Smart Meeting Summarizer (SMS)

A Web-Based Transcript Summarization Tool
Using NLP and Web Technologies to Automate and Simplify Meeting Notes

Prepared by: Aloys Osheen Alfred

Contact Information
Email: aloysalfred.bca22@doonbuinessschool.com

Submission Date
May 22nd 2025
Purpose :
As part of the evaluation process for the WebDeveloper position at THINK ACADEMICS .

Contents

Introduction

- Smart Meeting Summarizer (SMS) is an advanced AI-powered web application designed to transform lengthy meeting transcripts into concise, easy to understand summaries.
- In today's fast-paced environment, professionals and students often face challenges in digesting extensive meeting notes or recordings, leading to information overload and missed details
- SMS tackles this problem by leveraging natural language processing (NLP) to convert hours of discussion into clear, actionable bullet-point summaries within seconds.
- The intuitive web interface allows users to simply paste their meeting transcript and instantly receive a summary highlighting:
    - Key discussion points
    - Important decisions
    - Speaker Names
- This process not only saves significant time but also ensures that no critical information is missed or overlooked
- The rising demand for AI-driven meeting assistants underscores the need for efficient, automated solutions like SMS, which demonstrate immense time-saving and productivity benefits.
- By positioning itself as a productivity partner, SMS enables users to delegate the burdensome task of note-taking and summarization to AI, allowing them to focus on meaningful participation and decision-making during meetings.
- Ultimately, SMS empowers teams, professionals, and students to stay organized, informed, and efficient—turning every meeting into a source of clear, actionable insights.

Problem Statement

In modern workplaces and educational settings, meetings are a constant often resulting in lengthy, repetitive transcripts or exhaustive minutes that are time-consuming and inefficient to review. Critical ideas, decisions, and action items frequently become buried within pages of dialogue, making it difficult for busy professionals and students to extract what truly matters. As highlighted by industry experts, reading raw meeting transcripts can be a dreadful, disorganized, and repetitive experience. With back-to-back meetings now the norm, there's rarely enough time to manually distill each conversation into clear, actionable points. This inefficiency not only risks missed information and miscommunication but also threatens overall productivity and team alignment. There is a pressing need for a solution that delivers concise, accurate, and easy meeting summaries empowering stakeholders to quickly recall key outcomes and next steps without wading through entire transcripts or relying solely on memory.

SMS took the following as problems to rectify :

- Lengthy Transcripts: Meetings often generate word-for-word transcripts or detailed minutes that are overwhelming and impractical to read in full.
- Buried Information: Important insights, decisions, and action items can easily become lost within pages of unstructured conversation.
- Time Constraints: With packed schedules and frequent meetings, individuals rarely have the opportunity to manually summarize or review every discussion.
- Cognitive Overload: Sifting through disorganized, repetitive transcripts can be mentally exhausting and leads to disengagement.
- Risk of Miscommunication: Inefficient documentation increases the likelihood of missed details, poorly informed decisions, and team misalignment.
- Need for Efficiency: Organizations and individuals require a faster, smarter way to capture and recall meeting essentials without sacrificing accuracy or clarity.
- Summary Quality: The challenge lies in producing summaries that are both precise and easy to digest, ensuring stakeholders can quickly grasp decisions, tasks, and highlights from any meeting.

Proposed Solution

The SMS (Smart Meeting Summarizer) solution offers a web-based solution that utilizes intelligent approaches to create concise abstractions of lengthy meeting transcripts.

