

School of Computer Science and Engineering

(Computer Science & Engineering) Faculty of Engineering & Technology

Jain Global Campus, Kanakapura Taluk - 562112 Ramanagara District, Karnataka, India

#### 2024-2025

**( VI Semester)**

**A Project Report on**

## “Document Flashcard Generator”

**Submitted in partial fulfilment for the award of the degree of**

**Bachelor of Technology in**

### COMPUTER SCIENCE AND ENGINEERING

**Submitted by**

**Mohamed Faiz V S (22BTRCN173),**

**Karthik Jadhav (22BTRCN135),**

**K S Sirisha (22BTRCN129),**

**Shashank.S (22BTRCN257),**

**Y Chandana (22BTRCN330)**

**Under the guidance of**

**Dr. Guruvammal. S**

Assistant Professor

Department of Computer Science and Engineering

School of Computer Science & Engineering

Faculty of Engineering & Technology

JAIN (Deemed to-be University)



Department of Computer Science and Engineering

School of Computer Science & Engineering Faculty of Engineering & Technology

Jain Global Campus, Kanakapura Taluk - 562112 Ramanagara District, Karnataka, India

# CERTIFICATE

This is to certify that the project work titled **“Document Flashcard Generator”** is carried out by **Mohamed Faiz V S (22BTRCN173), Karthik Jadhav (22BTRCN135), K S Sirisha (22BTRCN129), Shashank.S (22BTRCN257), Y Chandana (22BTRCN330),** a bonafide student(s) of Bachelor of Technology at the School of Engineering & Technology, Faculty of Engineering & Technology, JAIN (Deemed-to-be University), Bangalore in partial fulfillment for the award of degree in Bachelor of Technolgy in Computer Science and Engineering, during the year **2024-2025**.

|  |  |  |
| --- | --- | --- |
| **Dr. Guruvammal. S** | **Dr. Mahesh TR ,** | **Dr. Geetha G** |
| Asst/Asso/ Professor Dept. of CS&E,  Date: | Program Head,  School of Computer Science & Engineering  Faculty of Engineering & Technology  JAIN (Deemed to-be University) Date: | Director,  School of Computer Science & Engineering  Faculty of Engineering & Technology  JAIN (Deemed to-be University)  Date: |

Name of the Examiner Signature of Examiner

2.

# DECLARATION

We **Mohamed Faiz V S (22BTRCN173), Karthik Jadhav (22BTRCN135), K S Sirisha (22BTRCN129), Shashank.S (22BTRCN257), Y Chandana (22BTRCN330),**student of 6th semester B.Tech in **Computer Science and Engineering**, at School of Engineering & Technology, Faculty of Engineering & Technology, **JAIN (Deemed to- be University)**, hereby declare that the internship work titled **“Document Flashcard Generator”** has been carried out by us and submitted in partial fulfilment for the award of degree in **Bachelor of Technology in Computer Science and Engineering** during the academic year **2024-2025**. Further, the matter presented in the work has not been submitted previously by anybody for the award of any degree or any diploma to any other University, to the best of our knowledge and faith.

|  |  |
| --- | --- |
| Name1:  USN : | Signature |
| Name2:  USN : | Signature |
| Name3: USN :  Name4:  USN :  Name5  USN : | Signature  Signature    Signature |
| Place : Bangalore Date : |  |

**ACKNOWLEDGEMENT**

*It is a great pleasure for me to acknowledge the assistance and support of a large number of individuals who have been responsible for the successful completion of this project work.*

*First, we take this opportunity to express my sincere gratitude to Faculty of Engineering & Technology, JAIN (Deemed to-be University) for providing me with a great opportunity to pursue my Bachelors / Master’s Degree in this institution.*

*we are deeply thankful to several individuals whose invaluable contributions have made this project a reality. we wish to extend my heartfelt gratitude to* ***Dr. Chandraj Roy Chand, Chancellor****, for his tireless commitment to fostering excellence in teaching and research at Jain (Deemed-to-be-University). we are also profoundly grateful to the honorable* ***Vice Chancellor, Dr. Raj Singh, and Dr. Dinesh Nilkant, Pro Vice Chancellor****, for their unwavering support. Furthermore, we would like to express my sincere thanks to* ***Dr. Jitendra Kumar Mishra, Registrar****, whose guidance has imparted invaluable qualities and skills that will serve us well in our future endeavors.*

*we extend my sincere gratitude to* ***Dr. Hariprasad S A, Director*** *of the Faculty of Engineering & Technology,* ***and Dr. Geetha G, Director*** *of the School of Computer Science & Engineering within the Faculty of Engineering & Technology, for their constant encouragement and expert advice. Additionally,we would like to express my appreciation to* ***Dr. Krishnan Batri, Deputy Director (Course and Delivery), and Dr. Deepak K Sinha, Deputy Director (Students & Industry Relations),*** *for their invaluable contributions and support throughout this project.*

*It is a matter of immense pleasure to express my sincere thanks to* ***Dr. MAHESH T R, Program Head, Computer Science and Engineering****, School of Computer Science & Engineering Faculty of Engineering & Technology for providing right academic guidance that made my task possible.*

*I would like to thank our guide DR.GURUVAMMAL.S , Associate /* ***Assistant Professor****,*

***Dept. of Computer Science and Engineering****, for sparing his/her valuable time to extend help in every step of my work, which paved the way for smooth progress and fruitful culmination of the project.*

*I would like to thank our Project Coordinator* ***Dr. Project Coordinator****, and all the staff members of Computer Science and Engineering for their support.*

*we are also grateful to my family and friends who provided me with every requirement throughout the course.*

*we would like to thank one and all who directly or indirectly helped me in completing*

*the work successfully.*

*Signature of Student(s)*

# ABSTRACT

Creating effective study aids like flashcards from extensive documents is a traditionally time-consuming process, often hindering efficient learning. This project addresses this challenge by developing the "Document Flashcard Maker," a web-based application designed to automate the generation of question-answer pairs directly from user-uploaded files. Leveraging the power of Google's Gemini Large Language Model (LLM) via its API, the application processes common document formats (PDF, DOCX, PPTX) to intelligently extract key information and formulate concise, relevant flashcards, thereby significantly reducing manual effort and providing users with personalized study tools tailored to their specific materials.

Built as a client-side application using standard web technologies (HTML, CSS, JavaScript), the Document Flashcard Maker features an intuitive user interface with interactive, flippable flashcards and navigation controls. It integrates the Gemini API using Server-Sent Events (SSE) for a responsive user experience, allowing flashcards to be displayed incrementally as they are generated. The application incorporates essential usability features such as a history function to save and revisit previous flashcard sets and a settings panel for API key management, both utilizing browser localStorage for persistence. The resulting tool successfully demonstrates the feasibility of applying modern AI to enhance educational workflows, offering a practical solution for accelerating knowledge acquisition from digital documents.

