Data Flow

Messaging:

User -> Frontend -> Request -> Backend -> Script Logic -> response -> POST -> Frontend -> Rendered in chat page

Login:

User -> Login -> Request -> Backend -> Validation -> User Data -> POST -> Frontend -> Context

Signup:

User -> Signup -> Request -> Backend -> Database

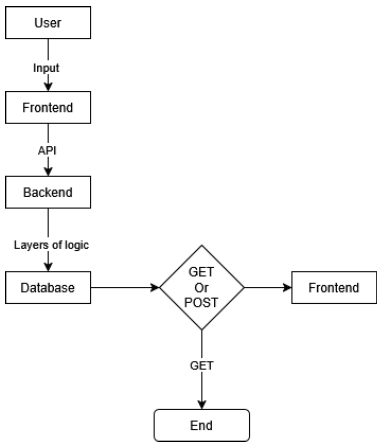


Figure 2. Simple data flow diagram of the whole system design

Key Design Decisions

Client–server architecture
Separates user interface logic from backend processing, improving maintainability.

REST-style endpoints
Enables clear communication between frontend and backend using structured requests and responses.

Modular design
Backend logic is separated into independent modules, making the system easier to extend and test. Similarly, the Frontend is broken into custom components and logic to accompany components.

Stateless requests where possible
Each request contains all required information, reducing server-side dependency on session state.

Centralised User Data Storage

User data is all stored within the same JSON, this is to simplify data storage and usage. Holding the username and password followed by an array of chats.

Historical Figure Chatbot – Project Report (Software Engineering 25/26)

User Interface Design

Our primary UI goal is to create a simplistic, intuitive and historically themed interface that fits with Shakespeare and is accessible to a wide range of users. To achieve this goal, we decided to take UI inspiration from other major AI chatbot websites because our primary target audience are students. By doing this, we hope to leverage Jacob's law which states that users prefer to use interfaces that work in a similar way to others they already know (LawsofUX.com, 2025). Therefore, previous chatbot users can transfer their existing knowledge from previous systems to our own, allowing them to intuitively understand the UI, reducing cognitive load and helping the system with its initial usability. Our design aligns with standard usability principles, including recognition over recall and predictable interaction patterns (Norman, 2013). Additionally, research shows that aesthetically pleasing interfaces are perceived as easier to use, supporting our thematic design choices (Lavie & Tractinsky, 2004).

These inspirations can be defined into specific regions, breaking the page into distinct areas. A bottom message input bar paired with an identifiable send button. A central conversation area which displays the ongoing dialogue between the AI and user. A collapsable left-hand side panel that contains the user's chat history as well as a new chat button located at the top within. The user login button is also present in the bottom left corner, just like many other websites with user account capabilities. Due to the simplicity and familiarity of this UI design, this layout ensures that navigation is predictable and supports quicker learning, as well as making sure that user attention is directed mostly towards the core interaction at the centre of the screen (Figure 3).

To reinforce the Shakespearean theme, we selected a colour palette with historical styling, including dark browns, parchment tones and a high-contrast white text for readability. Not only does this design choice enhance thematic immersion but it also aligns with the Aesthetic-usability effect, which suggests that an interface that is more visually pleasing are also perceived as easier to use. Therefore, a polished and coherent visual style should be crucial for user satisfaction and reduces the likelihood of perceived friction.

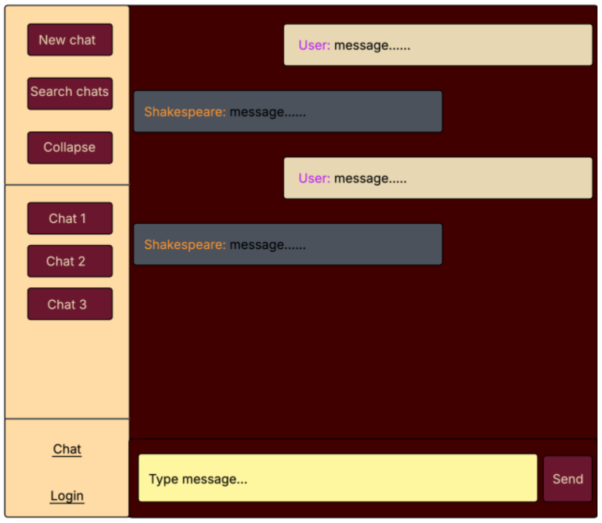


Figure 3. Chatbot interface wireframe, displaying example conversation with whole user interface and all possible buttons for user