- SMS utilizes lightweight Python- and Flask-based natural language processing techniques to extract meaningful content from meeting transcripts without the performance demands of heavyweight models.
- Users can upload their meeting transcripts into the platform in order to receive well-organized bullet-point summaries of key information coupled with speaker intentions and recommended actions without needing to read the entire conversation.
- The interface of SMS offers an HTML and JavaScript-based platform that utilizes CSS for latest user experience features such as responsive layouts and dark and light themes and file export capabilities for .pdf and .txt formats.
- SMS offers automated note-taking capability which reduces the time professionals and teachers require to view discussions for remaining in sync with their deliverables.
- The platform combines real-time feedback operations with simple user interface elements and effective user interaction patterns which support different technical capabilities.
- When SMS combines complicated conversations into easily readable summaries it lessens mental fatigue as well as enhances information retention and ensures critical points stay in view.
- Through its smart processing abilities the platform transforms messy transcripts to readable output which results in increased productivity along with better decision making and new meeting documentation habits for modern teams.

Objectives

- Several main goals served as the foundation for the Smart Meeting Summarizer project:

- **Automate Meeting Note Summarization**:
- Create a system that can automatically distill long meeting transcripts into a comprehensible summary that precisely captures all significant decisions and points.

- **User-Friendly Frontend Interface**:
- Design a sophisticated and user-friendly web interface that makes it simple for users to paste text input and get results. The user interface should be clear, simple to use.

- **Accessibility and Responsive Design:**
- Put into practice a responsive frontend design that functions flawlessly on mobile, tablet, and desktop computers. To show that you care about accessibility and user experience, make sure features like dark mode are available for user comfort.

- **AI Backend Integration:**
- Connect the frontend to a robust NLP based flask backend. The goal is to demonstrate full-stack development abilities by seamlessly integrating a web interface with the AI model's functionality.

- **Productivity-Boosting Interactive Features:**
- Offer features that increase user productivity, like the ability to download the summary as a text file or PDF, copy it to the clipboard instantly, and show useful info (the summary's word count and reading time).

- **Scalability and Monetization Planning:**
- To lay the foundation for future monetization or expanded features, design the system architecture to support future scaling (e.g., handling higher load or larger transcripts) and integrate a basic model of user plans (Free vs. Pro) in the user interface.

System Architecture

The Smart Meeting Summarizer (SMS) utilizes a modular architecture which delivers separation between user interface elements and backend components. The architecture delivers a high-quality experience to end-users across all devices while delivering server-side natural language processing efficiency.

- **Frontend**
- The implementation includes HTML5 with CSS3 and JavaScript while the CSS3 structure incorporates base.css along with component1.css and responsive.css.
- The interface consists of a responsive multi-section web application which displays content in a clear visual hierarchy:
- Navigation bar displaying SMS branding alongside dark and light theme switch and section links for Plan, About, Contact Us
- The main hero section contains a tagline and the primary interface elements
- The Summarizer input section has text area for receiving input and summary display for showing output
- Users can export summaries through copy to clipboard and download options for .txt and .pdf files
- The system uses cards to present different types of information to users

- **Backend**
- Framework: Python Flask
- The main page gets rendered through the endpoint / which supports both GET and POST requests.
- Users can test the backend functionality by accessing the /ping route.
- The transcript submission process uses POST requests which are sent through the form to the / route.
- The summary_engine.py defines a centralized function that receives the transcript for summarization.
- The engine depends on to create abstractive summaries.
- The final summary output gets sent to the frontend through the HTML template for display.

- **Integration**

- The frontend and backend were integrated using Flask's templating engine. When a user submits a transcript via the HTML form, the data is sent to the backend using a POST request. The Flask server processes the input through the BART summarization model and returns the generated summary. This summary is dynamically rendered back into the same HTML page using Jinja2, enabling a seamless, responsive user experience.

- **Error Handling:**
- The submission holds a try except block so that the users see a message when the summary cant be loaded and not a site crash.

Technology Stack

Smart Meeting Summarizer combines modern front-end and back-end technologies:

- **Frontend:**

- **HTML5**: Structures the web interface, organizing sections like Home, About, Plans, and Contact.
- **CSS3**: Handles styling and responsiveness, including custom CSS variables for easy theme switching (e.g., dark mode). Features like Flexbox layouts and hover effects create a polished and responsive look.
- **JavaScript** : Adds interactivity, such as form handling, auto-resizing text areas, dark mode toggles, and export functionality using libraries like jsPDF for PDF generation.