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| List of Figures | ix |
| List of Tables | x |
| Nomenclature used | xi |
| Chapter 1 | 01 |
| 1. Introduction |  |
| 1.1 Background & Motivation |  |
| 1.2 Objective |  |
| 1.3 Delimitation of research |  |
| 1.4 Benefits of research |  |
| Chapter 2 | 04 |
| 1. LITERATURE SURVEY    1. Literature Review |  |
| 2.2 Inferences Drawn from Literature Review |  |
| Chapter 3 | 06 |
| 3. Problem Formulation and Proposed Work |  |
| 3.1 Introduction |  |
| 3.2 Problem Statement |  |
| 3.3 System Architecture /Model |  |
| 3.4 Proposed Algorithms |  |
| 3.5 Proposed Work |  |
| 4. Implementation | 11 |
| 4.1 Hardware Design and Implementation |  |

|  |  |
| --- | --- |
| **4.2 Software algorithm** |  |
| **Chapter 5** | **12** |
| **5. Results And Discussion** |  |
| **Conclusions And Future Scope** | **16** |
| **References (IEEE format )** | **18** |
| **Appendices** | **Xii** |
|  |  |
| **Appendix – II** | **xvi** |
| **Details of Paper Publication(Along with Paper)** | **x** |
| **Information Regarding Student** | **xvii** |
| **Photograph Along With Guide** | **xviii** |
| **PLAGIARISM REPORT (SHOULD BE TAKEN FROM LIBRARY AND IT SHOULD BE LESS THAN**  **20% THE COMPLETE REPORT – EXCLUDING REFENCES)** |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| Fig. No. | Description of the figure | Page No. |
| 1.1 | App UI | 5 |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| Table No. | Description of the Table | Page No. |
| 5.1 | Summary of Implemented Features and Results | 15 |

# NOMENCLATURE USED

|  |  |
| --- | --- |
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| Base64 | An encoding scheme to represent binary data in an ASCII string format. |
| CSS | Cascading Style Sheets |
| DOM | Document Object Model |
| DOCX | Office Open XML Document (Microsoft Word) |
| Gemini API | A family of large language models developed by Google AI. |
| Glassmorphism | A UI design trend using background blur, transparency, and frosted-glass effects. |
| HTML | HyperText Markup Language |
| HTTP | Hypertext Transfer Protocol |
| IEEE | Institute of Electrical and Electronics Engineers |
| JavaScript (JS) | A high-level programming language used primarily for web development. |
| JSON | JavaScript Object Notation |
| LLM | Large Language Model |
| MIME Type | Multipurpose Internet Mail Extensions type, indicating the nature and format of a document. |
| PDF | Portable Document Format |
| PPTX | Office Open XML Presentation (Microsoft PowerPoint) |
| SSE | Server-Sent Events |
| UI | User Interface |
| UX | User Experience |
| localStorage | Web storage mechanism allowing websites to store key/value pairs in a web browser. |

# INTRODUCTION

## 1.1 Background & Motivation

In the modern educational landscape, efficient learning and knowledge retention strategies are paramount. Traditional methods like manual note-taking and flashcard creation are often time-consuming and laborious, especially when dealing with extensive documents or presentations. Flashcards, in particular, are a proven tool for active recall and spaced repetition, significantly enhancing memory consolidation. However, the process of identifying key information, formulating concise questions, and writing corresponding answers from source material can be a significant bottleneck for students and educators alike.

Simultaneously, the field of Artificial Intelligence (AI), specifically Large Language Models (LLMs), has demonstrated remarkable capabilities in natural language understanding, text generation, and information extraction. Models like Google's Gemini possess the ability to process vast amounts of text and generate human-like responses, summaries, and even structured data formats. This advancement presents a unique opportunity to automate and enhance traditional study methods.

The motivation for this research stems from the intersection of these two areas: the recognized effectiveness of flashcards as a learning tool and the potential of modern LLMs to automate their creation. There is a clear need for a tool that can bridge the gap between digital documents (like PDFs, Word documents, and PowerPoint presentations) and personalized, ready-to-use study aids. By leveraging an LLM, we can potentially transform lengthy documents into concise, targeted question-answer pairs, saving users significant time and effort while promoting more effective learning. This project, the "Document Flashcard Maker," aims to address this need by creating a user-friendly web application powered by the Gemini API.

## 1.2 Objective

The primary objective of this research is to design, implement, and evaluate a web-based application, the "Document Flashcard Maker," capable of automatically generating interactive flashcards from user-uploaded documents.

Specific sub-objectives include:

1. Developing a client-side web application using HTML, CSS, and JavaScript that provides a clean and intuitive user interface (UI) adopting a modern 'glassmorphism' aesthetic.
2. Implementing functionality to accept user uploads of common document formats, specifically PDF, DOCX, and PPTX files.
3. Integrating the Google Gemini API (specifically the gemini-2.0-flash model via its streaming endpoint) to process the content of the uploaded documents.
4. Designing effective prompts and utilizing the API's streaming capabilities to generate relevant, unique, and concise question-answer pairs in JSON format.
5. Developing an interactive flashcard component allowing users to view questions, flip cards to see answers, and navigate through the generated set.
6. Implementing a history feature using browser localStorage to save previous generation sessions (document name, generated cards) for later review.
7. Providing a settings interface for users to securely input and save their Gemini API key in localStorage.
8. Implementing controls for managing the generation process, including batch generation and setting a maximum limit on the number of flashcards per document (MAX\_CARDS).

## 1.3 Delimitation of research

The scope of this research is defined by the following delimitations:

1. **File Formats:** The application will initially only support PDF, DOCX, and PPTX file formats. Other formats like images, plain text, or web pages are outside the scope of this phase.
2. **AI Model:** The project exclusively utilizes the Google Gemini API (gemini-2.0-flash model). Comparative analysis with other LLMs is not included.
3. **Generation Quality:** The quality, accuracy, and relevance of the generated flashcards are inherently dependent on the capabilities of the Gemini model and the quality/structure of the source document. Fine-tuning the model itself is beyond the scope; reliance is placed on effective prompt engineering.
4. **Language:** While the Gemini model supports multiple languages, the primary focus and testing of the UI and prompts are geared towards English language documents.
5. **Learning Algorithms:** The application focuses on generation and display. Advanced learning features like Spaced Repetition Systems (SRS) are not implemented in this version.
6. **Platform:** The application is designed as a client-side web application running entirely in the user's browser. No custom server-side backend (beyond the external Gemini API) is developed.
7. **Error Handling:** While basic error handling for file reading and API communication is included, exhaustive handling of all possible API errors or document parsing issues is not guaranteed.
8. **Security:** The API key is stored in browser localStorage, which is convenient but may not be suitable for highly sensitive environments. More robust key management solutions are not explored.