Accessibility is a core factor in every part of our design, and to adhere to this we followed WCAG guidelines to ensure that the interface is usable by those with different kinds of disabilities. The key measures include a high contrast ratio between background and text to help with readability, including for users with colour-vision impairments. And our choice of the dark brown on white colour scheme achieves a strong positive and negative contrast to one another. We also made sure the system supports keyboard only navigation, making sure all buttons, links and input fields are accessible without the use of a mouse. ARIA labels are also applied to all interactive components so that users can interpret elements even if the visual rendering fails during runtime. Another key accessibility feature is the shape and design of our buttons, using different contrasts and to reduce the uncertainty on whether the button is interactable or not reducing the cognitive thinking time as well as supporting users with cognitive limitations. The system also supports different screen dimensions in case users want to zoom in as well as allowing access on other devices on different screen resolutions and aspect ratios. Finally, in accordance with Hick's law, we made sure to not add to many features to the UI to reduce unnecessary choices and ensuring that users can focus on and interact with the chatbot quickly without in interference. These decisions align with recognised accessibility research stating that contrast and clear visual hierarchy significantly improve usability for diverse audiences (Lidwell, Holden, & Butler, 2010)

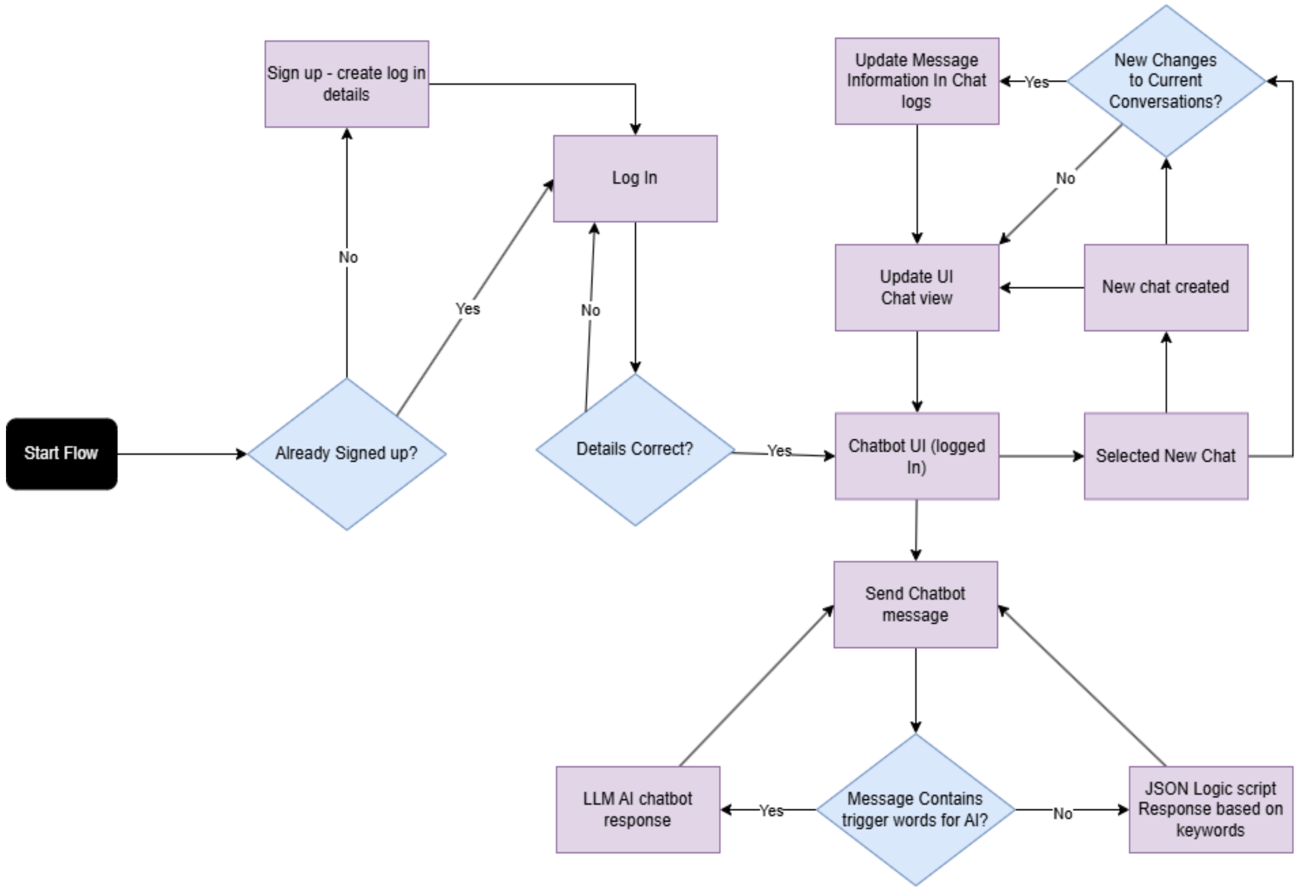

Start Flow
Already Signed up?
No
Sign up - create log in details
Yes
Log In
No
Details Correct?
Yes
Chatbot UI (logged In)
Selected New Chat
New chat created
Update UI Chat view
New Changes to Current Conversations?
Yes
Update Message Information In Chat logs
No
Send Chatbot message
Message Contains trigger words for AI?
Yes
LLM AI chatbot response
No
JSON Logic script Response based on keywords

Figure 4. User flow chart diagram, displaying the design overview of the main Chatbot interface and the decisions the user can make

IMPLEMENTATION

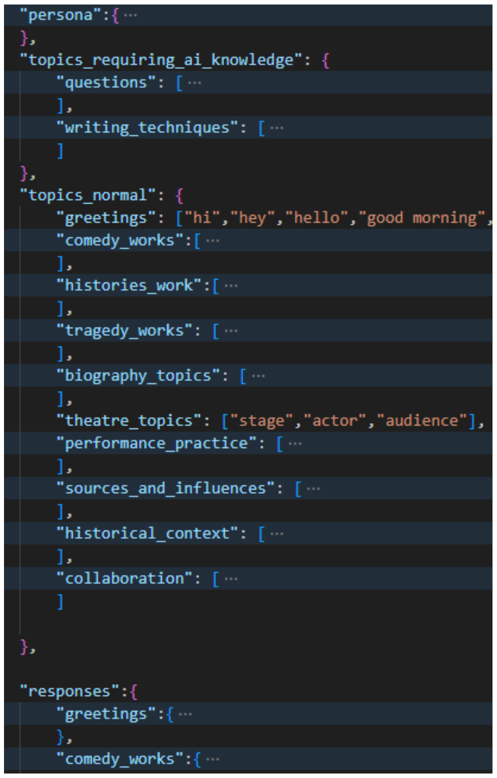