- **Backend:**

- **Python & Flask:** Manages server-side logic, routing, and templating.
- **Hugging Face Transformers**: Powers the NLP engine with BART Large CNN for summarization.

- **Additional Tools:**

- **Font Awesome:** For icons.
- **Google Fonts:** For clean typography.
- **Git**: For version control during development.

This stack balances front-end proficiency with back-end integration, showcasing full-stack development skills.

Key Features

- **AI-Powered Summarization:**
- The system generates brief and organized summaries through advanced natural language processing without manual supervision of long meeting records.

- **Real-Time Performance:**
- The summarizer operates in real-time to complete input tasks quickly and presents users with visual feedback indicators through loaders for better processing awareness.

- **Intuitive Design:**
- The interface features a minimalist design that includes spacious text boxes and clear operational buttons to maximize the user's experience while presenting essential data elements including word quantification and reading duration predictions.

- **Copy & Export Options:**
- The system enables users to perform two operations: they can copy summaries to their clipboard and also export the summaries to downloadable .txt or .pdf files which can be saved for later use.

- **Dark Mode:**
- The system lets users choose between dark and light display modes to optimize visual clarity based on different lighting conditions.

- **Responsive Design:**
- The website functions flawlessly on any device type including desktop computers and tablets and smartphones because of its adaptive design and CSS responsiveness.

- **Additional Sections**:
- The website structure contains distinct sections for About, Plans, Reviews, and Contact which combine to form a professional appearance while creating better user connections.

- **Error Handling:**
- The system displays user-friendly messages when it detects invalid input to prevent crash.

Testing and Evaluation

The Smart Meeting Summarizer (SMS) application underwent rigorous testing across various functional and non-functional dimensions to ensure reliability, responsiveness, and accuracy.

- **Test Cases Covered**

- The summarizer was tested with long transcripts (500+ words), and the output was generated in under 10 seconds on a local machine.
- Submissions with empty input were gracefully handled with a user-friendly error message indicating that input is required.
- Input fields were tested with random filler text, emojis, speaker labels (e.g., "Aloys:"), and special characters all of which were processed correctly or safely ignored by the model.
- The "Summarize" button was clicked repeatedly to test if concurrent requests were blocked, which worked as intended the button was disabled until results were returned.
- Additional features such as the copy, export to .txt, and word count display were verified for correct functionality.
- Responsiveness was tested across various device dimensions, including desktop, tablet, and mobile views, using Chrome DevTools.
- The scroll navigation to About, Plans, and Contact sections was also tested and worked smoothly.

- **Frontend/UI Testing**

- The interface was tested on Chrome, Edge, and Firefox to ensure consistent rendering.
- Dark Mode toggle functioned correctly across all views and devices.
- Input field scaled appropriately for large pasted content, and meta details like reading time and word count updated dynamically.
- Loading states were shown while the backend was processing, with smooth transitions and clear feedback.
- Error and Exception Handling
- When the backend was unreachable or failed to return a response, the frontend displayed a fallback message to prevent user confusion.
- Submissions with unreadable or excessive special characters didn't crash the application.
- Model loading failures and backend exceptions were tested by deliberately simulating timeout and null output scenarios.
- All errors were handled gracefully without crashing the UI, maintaining a seamless user experience.

- **Deployment Observations**

- The app was deployed and tested across Render, Railway, and Choreo platforms.
- Render built the app successfully, but gave a 502 error due to memory limitations when loading the transformer model.
- Railway failed due to the presence of pywin32, a Windows-only dependency that isn't compatible with Linux-based deployment environments.
- Choreo ran into disk quota issues, primarily because the transformer model size exceeds free-tier constraints.
- Attempts were made to optimize requirements, reduce model load, and explore containerized deployment, but the infrastructure remained the bottleneck.
- Despite these challenges, the entire application runs smoothly on local Flask server, and all functionality has been demonstrated and screen-recorded as proof of execution.