## 1.4 Benefits of research

The successful implementation of the "Document Flashcard Maker" offers several potential benefits:

1. **Time Efficiency:** Automates the tedious process of creating flashcards, saving significant time for students, educators, and professionals who need to learn or review document content.
2. **Enhanced Learning:** Provides readily available active recall tools directly derived from specific source materials, potentially improving comprehension and retention.
3. **Accessibility:** As a web-based tool, it is accessible from any device with a modern web browser, requiring no installation.
4. **Personalization:** Generates flashcards tailored to the user's specific documents, ensuring relevance.
5. **Document Versatility:** Handles common document types used in academic and professional settings (PDF, DOCX, PPTX).
6. **Demonstration of AI Application:** Serves as a practical example of leveraging modern LLMs for creating valuable educational technology tools.
7. **Usability:** Features like history management and a clear UI aim to provide a positive user experience (UX).

# 2. LITERATURE SURVEY

## 2.1 Literature Review

The development of the "Document Flashcard Maker" builds upon existing work in several areas: educational technology, natural language processing, and web development practices.

Firstly, the pedagogical value of flashcards is well-established. Studies have consistently shown that active recall, facilitated by flashcards, is a highly effective learning strategy compared to passive review. Platforms like Anki and Quizlet have popularized digital flashcards, often incorporating Spaced Repetition Systems (SRS) to optimize review schedules. However, these platforms typically require manual creation or importing pre-made decks, lacking the ability to automatically generate cards from arbitrary user documents.

Secondly, the advent of powerful LLMs has opened new frontiers in automated content generation for educational purposes. Research has explored using AI for text summarization, automatic question generation (QG), and creating educational dialogues. Early QG systems often relied on rule-based methods or template filling, but recent transformer-based models like those in the GPT and Gemini families demonstrate superior fluency and contextual understanding. Several studies focus on generating question-answer pairs from text, though often targeting specific domains or requiring complex pipelines. Integrating these capabilities directly with user-uploaded documents in various formats presents an ongoing challenge.

Thirdly, web technologies provide the foundation for accessible and interactive applications. Modern JavaScript frameworks and vanilla JS, combined with HTML5 and CSS3, enable the creation of sophisticated client-side applications. Techniques like using the FileReader API for handling local files, fetch for asynchronous API communication (including handling streams like SSE), and localStorage for client-side persistence are standard practices. UI design trends like glassmorphism aim to enhance visual appeal and user engagement. The use of streaming API endpoints, as employed in this project with Gemini, is crucial for handling potentially long generation processes without freezing the UI and providing incremental feedback to the user.

## 2.2 Inferences Drawn from Literature Review

Based on the literature surveyed, the following inferences can be drawn:

1. There is strong pedagogical justification for using flashcards as a learning tool.
2. While popular digital flashcard platforms exist, there is a gap in tools that seamlessly integrate document uploading (especially diverse formats like PDF, DOCX, PPTX) and automatic, AI-powered flashcard generation based *on that specific document's content*.
3. LLMs like Gemini have the core capability needed for extracting information and generating question-answer pairs from text, but effective implementation requires careful prompt engineering and handling of API interactions.
4. Modern web technologies provide the necessary tools (File API, Fetch API, SSE, localStorage, DOM manipulation) to build such an application entirely on the client-side, enhancing accessibility.
5. Using streaming API endpoints is advantageous for providing a responsive UX during potentially time-consuming AI generation tasks.
6. The proposed "Document Flashcard Maker" directly addresses the identified gap by combining document processing, LLM integration via streaming, interactive display, and history management within a single, accessible web application.



Figure .1 - App UI

# 3. PROBLEM FORMULATION AND PROPOSED WORK

## 3.1 Introduction

As established, the manual creation of flashcards from documents is inefficient. While AI offers a solution, integrating this capability into a user-friendly tool that handles common document formats and provides a smooth workflow remains a challenge. This chapter formally defines the problem addressed by this project and outlines the proposed system architecture and methodology.

## 3.2 Problem Statement

To design and develop a client-side web application ("Document Flashcard Maker") that:

* Accepts user-uploaded documents in PDF, DOCX, and PPTX formats.
* Securely processes these documents within the user's browser to extract textual content suitable for an LLM.
* Interacts with the Google Gemini API (gemini-2.0-flash model) using a streaming connection (SSE).
* Employs carefully crafted prompts and maintains chat history context to instruct the LLM to generate a set of unique, relevant, and concise flashcards (question-answer pairs) directly based on the uploaded document's content.
* Parses the streamed JSON responses from the API to incrementally build a list of flashcards, up to a defined maximum (MAX\_CARDS = 35).
* Presents these flashcards interactively in the UI, allowing flipping and navigation.
* Persists generated flashcard sets and the user's API key using browser localStorage for history recall and ease of use.
* Provides clear user feedback regarding the generation process and manages application state effectively.

## 3.3 System Architecture / Model

The proposed system is a client-side web application architecture, meaning the core logic runs within the user's web browser. No custom server-side backend is required, aside from the external Google Gemini API service. The main components are:

1. **Frontend (Client-Side Application):**
   * **UI Layer (HTML/CSS):** Defines the structure (using HTML elements like input[type=file], button, div.flashcard, etc.) and visual presentation (using CSS with variables for the "dusk" theme and glassmorphism effects). Handles layout (sidebar, main content) and responsiveness.
   * **Logic Layer (JavaScript):** Manages the application's state (flashcards, currentCardIndex, apiKey, history, chatHistory, isGenerating). Handles user interactions (button clicks, file selection) via event listeners. Contains logic for:
     + File Handling (FileReader API to read files as Base64).
     + MIME Type determination.
     + API Interaction (fetch API to call Gemini streaming endpoint, SSE processing).
     + Prompt Construction (including SYSTEM\_INSTRUCTION and dynamic requests).
     + JSON Parsing (from API stream).
     + DOM Manipulation (updating flashcard content, counters, button states, history list).
     + State Management.
     + localStorage Interaction (saving/loading API key and history).
2. **External Services:**
   * **Google Gemini API:** The LLM service (generativelanguage.googleapis.com) that receives the document context (as Base64 data initially, then via chat history) and prompts, and streams back the generated flashcard JSON objects.
3. **Data Storage (Client-Side):**
   * **Browser localStorage:** Used to store the user's Gemini API key and the flashcard generation history (list of sessions with metadata and card data).