The frontend was implemented as a react application using the Vite as the build tool, styling was handled using components allowing for effective use of CSS and improving the maintainability and readability of the user interface. The front end includes dedicated pages for user authentication, the main chat interface, and error handling. The backend was developed using Node.js with an Express.js server framework, and is responsible for user authentication, chat handling, message processing and communicating with the external AI API services. Environment variables are stored in a .env file which is used specifically to manage sensitive configurations such as the API keys, ensuring security and privacy by separating development and production environments. And throughout development, coding conventions, file organisations and function naming were followed as to improve the maintainability and readability of the code.

For the main system components, the system follows an architecture consisting of a front end, backend server layer, and a JSON-based dialogue engine with an implementation of the google Gemini 2.5 API large language model. The front end handles all user interactions on the user interface, including message inputs, chat displays, authentication workflow and error handling. The front end communicates with the backend through HTTP requests, and the application only renders the application right after the backend connectivity is confirmed. The backend in our implementation essentially acts as the control layer, which manages REST endpoints for user authentication, chat session management and message processing. It also contains controllers for handling the chatbot logic, middleware for authentication and request validation, as well as service modules for external APIs. Finally, the JSON scripting engine contains a structured file called “script_pathway.json”, in which this file defines the chatbot's persona (only called when calling the LLM), and list of topic categories that hosts a list of trigger keywords. There is also a responses section which are pre-scripted for when those specific keywords are detected through the message. This allowed the chatbot to be incredibly responsive and fast without the need for AI generation for every message. These components together allow for a hybrid chatbot model, which can have pre-determined scripted responses as well as AI-generated content when appropriate.