USP of SMS

- **Tailored for Meetings:**
- Meeting-style transcripts receive specialized training and structuring to maintain flow alongside speaker significance and decision point preservation beyond what general tools can deliver.

- **Accessible and Simple:**
- No registration, login, or setup required just paste your transcript and click to summarize in a distraction-free environment.

- **Polished Design:**
- A modern user interface combines with thoughtful design elements to present superior user experience including advanced functions such as dark mode and export tools.

- **Speed:**
- The tool provides instant summary generation which proves beneficial for professionals attending consecutive meetings that require brief recaps.

- **Free with Growth Potential:**
- Presently accessible at no cost to every user while future plans may establish paid services which will be available through Free vs Pro subscription options.

- **Human-Like Quality:**
- The summaries presented have clear readability and maintain an authentic voice which ensures immediate usability and trustworthiness.

Smart Meeting Summarizer WebApp Screenshots

Figure 1: Home Page (Light Mode)

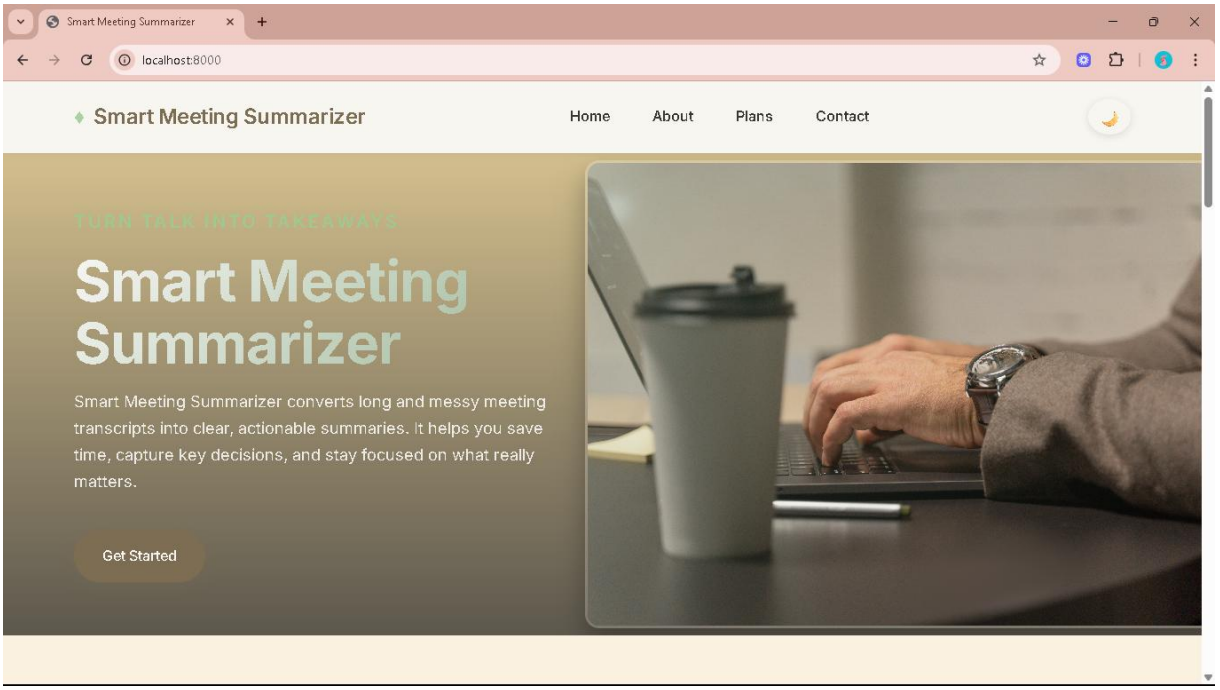


Figure 2: Transcript input Display Displays sample input

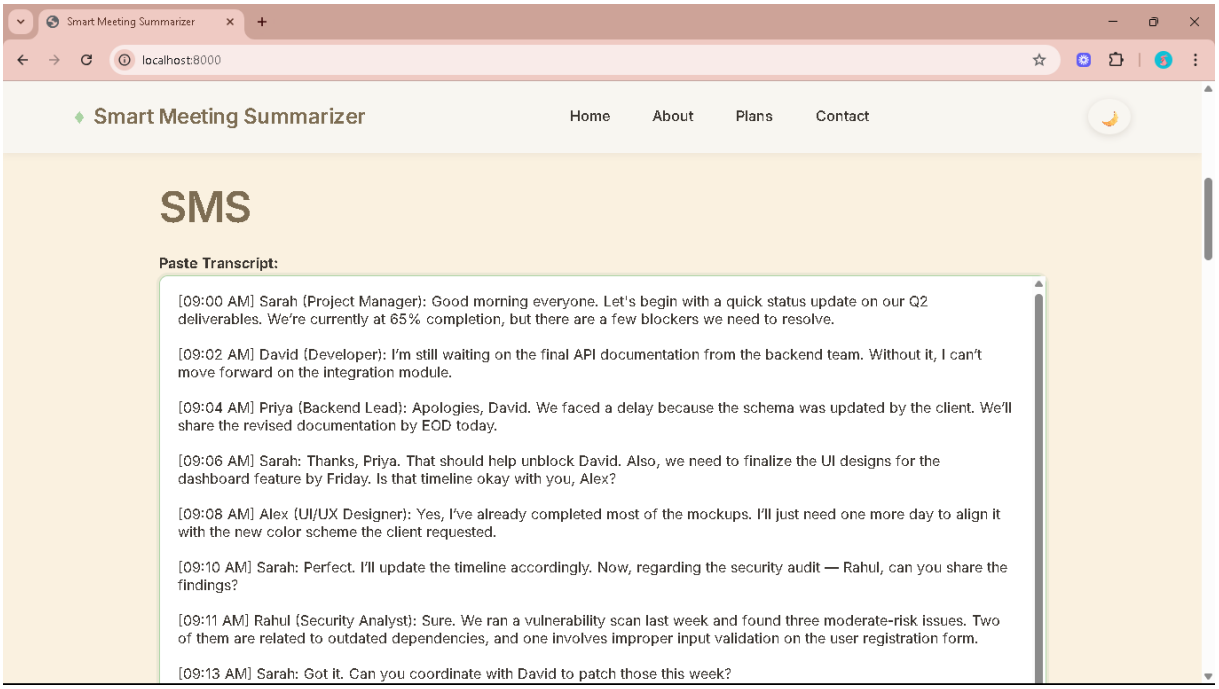


Figure 3: Summarized output Illustrates summary with bullt points and speaker names along with summary info. and other options .