**(Diagram would typically be included here showing UI -> JS Logic -> Fetch -> Gemini API -> Fetch Response (SSE) -> JS Logic -> UI Update, and JS Logic <-> localStorage)**

**3.4 Proposed Algorithms**

The core processes involve several algorithmic steps, primarily implemented in JavaScript:

1. **File Processing Algorithm:**
   * On file input change (handleFileSelect):
     + Get the File object.
     + If a file exists: Display filename, reset current state (flashcards, index, history ID, chat history).
     + Use FileReader.readAsDataURL to read the file.
     + On load (readFileAsBase64): Extract the Base64 string (removing the prefix). Determine mimeType using getMimeType.
   * Return Base64 data and MIME type.
2. **Flashcard Generation Algorithm (generateFlashcards - Streaming):**
   * Check if API key exists and generation limit (MAX\_CARDS) isn't reached.
   * Calculate requestedCount (min of CARDS\_PER\_BATCH and remaining allowed cards).
   * Set isGenerating state to true, update UI status.
   * Construct the Gemini API endpoint URL for streaming (streamGenerateContent).
   * Prepare currentUserMessage:
     + If chatHistory is short (initial call): Include prompt with requestedCount, format instructions, *and* the inlineData (Base64 file content + MIME type).
     + Else (subsequent calls): Include prompt requesting requestedCount *more* unique cards based on the document provided earlier (relying on API's context).
   * Push currentUserMessage to chatHistory.
   * Prepare the request body including contents: chatHistory, generationConfig, and safetySettings.
   * Call fetch with "POST", headers, and stringified body.
   * Handle initial response: Check response.ok. If not okay, read error and throw.
   * Get ReadableStream reader (response.body.getReader()).
   * Initialize decoder, buffer, currentLine, accumulatedResponseText, cardsGeneratedInThisStream.
   * **Streaming Loop:**
     + reader.read().
     + If done, break loop.
     + Decode value and append to buffer.
     + Split buffer into lines (\n). Keep the last (potentially incomplete) line in buffer.
     + For each complete line:
       - If line starts with data:: Extract JSON string.
       - Parse JSON data chunk.
       - Extract text part from candidates.content.parts.
       - Append text to accumulatedResponseText and currentLine.
       - Split currentLine by \n (as model might output multiple JSONs in one chunk). Keep last part in currentLine.
       - For each complete JSON line:
         * Trim and parse the line as JSON (card).
         * Validate card has question and answer strings.
         * If valid and flashcards.length < MAX\_CARDS: Push card to flashcards array, increment cardsGeneratedInThisStream, update UI (updateCardDisplay, updateCardCounter, updateControls), update history item in localStorage if currentHistoryId exists.
       - If flashcards.length >= MAX\_CARDS, break inner loop (and potentially outer stream loop).
   * **After Loop:** Process any remaining currentLine for a final JSON object.
   * If accumulatedResponseText is not empty, push { role: "model", parts: [{ text: accumulatedResponseText }] } to chatHistory.
   * Update final UI status.
   * Handle potential errors during streaming, potentially removing the last user message from chatHistory on failure.
   * Set isGenerating state to false.
3. **History Management Algorithms:**
   * loadHistory: Read 'flashcardHistory' from localStorage, parse JSON (handle errors/reset if invalid), store in history array, call displayHistory.
   * saveHistoryItem: Create historyItem object (ID, filename, date/time, cards), unshift to history array, limit array size, save stringified history to localStorage, set currentHistoryId, call displayHistory.
   * updateHistoryItem: Find item by ID in history, update its cards and cardCount, save updated history to localStorage, call displayHistory.
   * displayHistory: Clear current list, iterate history array, create li elements with data attributes and content, append to historyList, highlight active item based on currentHistoryId.
   * handleHistoryClick: Get historyId from clicked li, find item in history, load its cards into flashcards state, reset index, set currentHistoryId, clear chatHistory, update UI, refresh history display.

## 3.5 Proposed Work

The development process followed these general steps:

1. **Requirement Analysis:** Defined the core features, target users, supported formats, and technical constraints based on the project objectives.
2. **UI/UX Design:** Designed the user interface structure (HTML) and visual style (CSS), opting for a clean, dark theme with glassmorphism elements for visual appeal and clarity. Focused on intuitive controls for file upload, card navigation, and settings.
3. **Frontend Development:**
   * Implemented the static structure using HTML.
   * Styled the application using CSS, including layout, theme variables, and component styles (buttons, flashcards, popover, sidebar).
   * Developed the core client-side logic using vanilla JavaScript (script.js).
4. **File Handling Implementation:** Integrated the FileReader API to read user-selected files as Base64 data.
5. **API Integration:** Implemented the fetch call to the Gemini streaming endpoint (streamGenerateContent), including constructing the request payload (with prompts, file data, history, configuration) and processing the Server-Sent Events (SSE) response. Developed logic to parse JSON objects from the stream.
6. **Flashcard Component:** Created the HTML/CSS for the flippable flashcard and the JavaScript logic to handle the flip animation (is-flipped class) and display question/answer content.
7. **Navigation and Controls:** Implemented previous/next buttons and the card counter, including logic to disable buttons appropriately and trigger generation of more cards when nearing the end (showNextCard).
8. **History Feature:** Developed functions to save, load, update, and display generation history using browser localStorage. Implemented click handling to load past sessions.
9. **Settings Feature:** Created the popover UI for API key input and implemented logic to save/load the key from localStorage.
10. **Testing and Refinement:** Conducted functional testing (file uploads, generation, navigation, history, settings) and usability testing. Refined prompts and UI based on results. Added basic error handling and status messages.

# 4. IMPLEMENTATION

## 4.1 Hardware Design and Implementation

This project is a software-based web application designed to run in a standard web browser on the client-side. As such, there was no custom hardware design or implementation involved. The necessary hardware components are:

1. **Client Device:** Any computer (desktop, laptop) or mobile device capable of running a modern web browser (like Chrome, Firefox, Safari, Edge) that supports HTML5, CSS3, JavaScript ES6+, the Fetch API, the FileReader API, and localStorage. The performance depends on the client's CPU and RAM for processing JavaScript and rendering the UI.
2. **Server Infrastructure (External):** The Google Cloud infrastructure hosts the Gemini API. The specifics of this hardware are managed by Google and are abstracted away from the application developer and user. Communication occurs over standard HTTPS protocols.

Therefore, sections 4.1.1 through 4.1.5 detailing specific custom hardware components are not applicable to this software-centric project.