The chatbot logic has been implemented as a hierarchal decision in the backend controller "chatController". Firstly, the user messages are transmitted from the client to the server as a structured JSON. The backend then follows deterministic checks, making sure that the message input has been normalised (converted to lower case) for fair comparisons, it then does a keyword analysis, and topic matching against the JSON scripting engine (Figure). In the JSON file, a key section is its list of words and phrases called "topics_requiring_ai_knowledge", which contains a list of questions and writing techniques, with words that suggest an open-ended or creative question is being asked, therefore the Google Gemini is used in escalation of such a complex question. The server then gets the Shakespearean persona that is also located in the "script_pathway.json" which contains the name, context, tone, traits and limitation. These are incredibly important as they ensure that the AI responds within pre-defined behavioural boundaries and therefore cannot independently define the context themselves. The server then passes through the message and the persona to the Google Gemini API, which produces a response dependent on the user's message and aligning with the Shakespearean persona. If the message does not contain any of the words listed in "topics_requiring_ai_knowledge" then it will move onto a secondary check to match even more keywords for a pre-defined response.

First, we compare the contents of the user's message to the "topics_normal" section in the JSON (Figure 5), which contains a collection of greetings, Shakespeare's works including comedies, histories and tragedies, as well as biography topics, theatre topics, performance tools, sources and influences, historical context and collaborations. All these sections have a list of keywords and phrases, and if any are matched in the user's message, then a pre-determined response found in the "responses" section of the script pathway will be returned as the full message. If multiple keywords are matched, then their responses are concatenated together to produce a full message. If the message contained no keywords, then Shakespeare would just respond saying he knew nothing of what has been said. By having this decision-making process within the express backend, the system avoids functioning as just a simple message relay, and instead maintains control over response selection, persona consistency and the overall system behaviour, thus providing server-side processing as a hybrid chatbot design.

Many tests were performed manually through methods called within the file itself. These tests included message inputs of gibberish words, mentions of technology outside of Shakespeare's lifespan, using higher/lower case letters, triggering multiple of the keywords at once. And thus, all these results provided responses that were reasonable and expected of the Shakespeare chatbot. A health-check endpoint was also implemented in the backend and called to the frontend to verify server availability before the render of the UI. This therefore handled server downtimes and connection issues. Input validation was also performed at an API level as mandatory fields of username, chat ID, and message contents are required before requests were processed. And error handling was implemented throughout the code through many try-catches, returning suitable error codes when such issues are encountered. Finally, console logging was also widely used in the code to trace errors and data flow.



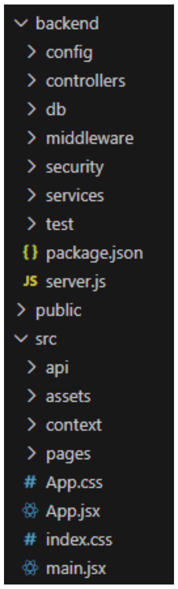
```
"persona":{ ···
},
"topics_requiring_ai_knowledge": {
    "questions": [ ···
    ],
    "writing_techniques": [ ···
    ]
},
"topics_normal": {
    "greetings": ["hi","hey","hello","good morning",
    "comedy_works":[ ···
    ],
    "histories_work":[ ···
    ],
    "tragedy_works": [ ···
    ],
    "biography_topics": [ ···
    ],
    "theatre_topics": ["stage","actor","audience"],
    "performance_practice": [ ···
    ],
    "sources_and_influences": [ ···
    ],
    "historical_context": [ ···
    ],
    "collaboration": [ ···
    ]

},

"responses":{
    "greetings":{ ···
    },
    "comedy_works":{ ···
```