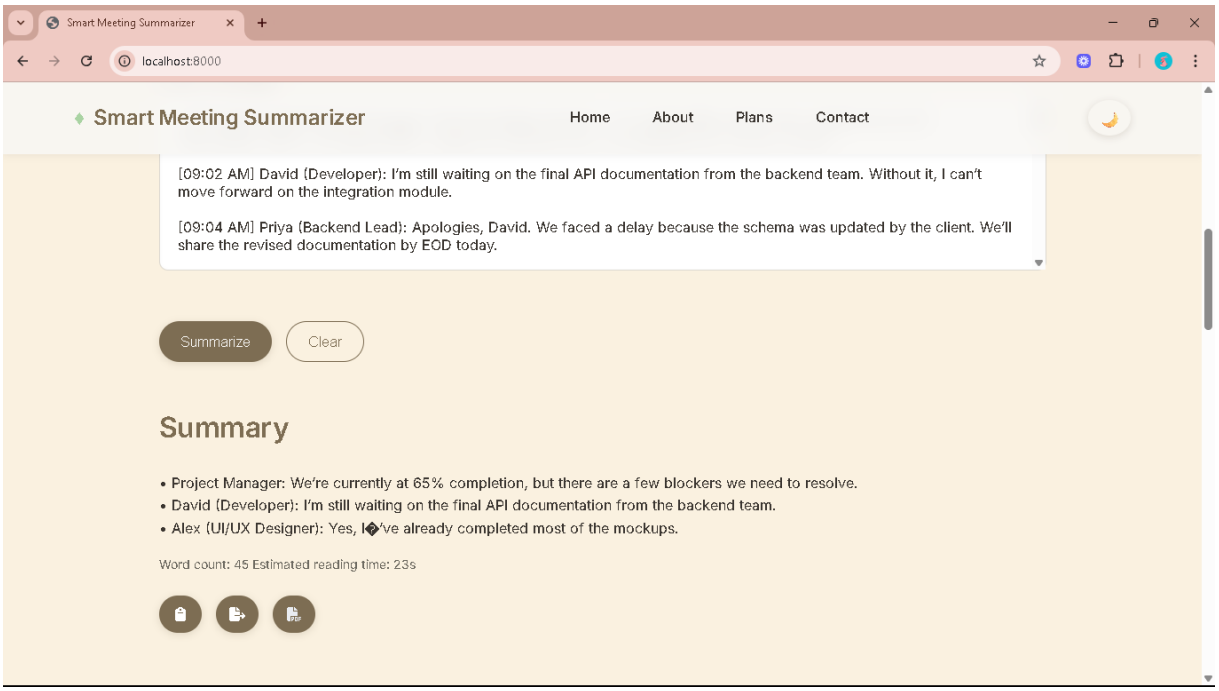
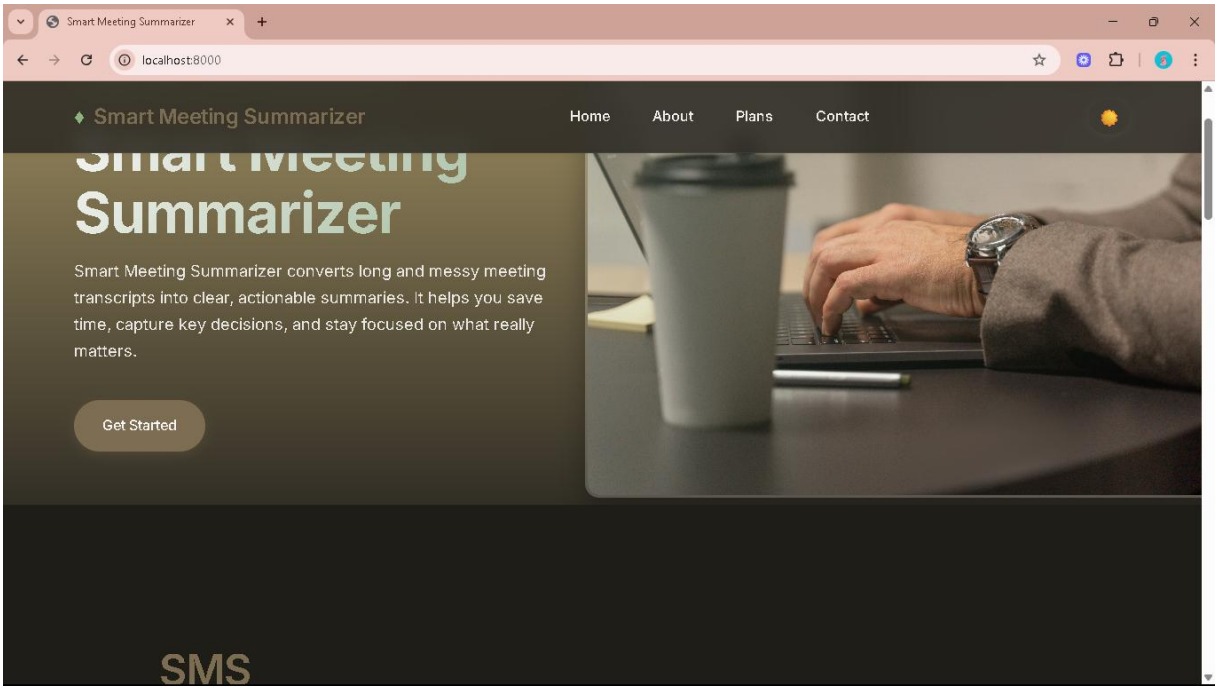


Figure 4: Dark Mode Activation Illustrates theme toggle and dark mode appearance.