## 4.2 Software algorithm

The core software implementation resides in the script.js file, orchestrating the interactions between the HTML structure, CSS styling, browser APIs, and the external Gemini API. The implementation follows the algorithms outlined in Chapter 3. Key aspects of the implementation include:

1. **DOM Element Selection:** Constant variables are declared at the beginning of the script to hold references to essential DOM elements (e.g., fileInput, generateBtn, flashcardElement, apiKeyInput, historyList), improving performance and code readability.
2. **State Management:** Global variables (flashcards, currentCardIndex, apiKey, isGenerating, currentHistoryId, history, chatHistory) are used to maintain the application's state throughout the user session. apiKey and history are persisted across sessions using localStorage.
3. **Event Handling (addEventListeners):** Event listeners are attached to UI elements to trigger corresponding functions:
   * generateBtn: Calls handleGenerateClick.
   * flashcardElement: Calls flipCard.
   * prevCardBtn/nextCardBtn: Call showPreviousCard/showNextCard.
   * settingsBtn/closeSettingsBtn: Call toggleSettingsPopover.
   * saveSettingsBtn: Calls saveSettings.
   * fileInput: Calls handleFileSelect.
   * historyList: Uses event delegation to call handleHistoryClick.
4. **File Handling Logic (handleFileSelect, readFileAsBase64, getMimeType):** Implements the file reading process using FileReader asynchronously, converting the file to a Base64 string and determining its MIME type based on the file extension. This prepares the data for the API request.
5. **API Interaction (generateFlashcards):** This asynchronous function is central to the application.
   * It performs pre-checks (API key, card limits).
   * Constructs the API request, critically including the contents array (chatHistory) which maintains the conversation context with the Gemini model. The initial user message includes the inlineData part containing the Base64 file content. Subsequent messages rely on the model remembering the context and just ask for more cards.
   * Uses fetch to make the POST request to the streaming endpoint (/v1/models/gemini-2.0-flash:streamGenerateContent?key=...&alt=sse).
   * Processes the response stream using ReadableStream and TextDecoder. It iteratively reads chunks, decodes them, buffers potentially incomplete lines, and processes complete Server-Sent Events (data: lines).
   * Crucially, it parses the JSON content within the data part of the SSE message. It handles potential multiple JSON objects per chunk and incomplete JSON objects spanning chunks by buffering (currentLine).
   * Validates parsed objects (card.question && card.answer) before adding them to the flashcards array.
   * Updates the UI incrementally as cards are received using updateCardDisplay, updateCardCounter, and updateControls.
   * Updates the corresponding history item in localStorage in real-time if loaded from history.
   * Maintains the chatHistory by adding the full accumulated model response after the stream finishes, ensuring context for subsequent "generate more" requests.
   * Includes error handling for the fetch call and stream processing.
6. **UI Update Functions (updateCardDisplay, flipCard, showNextCard, showPreviousCard, updateControls, updateCardCounter):** These functions manipulate the DOM to reflect the current application state: displaying the correct question/answer, toggling the is-flipped class, updating the counter text (X / Y), and enabling/disabling navigation and generation buttons based on the state (currentCardIndex, flashcards.length, isGenerating). The showNextCard function also includes the logic to trigger generateFlashcards proactively when the user gets close to the end of the current batch.
7. **Settings Management (toggleSettingsPopover, saveSettings):** Handles the display of the settings popover and the saving/loading of the apiKey to/from localStorage.
8. **History Management (loadHistory, saveHistoryItem, updateHistoryItem, displayHistory, handleHistoryClick, applyHistoryStyles):** Implements the full lifecycle of the history feature: loading from storage on startup, saving new sessions, updating existing sessions (when more cards are generated), rendering the history list in the sidebar (including applying dynamic CSS styles), and handling clicks to load previous states. Uses localStorage as the persistence mechanism.
9. **Constants and Configuration:** Uses constants (GEMINI\_MODEL, SYSTEM\_INSTRUCTION, CARDS\_PER\_BATCH, MAX\_CARDS) for easy configuration and clarity. The SYSTEM\_INSTRUCTION is particularly important for guiding the LLM's output format.

The implementation utilizes modern JavaScript features (async/await, fetch, arrow functions, template literals) and adheres to a structured approach, separating concerns like API interaction, UI updates, state management, and local storage operations into distinct functions.

# 5. RESULTS AND DISCUSSION

The implementation phase resulted in a functional "Document Flashcard Maker" web application that successfully meets the objectives outlined in Chapter 1. Users can upload documents (PDF, DOCX, PPTX), and upon providing a valid Gemini API key, the application initiates the flashcard generation process.

**Results:**

1. **Functionality:** The application correctly handles file uploads and reads their content. It successfully communicates with the Gemini API's streaming endpoint. Generated question-answer pairs, formatted as JSON by the API according to the system prompt, are received and parsed correctly.
2. **User Interface:** The UI, styled with CSS featuring a dark "dusk" theme and glassmorphism effects, presents a modern and visually appealing interface. The flashcard component flips smoothly on click, revealing the answer. Navigation buttons (Previous/Next) and the card counter (X / Y) function as expected. The history list in the sidebar dynamically updates and allows loading previous sessions. The settings popover provides a clear way to manage the API key.
3. **Flashcard Generation:** The application generates flashcards based on the document content. The use of the streaming API provides real-time feedback, with cards appearing incrementally as they are generated, enhancing the user experience compared to waiting for a full batch. The quality and relevance of the cards depend heavily on the source document's clarity and the Gemini model's interpretation, exhibiting variability as expected from current LLMs. The mechanism to request more cards (CARDS\_PER\_BATCH) when nearing the end of the current set generally works, up to the MAX\_CARDS limit.
4. **History & Settings:** The localStorage implementation successfully persists the API key and generation history across browser sessions. Loading from history restores the flashcard set correctly.
5. **Performance:** For moderately sized documents, the application remains responsive. The streaming approach prevents the UI from freezing during API calls. Performance bottlenecks are primarily related to the file reading time (for very large files) and the latency of the Gemini API response.

**Discussion:**

The results demonstrate the feasibility of using client-side JavaScript and a powerful LLM API to create a useful educational tool. The integration of the Gemini streaming API (streamGenerateContent) proved particularly effective for handling the potentially lengthy generation process, offering a much better UX than a standard request-response cycle. The incremental display of cards keeps the user engaged and informed of progress.

The quality of generated flashcards is a key area for discussion. While often relevant and useful, the AI occasionally produces overly generic questions, overly specific/context-dependent answers, or sometimes misinterprets nuanced text. The prompt engineering, especially the SYSTEM\_INSTRUCTION demanding unique JSON objects per line, was crucial for reliable parsing but doesn't guarantee pedagogical perfection. The uniqueness constraint in the prompt (Do not repeat questions) helps but isn't foolproof, especially across multiple generation requests within the same session (chatHistory context helps mitigate this).

The choice of client-side implementation offers advantages in simplicity (no backend deployment needed) and privacy (document data is processed locally before being sent to the API, though content *is* sent to Google). However, it relies entirely on the user's browser capabilities and network connection. Storing the API key in localStorage is convenient but less secure than server-side solutions.