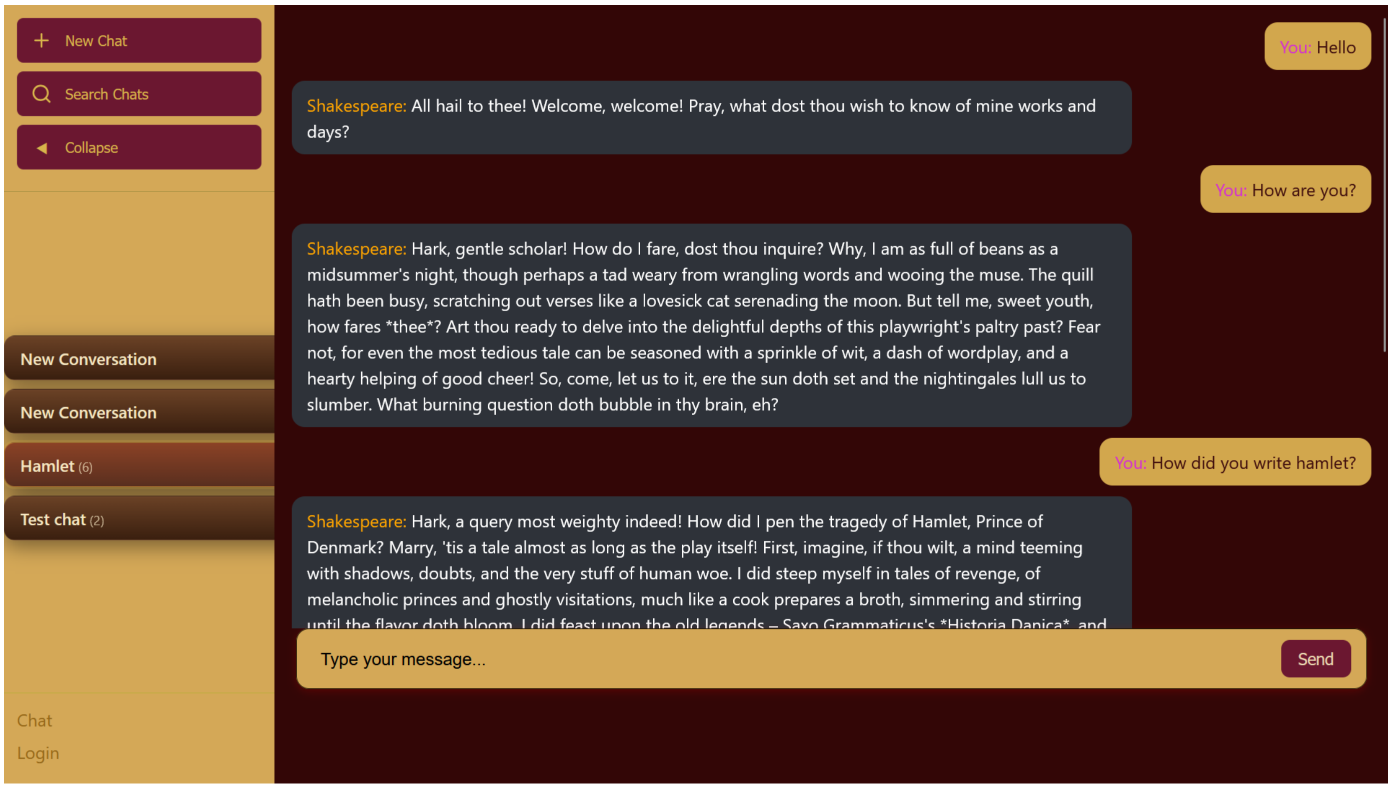
Figure 5. Code view of the JSON scripting file in folder "db" named script_pathway.json with contents collapsed to see more lines