These visuals will provide clear evidence of SMSs design and usability.

Deployment Summary

- **Challenges Faced**

Initially, I attempted to deploy my Smart Meeting Summarizer app using various platforms like Render, Choreo, Railway, and Hugging Face Spaces. However, due to memory limitations and large model size (BART), every deployment attempt failed with storage or server errors. Payment barriers for higher tiers also restricted access because on platforms like railway I tried to purchase the plan so that I can upload my site with heavy NLP model but because of international payment criteria it kept on declining my payment, making direct hosting infeasible despite multiple optimization efforts.

- **Final Deployment Strategy**

To overcome hosting constraints while still demonstrating the full application workflow, I opted for a hybrid deployment approach using Netlify and YouTube. The website frontend (HTML, CSS, JS) was deployed using Netlify's free static site hosting with HTTPS support. Since the summarizer backend (NLP model) couldn't be hosted freely due to size, I recorded a walkthrough of the live app running locally and uploaded the video to YouTube. The video was then embedded into the website to simulate complete functionality.

Final Deployment Highlights:

- Static frontend hosted with Netlify (free, HTTPS-enabled)
- Screen recording created from localhost
- Video uploaded as Unlisted YouTube video
- Embedded in site using <iframe> for seamless integration
- Demonstrates full UI/UX and working summarizer engine

GitHub Repository & Version Control

This project, Smart Meeting Summarizer, is version-controlled using Git and publicly hosted on GitHub for full transparency, collaboration, and reproducibility.

GitHub Repository Link:

https://github.com/aloys3125/SMS.git

- **Version Control Practices**

- Git was used extensively to track incremental changes throughout the project development cycle.
- After each major feature addition, bug fix, or UI enhancement, the changes were staged using git add, committed with a descriptive message using git commit, and pushed to the remote repository via git push.
- This structured workflow helped maintain a clear development history and enabled seamless debugging or rollbacks if needed.
- All code, assets, models, and required dependency files (e.g., requirements.txt, app.py, summary_engine.py, etc.) were versioned and regularly updated.
- ReadMe and documentation files were maintained for clarity and onboarding ease.

- **Repository Contents**

- Backend: Python files including the Flask app (app.py) and the NLP engine (summary_engine.py).
- Frontend: HTML, CSS (split into modular files), and JavaScript.
- Static and Templates: Organized directory structure following Flask conventions.
- Documentation file SMS_Documentation.pdf
- Supporting Files: requirements.txt to install all necessary dependencies.

Future Enhancements

While SMS is already functional, we have exciting plans for growth:

- **Real-Time Integration:**
- Users can instantly receive meeting summaries through direct connections with Zoom, Google Meet and Teams.

- **Audio Support :**
- The software should support voice recordings through transcription conversion and summarization to eliminate the need for manual transcription.

- **Longer Transcripts :**
- The system utilizes transcript segmentation together with multiple summary generation steps to handle long meeting durations that exceed memory limits.

- **User Accounts:**
- A secure user login system should be implemented to enable people to save their summaries through cloud storage.

- **Collaboration :**
- Teams can work together more effectively through summary sharing which enables team members to perform joint editing and annotations as well as review functions.

- **Customization :**
- Users should be able to determine the length and structure and tone of their summaries based on their specific requirements as well as organizational needs.

- **Multi-Language Support :**
- The system should be able to process different languages in addition to English so it can better serve its global audience and various multilingual teams.