Challenges encountered during development included handling the asynchronous nature of file reading and API calls, reliably parsing the streamed JSON data (which occasionally had minor formatting inconsistencies from the API requiring robust parsing logic), and managing the application state effectively, especially when loading from history or generating additional card batches. The MAX\_CARDS limit was implemented as a practical measure to prevent runaway API usage and associated costs.

Overall, the "Document Flashcard Maker" successfully automates flashcard creation, offering significant time savings. While AI-generated content may require user review or occasional editing (a potential future feature), it provides a strong starting point for study materials derived directly from source documents.

|  |  |
| --- | --- |
| **Feature** | **Key Result/Observation** |
| **Document Upload** | Supports PDF, DOCX, PPTX formats via browser file input. |
| **File Processing** | Successfully reads files as Base64 using FileReader API. |
| **Gemini API Integration** | Communicates with gemini-2.0-flash model via fetch. |
| **Streaming Generation (SSE)** | Receives & parses JSON responses incrementally via Server-Sent Events. |
| **Flashcard Generation** | Generates Q/A pairs based on document content; quality varies with source & model. |
| **Interactive Flashcard UI** | Cards display Q/A and flip on click; navigation buttons functional. |
| **Batch Generation Control** | Generates CARDS\_PER\_BATCH (5) initially & requests more near end, up to MAX\_CARDS (35). |
| **History Management** | Saves/Loads sessions (filename, cards) to/from localStorage. |
| **Settings (API Key)** | Popover allows users to input/save API key to localStorage. |
| **UI/UX (Glassmorphism)** | Visual design applied using CSS; provides status updates during generation. |

Table 5.1 - Summary of Implemented Features and Results

# CONCLUSIONS AND FUTURE SCOPE

## Conclusions:

This project successfully achieved its objective of developing the "Document Flashcard Maker," a client-side web application that leverages the Google Gemini API to automatically generate flashcards from PDF, DOCX, and PPTX documents. The application provides an intuitive interface for file uploading, interactive flashcard review (question/answer flipping, navigation), history management, and API key configuration.

The use of the Gemini streaming API demonstrated significant advantages in user experience by providing incremental results. The implementation effectively utilizes modern web technologies (HTML5, CSS3, JavaScript, Fetch API, FileReader, localStorage) to create a functional and accessible tool. The project highlights the potential of LLMs to transform traditional study methods by automating content creation tasks. While the quality of AI-generated content has inherent variability, the application provides a valuable service by significantly reducing the time and effort required for manual flashcard creation. The history feature further enhances its utility by allowing users to revisit previously generated study sets.

## Future Scope:

While the current application is functional, several avenues exist for future enhancement:

1. **Expanded File Format Support:** Add support for other common formats like plain text (.txt), Markdown (.md), and potentially image files (using OCR via an API or library).
2. **Editable Flashcards:** Allow users to edit the generated questions and answers to correct inaccuracies or tailor them further.
3. **Spaced Repetition System (SRS):** Integrate an SRS algorithm (like SM-2 used in Anki) to schedule card reviews for optimal learning.
4. **Improved Generation Control:** Offer users options to influence generation, such as specifying the desired difficulty level, type of questions (e.g., definitions, concepts), or focusing on specific sections of the document.
5. **Model Selection:** Allow users to choose different Gemini models (e.g., Pro models for potentially higher quality at a higher cost/latency) or even integrate other LLM APIs.
6. **Backend Implementation:** Develop a server-side component to handle tasks like:
   * More secure API key management.
   * Centralized user accounts and history storage.
   * Potentially pre-processing documents on the server to handle very large files or complex formats more robustly.
7. **Enhanced Error Handling:** Implement more detailed error reporting from the API and document processing stages.
8. **Sharing and Collaboration:** Allow users to share generated flashcard decks with others.
9. **Offline Access:** Explore Progressive Web App (PWA) features for improved offline access to saved history items.
10. **Accessibility Improvements:** Conduct thorough accessibility testing and implement ARIA attributes to ensure usability for individuals using assistive technologies.

These future steps could further enhance the "Document Flashcard Maker," transforming it into a more comprehensive and robust learning platform.

## REFERENCES (IEEE format)

1. J. Dunlosky, K. A. Rawson, E. J. Marsh, M. J. Nathan, and D. T. Willingham, "Improving Students’ Learning With Effective Learning Techniques: Promising Directions From Cognitive and Educational Psychology," \*Psychological Science in the Public Interest\*, vol. 14, no. 1, pp. 4–58, Jan. 2013.
2. M. Kerres, "Digitalization and Education: On the Socio-Technical Transformation of Education," \*Open Education Studies\*, vol. 2, no. 1, pp. 184–193, Jan. 2020.
3. P. C. Karpicke and J. R. Blunt, "Retrieval Practice Produces More Learning than Elaborative Studying with Concept Mapping," \*Science\*, vol. 331, no. 6018, pp. 772–775, Feb. 2011.
4. A. Vaswani et al., "Attention Is All You Need," in \*Advances in Neural Information Processing Systems 30 (NIPS 2017)\*, 2017, pp. 5998–6008.
5. Google, "Gemini: A Family of Capabilities," \*DeepMind\*, Dec. 2023. [Online]. Available: https://deepmind.google/technologies/gemini/
6. O. Zawacki-Richter, V. I. Marín, M. Bond, and F. Gouverneur, "Systematic review of research on artificial intelligence applications in higher education – where are the educators?," \*Int J Educ Technol High Educ\*, vol. 16, no. 39, 2019.
7. J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," \*arXiv preprint arXiv:2201.11903\*, 2022.
8. P. A. Wozniak, "Theoretical Aspects of Spaced Repetition in Learning," \*Ph.D. Thesis, University of Economics\*, Wroclaw, Poland, 1990.
9. K. K. Patel, R. R. Shah, and H. M. Borisa, "A Survey on Automatic Question Generation from Text," \*Engineering Applications of Artificial Intelligence\*, vol. 101, p. 104209, May 2021.
10. X. Du, J. Shao, and C. V. Jawahar, "Learning to Ask: Neural Question Generation for Reading Comprehension," \*arXiv preprint arXiv:1705.00106\*, 2017.
11. M. Haverbeke, \*Eloquent JavaScript\*, 3rd ed. No Starch Press, 2018.
12. MDN Web Docs, "Web Storage API," \*Mozilla Developer Network\*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web\_Storage\_API
13. A. Pratas, "Glassmorphism in user interfaces," \*UX Collective\*, Nov. 2020. [Online]. Available: https://uxdesign.cc/glassmorphism-in-user-interfaces-1f59bb6346c9
14. MDN Web Docs, "Using server-sent events," \*Mozilla Developer Network\*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent\_events/Using\_server-sent\_events
15. ISO 32000-2:2020, "Document management — Portable document format — Part 2: PDF 2.0," \*International Organization for Standardization\*, Geneva, CH, 2020.
16. ECMA-376, "Office Open XML File Formats," \*Ecma International\*, 5th ed., Part 1, Dec. 2016.
17. J. Nielsen, \*Usability Engineering\*. Morgan Kaufmann, 1993.
18. W3C, "Web Content Accessibility Guidelines (WCAG) 2.1," \*W3C Recommendation\*, Jun. 2018. [Online]. Available: https://www.w3.org/TR/WCAG21/
19. WHATWG, "Fetch Standard," \*Web Hypertext Application Technology Working Group\*. [Online]. Available: https://fetch.spec.whatwg.org/
20. K. Simpson, \*You Don't Know JS Yet: Async & Performance\*, 2nd ed. GetiPub, Inc., 2021.