To ensure code quality the codebase has consistent structure, and the files have modular organisation, with clear frontend, backend and middleware separation. The frontend has organisation as it las logical grouping with pages, assets, context and api. As the backend has a separation of concerns with controllers, services, security, config and database having their own section, making sure the structure is feature based and clear for others to see (Figure 6). Functions and file names are descriptive of their roles making the code easier to navigate and understand. Naming conventions are followed as such for camelCase for JavaScript and snake_case for JSON names. Sensitive data such as Google Gemini's API keys are stored and managed using environment variables and management is handled through npm. JSDoc comments as well as inline comments are scattered everywhere throughout the code, ensuring each function and section of the codebase can be readable and well defined for others to use. Finally, since we used GitHub to collaborate on this project, we used branching a lot when working on different parts of the code. These branches include a "ui-tbox" branch that only works on the frontend, and another branch called "json-script-logic", which works on specifically the JSON scripting logic in the backend. These branches were then regularly merged into main, where we would do code reviews on what has changed and if there are any issues to deal with. Overall, these different techniques and methods we used helped greatly with keeping the code quality high and making sure collaboration goes as smoothly as possible.



```
backend
  config
  controllers
  db
  middleware
  security
  services
  test
  package.json
  server.js
public
src
  api
  assets
  context
  pages
  App.css
  App.jsx
  index.css
  main.jsx
```

Figure 6. Screenshot of the project file directory, showcasing all the backend and frontend folder locations and hierarchical structure



You: Hello

Shakespeare: All hail to thee! Welcome, welcome! Pray, what dost thou wish to know of mine works and days?

You: How are you?

Shakespeare: Hark, gentle scholar! How do I fare, dost thou inquire? Why, I am as full of beans as a midsummer's night, though perhaps a tad weary from wrangling words and wooing the muse. The quill hath been busy, scratching out verses like a lovesick cat serenading the moon. But tell me, sweet youth, how fares *thee*? Art thou ready to delve into the delightful depths of this playwright's paltry past? Fear not, for even the most tedious tale can be seasoned with a sprinkle of wit, a dash of wordplay, and a hearty helping of good cheer! So, come, let us to it, ere the sun doth set and the nightingales lull us to slumber. What burning question doth bubble in thy brain, eh?

You: How did you write hamlet?

Shakespeare: Hark, a query most weighty indeed! How did I pen the tragedy of Hamlet, Prince of Denmark? Marry, 'tis a tale almost as long as the play itself! First, imagine, if thou wilt, a mind teeming with shadows, doubts, and the very stuff of human woe. I did steep myself in tales of revenge, of melancholic princes and ghostly visitations, much like a cook prepares a broth, simmering and stirring until the flavor doth bloom. I did feast upon the old legends – Saxo Grammaticus's *Historia Danica*, and

Figure 7. Screenshot of the implemented final product of the primary chatbot user interface, includes a conversation with the chatbot (Shakespeare) with pre-determined responses and AI generated responses, as well as chat options on the left side with the login button bottom left and a message bar in the bottom center of the screen with a send button next to it

PROJECT PROCESS

Throughout the development cycle our project followed a mixture of many different workflow methodologies. At the start of our project, we decided the structure for the project would use a waterfall methodology which revolves around careful planning of requirements and not returning to previous steps. However, after completing our first set of requirements we felt that as the project develops, we would get new ideas and better ways to approach things which lead us to adopt the agile methodology when starting to design and implement. This is almost the opposite of waterfall in that it revolves around returning to previous areas to modify and improve; this is because it is designed around working in short repetitive loops. This would allow us to constantly improve and develop ideas for our chatbot as skills were enhanced, and changes were made as well to the requirements. It also allowed us to backtrack and lower the priority of not as important requirements as to prevent scope creep and to focus more on main functionality. However, due to this methodology it can be chaotic with a general lack of structure, especially when backtracking and changing requirements, however this provided us with quick feedback and was very adaptable all through implementation.

To track our progress, we used a kanban board, used in GitHub where tasks were put up for allocation, and were in order of when they should have been done. This meant that each team member could assign themselves a task and manually alter the status and progress of their tasks, resulting in members doing tasks they were good at, and what they wanted to do themselves. Furthermore, the status feature proved useful as it provided an overview as to what needs to be done, what is in progress, and what is done. As for our roles, we primarily didn't have individuals just working on a specific section of the project, but instead we could just pick and choose what we wanted to do or help with in any section. This gave everyone a lot more freedom and allowed for a lot more adaptability in case of any problems. This also made sure each member was comfortable with the roles they were playing since they themselves chose to do such tasks.