- **Scalability :**
- The system must upgrade its cloud-based hosting infrastructure through autoscaling and load balancing alongside deployment for improved handling of large traffic and enterprise requirements.

Summary
from start to end

The development of Smart Meeting Summarizer (SMS) began as a practical solution for handling extensive meeting documentation in a more streamlined manner. The project demonstrates a well-organized software development process by combining complete web development with natural language processing and user interface/user experience design.
The project kicked off by establishing the core issue: lengthy meeting transcripts contain disorganized information that leads to missing essential elements. The objective focused on constructing an automatic summary system which relies on NLP technology to generate clean web-based summaries.

- The application was constructed through the following technologies:

- **Python & Flask (Backend Logic):**
  The backend is built using Python and Flask, enabling clean routing, form handling, and integration with the NLP engine. Flask was chosen for its simplicity and flexibility, allowing the app to process user-submitted transcripts, manage sessions, and return real-time summaries efficiently within a lightweight web server setup.
- **Hugging Face Transformers – BART Model (NLP Engine):**
  For summarization, the project utilizes the BART-Large CNN model from Hugging Face Transformers. This state-of-the-art model delivers high-quality abstractive summaries by understanding sentence semantics. It processes long transcripts and distills key points, ensuring the summaries are concise, readable, and context-aware, which is crucial for meeting transcripts.
- **HTML5, CSS3, JavaScript (Frontend):**
  The user interface is built with HTML5, CSS3, and JavaScript, making it responsive and interactive across devices. Features like dark mode, dynamic word count, export options, and smooth transitions enhance usability. JavaScript handles client-side interactivity while CSS ensures consistent layout styling, offering users a polished and intuitive experience.
- The entire project received version control treatment from Git as the developers maintained it on a public GitHub repository through frequent updates.
- Despite successful local development and testing, deploying the project on platforms like Render, Railway, and Choreo was hindered by the memory-intensive NLP model causing repeated failures.
- However, the project showcases strong technical execution using Flask, BART transformers, and a responsive frontend. A hosted walkthrough site and GitHub repo were created to demonstrate functionality and commitment.
- An alternative hosted project site was established to demonstrate the entire work through both screen recording of local hosted WebApp and documentation access while keeping the source code and NLP engine on GitHub available for inspection and to demonstrate functionality and commitment.
- The deployment obstacles in this experience showcase the value of technical expertise alongside resilience and the ability to adjust to situation-specific challenges and the determination to finish the assignment. The project demonstrates complete modern software development practices and problem-solving applications through its backend logic and model integration together with frontend design elements and testing and deployment implementations.

Resources Used

The following official libraries, frameworks, and deployment services were used in the development of the Smart Meeting Summarizer project:

- **Python Libraries & Frameworks**
  Flask (Web Framework)
- https://flask.palletsprojects.com/

- nltk (Natural Language Toolkit for preprocessing and tokenization)
- https://www.nltk.org/

- Transformers by Hugging Face (for BART model implementation)
- https://huggingface.co/docs/transformers/index

- PyTorch (for deep learning model support used with transformers)
- https://pytorch.org/

- **Frontend Technologies**
- HTML5 Documentation
- https://developer.mozilla.org/en-US/docs/Web/HTML

- CSS3 Documentation
- https://developer.mozilla.org/en-US/docs/Web/CSS

- JavaScript Documentation
- https://developer.mozilla.org/en-US/docs/Web/JavaScript

- Google Fonts (for typography used in the UI)
- https://fonts.google.com/

- Font Awesome (for icon usage)
- https://fontawesome.com/
-
- **Testing Tools & Debugging**
- Chrome Developer Tools (for responsive and UI testing)
- https://developer.chrome.com/docs/devtools/

- **Deployment Platforms (Attempted)**
- Render
- https://render.com/

- Railway
- https://railway.app/

- Netlify
- https://www.netlify.com/

- **GitHub (for version control and public repository hosting)**
- https://github.com/