**Appendix-I**

**Photographs (if any during visit to sites/ places regarding project )**

**Appendix - II Source Code**

Index.html

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>Flashcard Maker</title>

    <link *rel*="stylesheet" *href*="style.css">

</head>

<body>

    <aside *class*="sidebar">

        <h2>History</h2>

        <ul *id*="history-list">

*<!-- History items will be added here -->*

        </ul>

        <button *id*="settings-btn" *class*="settings-button">Settings</button>

    </aside>

    <main *class*="main-content">

        <h1>Document Flashcard Maker</h1>

        <section *class*="input-section glassmorphism">

            <h2>Upload Your Document</h2>

            <p>Upload a PDF, DOCX, or PPTX file to generate flashcards.</p>

            <input *type*="file" *id*="file-input" *accept*=".pdf,.docx,.pptx">

            <button *id*="generate-btn">Generate Flashcards</button>

            <p *id*="upload-status"></p>

        </section>

        <section *id*="flashcard-section" *class*="flashcard-container" *style*="display: none;">

             <h2>Flashcards</h2>

             <div *id*="flashcard" *class*="flashcard glassmorphism">

                 <div *class*="flashcard-inner">

                     <div *class*="flashcard-front">

                         <p *id*="flashcard-question">Question will appear here.</p>

                     </div>

                     <div *class*="flashcard-back">

                         <p *id*="flashcard-answer">Answer will appear here.</p>

                     </div>

                 </div>

             </div>

             <div *class*="flashcard-controls">

                <button *id*="prev-card-btn" *disabled*>Previous</button>

                <span *id*="card-counter">0 / 0</span>

                <button *id*="next-card-btn" *disabled*>Next</button>

             </div>

        </section>

    </main>

    <div *id*="settings-popover" *class*="popover glassmorphism" *style*="display: none;">

        <h2>Settings</h2>

        <label *for*="api-key-input">Gemini API Key:</label>

        <input *type*="password" *id*="api-key-input">

        <button *id*="save-settings-btn">Save</button>

        <button *id*="close-settings-btn">Close</button>

    </div>

    <script *src*="script.js"></script>

*<!-- Placeholder for Gemini API SDK if needed, or use fetch -->*

</body>

</html>

Script.js

*// DOM Elements*

const fileInput = document.**getElementById**('file-input');

const generateBtn = document.**getElementById**('generate-btn');

const uploadStatus = document.**getElementById**('upload-status');

const flashcardSection = document.**getElementById**('flashcard-section');

const flashcardElement = document.**getElementById**('flashcard');

const flashcardQuestion = document.**getElementById**('flashcard-question');

const flashcardAnswer = document.**getElementById**('flashcard-answer');

const prevCardBtn = document.**getElementById**('prev-card-btn');

const nextCardBtn = document.**getElementById**('next-card-btn');

const cardCounter = document.**getElementById**('card-counter');

const settingsBtn = document.**getElementById**('settings-btn');

const settingsPopover = document.**getElementById**('settings-popover');

const apiKeyInput = document.**getElementById**('api-key-input');

const saveSettingsBtn = document.**getElementById**('save-settings-btn');

const closeSettingsBtn = document.**getElementById**('close-settings-btn');

const historyList = document.**getElementById**('history-list');

*// State*

let flashcards = []; *// Array to hold generated flashcards { question: '...', answer: '...' }*

let currentCardIndex = 0;

let apiKey = localStorage.**getItem**('geminiApiKey') || '';

let isGenerating = false;

let currentHistoryId = null;

let history = [];

let chatHistory = []; *// To maintain conversation context with Gemini*

const CARDS\_PER\_BATCH = 5;

const MAX\_CARDS = 35;

*// Constants for Gemini API*

const GEMINI\_MODEL = "gemini-2.0-flash";

const SYSTEM\_INSTRUCTION = `You are a flashcard generator. Create unique and educational flashcards based on the uploaded document. Output ONLY JSON objects, one per line, each containing a "question" and an "answer" key. Example:\n{"question": "Q1?", "answer": "A1"}\n{"question": "Q2?", "answer": "A2"}\nDo not include any other text, explanations, markdown formatting, or JSON array brackets. Ensure questions are unique and different from previous ones in the chat history.`;

*// --- Initialization ---*

function **initialize**() {

    if (apiKey) {

        apiKeyInput.value = apiKey;

        console.**log**("API Key loaded from localStorage.");

    } else {

        console.**log**("No API Key found in localStorage. Please set one in Settings.");

    }

**loadHistory**();

**addEventListeners**();

}

*// --- Event Listeners ---*

function **addEventListeners**() {

    generateBtn.**addEventListener**('click', **handleGenerateClick**);

    flashcardElement.**addEventListener**('click', **flipCard**);

    prevCardBtn.**addEventListener**('click', **showPreviousCard**);

    nextCardBtn.**addEventListener**('click', **showNextCard**);

    settingsBtn.**addEventListener**('click', **toggleSettingsPopover**);

    closeSettingsBtn.**addEventListener**('click', **toggleSettingsPopover**);

    saveSettingsBtn.**addEventListener**('click', **saveSettings**);

    fileInput.**addEventListener**('change', **handleFileSelect**);

    historyList.**addEventListener**('click', **handleHistoryClick**);

}

*// --- File Handling ---*

function **handleFileSelect**(event) {

    const file = event.target.files[0];

    if (file) {

        uploadStatus.textContent = `Selected file: ${file.name}`;

*// Reset previous generation state if a new file is selected*

        flashcards = [];

        currentCardIndex = 0;

        currentHistoryId = null;

        chatHistory = []; *// Reset chat history with new file*

        flashcardSection.style.display = 'none';

**updateCardDisplay**();

**updateControls**();

    } else {

        uploadStatus.textContent = '';

    }

}

async function **handleGenerateClick**() {

    if (isGenerating) {

        console.**log**("Already generating...");

        return;

    }

    const file = fileInput.files[0];

    if (!file) {

        uploadStatus.textContent = 'Please select a file first.';

        return;