To support our collaboration, we used GitHub and Microsoft Teams. This provided us with a way to communicate and a way to collaboratively work on the project together. GitHub was very useful for practical development of the chatbot itself as well as giving us the ability to collaborate. This was due to GitHub's feature of creating branches, allowing us to test new additions and functions yet not having directly affected the main source code. The pull request feature was also useful for other team members to quickly review and make changes to make sure it fits merging with the main source code, as well as GitHub having a commit log, so it was easy to see where and what changes were made in a timeline. Microsoft Teams were our other collaborative tool, which was used to communicate as well as upload ideas, changes, and documentation for all to see and respond to.

Our team worked well together, each preferring yet working on various parts of the process (frontend, backend, documentation and testing). With this we would meet in person every week to discuss ideas and progress more formally. And, on Teams we were a lot more active with messages, updates, and ideas ensuring that anything that has happened to the team is notified. As for the challenges we faced, the most glaring one being that one of our teammates lacked fundamental knowledge for the required roles given to them. This resulted in the project (especially in coding) being only worked on by three people instead of four, which created challenges when it came to time allocation and completing the requirements. This directly caused challenges in the decision making of our project scope, and so with many requirements still left to do we had to sacrifice some not as important features to focus on making a good fundamental application without as many features as we previously listed. To resolve this issue, we therefore allocated the member to focus more on the design and reporting side of the project, rather than working on the implementation side.

REFLECTION

Throughout the course of making our project communication was very important and quite frequent with constant updates as to what people were doing and what had been done, ensuring everyone was clear as to the progress that we had made. The division of work was mostly split evenly between members; however not everyone undertook an equal share of different parts of the project. For example, frontend CSS and JSX features were mostly equally divided, but other parts of the project some people took on more backend or documentation than others, based on skill and keenness to do the tasks.

AI tools were often successfully utilized throughout development to complement parts of the project, such as the use of the copilot extension in visual studio code for assistance in debugging JavaScript and CSS, providing explanations of unfamiliar syntax and logic, suggesting improvements, adding comments for JSDoc coding practices, and used as well in refining our already predetermined requirements, acting as an assistant to point out problems. While the AI did not do everything for us, as we still did our own problem solving, it was indeed very useful in speeding up the processes of implementing the features and design, helping us not get stuck on small bugs and errors. And thus, we were able to complete most of the requirements which suffered from scope creep due to AI's helpful assistance.

As for challenges we faced, we suffered some time management issues because of scope creep and using agile methodology, and so due to its lack of structure as well as having a member who lacked foundational knowledge of the course, this resulted in some requirements taking longer than anticipated, such as UI design and choices as well as challenges of having to learn JavaScript and CSS for the first time for some of us which slowed the starting point of the project. We also had to change our wireframes and designs of our UI sometimes due to newly taught learning materials such as having accessibility in our design and following the WCAG standards. However, we were able to overcome most of these problems due to the same use of the agile methodology. This was because using agile allowed us to backtrack, change previous requirements and design because of increased team coordination and communication, as well as ensuring everyone is doing what they would prefer or what they are good at. But of course, even though we were able to complete most of our important requirements, due to the time constraints, this left many to still be incomplete.

In future projects, there are several things that could be done differently. Firstly, due to firsthand experience, we now have more of a realistic understanding of the time required for individual tasks to be completed. As a result, future projects are less likely to result in poor time management and tasks needing to be extended or rushed. Additionally, in the future greater emphasis will be placed on producing clearer and more detailed wireframes and design specifications at an earlier stage. As in this project several of our designs' reasoning were made after creating them rather than before or during them. This led to multiple reworks of our design, leading to massive losses of time as we also had scope creep when we introduced accounts and passwords as a requirement. Therefore, thinking of our design choices before creating would have prevented inconsistencies earlier on in the framework, making sure future projects could minimize reworks, reduce ambiguity, and ensure a quicker development cycle. Another key area of improvement is the design process of accessibility because we didn't learn about user accessibility until later into the course, it therefore wasn't considered until after the creation of our first design; this resulted in having to revise things to fit accessibility considerations (Johnson, 2020). In future projects, all aspects of software creation will be considered at once throughout the design process and implementation. For the actual code itself, future improvements would include having an automated test for chatbot logic for even more accurate responses, and having a clearer error handling strategy, since all error handling is scattered across many files.

Collectively as a group from this software engineering project, the team has developed a multitude of skills. Firstly, when it comes to the use of programming, we all have at least a basic understanding of JavaScript, CSS, and Express meaning that everyone should be capable and knowledgeable enough to understand and build parts of the frontend and backend. We have also learned the importance of communication, adaptability, thoughtful design and helpful use of AI prompt engineering. These lessons will be invaluable to us in the future, to speed up processes in a professional development environment where teamwork and problem solving are essential. Using an iterative development approach, the team gained a practical understanding of the software development lifecycle, which includes requirement analysis, design, implementation, testing and refinement, while also retaining software quality, accessibility standards, and ethical consideration. Security and usability were also key considerations, particularly in password handling, accessibility-focused UI design, and the ethical use of AI-generated content being used as a tool to help students, and those who need it.

REFERENCES

- Ace-7854(2025). Historics [Source code]. GitHub - https://github.com/Ace-7854/historics/
- World Wide Web Consortium (W3C). (2024). Web Content Accessibility Guidelines (WCAG) 2.2. https://www.w3.org/TR/WCAG22/
- JSDoc.org. (n.d.). JSDoc — Documentation for JavaScript. https://jsdoc.app/
- Google. (2023). *Google Gemini: Large language model API*. https://ai.google.dev/models/gemini
- Norman, D. A. (2013). *The design of everyday things* (Revised ed.). Basic Books.
- Lavie, T., & Tractinsky, N. (2004). Assessing dimensions of perceived visual aesthetics of web sites. *International Journal of Human–Computer Studies, 60*(3), 269–298. https://doi.org/10.1016/j.ijhcs.2003.09.002
- Lidwell, W., Holden, K., & Butler, J. (2010). *Universal principles of design* (2nd ed.). Rockport Publishers. https://www.rockpub.com/product/universal-principles-of-design-2nd-edition/
- Johnson, J. (2020). *Designing with the mind in mind: Simple guide to understanding user interface design guidelines.* Morgan Kaufmann https://www.elsevier.com/books/designing-with-the-mind-in-mind/johnson/978-0-12-817651-1
- European Union. (2016). *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation). Official Journal of the European Union*. https://eur-lex.europa.eu/eli/reg/2016/679/oj
- LawsofUX.com. (2025). *Jakob's Law*. https://lawsofux.com/jakobs-law/
- Meta Platforms, Inc. (2023). React: A JavaScript library for building user interfaces (Version 19.1.1). https://react.dev/
- Vite. (2024). Vite: Next generation frontend tooling (Version 7.2.4). https://vitejs.dev/
- React Router. (2024). React Router: Declarative routing for React (Version 7.9.5). https://reactrouter.com/
- styled-components. (2024). Visual primitives for the component age (Version 6.1.19). https://styled-components.com/
- Node.js Foundation. (2024). Node.js: JavaScript runtime built on Chrome's V8 JavaScript engine. https://nodejs.org/
- Express.js. (2024). Express: Fast, unopinionated, minimalist web framework for Node.js (Version 5.1.0). https://expressjs.com/
- Ranieri, G. (2024). node-argon2: The Node.js Argon2 library (Version 0.44.0). https://github.com/ranisalt/node-argon2
- Mozilla Developer Network. (2024). HTTP | MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/HTTP
- Lucide. (2024). Lucide: Beautiful & consistent icon toolkit (Version 0.552.0). https://lucide.dev/
- Nielsen Norman Group. (1994). 10 Usability Heuristics for User Interface Design. https://www.nngroup.com/articles/ten-usability-heuristics/
- Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S., Elmqvist, N., & Diakopoulos, N. (2016). Designing the user interface: Strategies for effective human-computer interaction (6th ed.). Pearson. https://www.pearson.com/en-gb/subject-catalog/p/designing-the-user-interface-strategies-for-effective-human-computer-interaction-global-edition/P200000005412/9781292153926
- Loader (2023) https://uiverse.io/ahmedkhaled0/red-ghost-loader