    }

    if (!apiKey) {

        uploadStatus.textContent = 'Please set your Gemini API Key in Settings.';

        settingsPopover.style.display = 'block';

        return;

    }

    isGenerating = true;

    generateBtn.disabled = true;

    uploadStatus.textContent = 'Processing file and generating flashcards...';

    flashcardSection.style.display = 'none';

    try {

*// Read file as base64*

        const fileData = await **readFileAsBase64**(file);

        if (!fileData) {

            throw new **Error**("Could not read file.");

        }

*// Determine MIME type based on file extension*

        const mimeType = **getMimeType**(file.name);

        console.**log**(`File read as ${mimeType}, generating flashcards...`);

*// Reset flashcards and chat history for new generation*

        if (currentHistoryId === null) {

            flashcards = [];

            chatHistory = []; *// Only reset if not continuing from history*

        }

        currentCardIndex = 0;

*// Generate initial batch of flashcards*

        await **generateFlashcards**(file.name, fileData, mimeType, CARDS\_PER\_BATCH);

        if (flashcards.length > 0) {

            flashcardSection.style.display = 'block';

            currentCardIndex = 0;

**updateCardDisplay**();

*// Save to history if this is a new generation (not loaded from history)*

            if (currentHistoryId === null) {

**saveHistoryItem**(file.name, flashcards);

            }

        } else {

            uploadStatus.textContent = 'Could not generate flashcards from the document.';

        }

    } catch (error) {

        console.**error**("Error during generation:", error);

        uploadStatus.textContent = `Error: ${error.message}`;

        flashcardSection.style.display = 'none';

    } finally {

        isGenerating = false;

        generateBtn.disabled = false;

**updateControls**();

    }

}

*// Read file as base64 for inlineData*

function **readFileAsBase64**(file) {

    return new **Promise**((resolve, reject) => {

        const reader = new **FileReader**();

        reader.**onload** = (event) => {

*// Get base64 string without the prefix (e.g., "data:application/pdf;base64,")*

            const base64String = event.target.result.**split**(',')[1];

**resolve**(base64String);

        };

        reader.**onerror** = (error) => {

            console.**error**("Error reading file:", error);

**reject**(error);

        };

        reader.**readAsDataURL**(file);

    });

}

*// Determine MIME type based on file extension*

function **getMimeType**(filename) {

    const extension = filename.**split**('.').**pop**().**toLowerCase**();

    const mimeTypes = {

        'pdf': 'application/pdf',

        'docx': 'application/vnd.openxmlformats-officedocument.wordprocessingml.document',

        'pptx': 'application/vnd.openxmlformats-officedocument.presentationml.presentation'

    };

    return mimeTypes[extension] || 'application/octet-stream';

}

*// --- Gemini API Integration (Streaming) ---*

async function **generateFlashcards**(filename, fileData, mimeType, count) {

    if (!apiKey) throw new **Error**("API Key not set.");

    if (flashcards.length >= MAX\_CARDS) return;

    const requestedCount = Math.**min**(count, MAX\_CARDS - flashcards.length);

    if (requestedCount <= 0) return;

    uploadStatus.textContent = `Generating ${requestedCount} flashcards... (Streaming)`;

*// Use the streaming endpoint (generateContent with alt=sse)*

    const endpoint = `https://generativelanguage.googleapis.com/v1/models/${GEMINI\_MODEL}:streamGenerateContent?key=${**encodeURIComponent**(apiKey)}&alt=sse`;

*// Add user message with file (only if it's the first call for this file)*

*// Subsequent calls only need the request for more cards*

    let currentUserMessage;

    if (chatHistory.length <= 1) { *// Assuming chatHistory[0] is system instruction*

        currentUserMessage = {

            role: "user",

            parts: [

                { text: `Generate ${requestedCount} flashcards from this document. Output ONLY JSON objects, one per line, like {"question": "Q", "answer": "A"}. Do not repeat questions.` },

                { inlineData: { mimeType: mimeType, data: fileData } }

            ]

        };

    } else {

        currentUserMessage = {

            role: "user",

            parts: [

                { text: `Generate ${requestedCount} more unique flashcards based on the document provided earlier. Output ONLY JSON objects, one per line, like {"question": "Q", "answer": "A"}. Do not repeat questions.` }

            ]

        };

    }

*// Add user message to chat history before sending*

    chatHistory.**push**(currentUserMessage);

*// Prepare the request body*

    const body = {

        contents: chatHistory,

        generationConfig: {

            temperature: 0.2,

            maxOutputTokens: 2048, *// Max tokens for the whole stream*

*// responseMimeType: "application/json", // Not used for streaming text*

        },

        safetySettings: [

            { category: "HARM\_CATEGORY\_DANGEROUS\_CONTENT", threshold: "BLOCK\_NONE" },

            { category: "HARM\_CATEGORY\_HARASSMENT", threshold: "BLOCK\_NONE" },

            { category: "HARM\_CATEGORY\_HATE\_SPEECH", threshold: "BLOCK\_NONE" },

            { category: "HARM\_CATEGORY\_SEXUALLY\_EXPLICIT", threshold: "BLOCK\_NONE" }

        ]

    };

    let accumulatedResponseText = ""; *// To build the full model response for history*

    let cardsGeneratedInThisStream = 0;

    try {

        const response = await **fetch**(endpoint, {

            method: "POST",

            headers: { "Content-Type": "application/json" },

            body: JSON.**stringify**(body)

        });

        if (!response.ok) {

*// Attempt to read error response, might not be SSE*

            const errorText = await response.**text**();

            let errorMessage = `API Error: ${response.status} ${response.statusText}`;

            try {

                const errorJson = JSON.**parse**(errorText);

                errorMessage += `. ${errorJson.error?.message || 'Unknown error'}`;

            } catch (e) { */\* Ignore parsing error \*/* }

            console.**error**("API Error Response Text:", errorText);

            throw new **Error**(errorMessage);

        }

        if (!response.body) {

            throw new **Error**("Response body is null");

        }

        const reader = response.body.**getReader**();

        const decoder = new **TextDecoder**();

        let buffer = "";

        let currentLine = "";

        while (true) {

            const { done, value } = await reader.**read**();

            if (done) {

                console.**log**("Stream finished.");

                break;

            }

            buffer += decoder.**decode**(value, { stream: true });

*// Process Server-Sent Events data*

            const lines = buffer.**split**('\n');

            buffer = lines.**pop**() || ""; *// Keep incomplete line*

            for (const line of lines) {

                if (line.**startsWith**('data:')) {

                    const dataContent = line.**substring**(5).**trim**();

                    try {

                        const parsedData = JSON.**parse**(dataContent);

                        if (parsedData.candidates && parsedData.candidates[0].content && parsedData.candidates[0].content.parts) {

                            const textPart = parsedData.candidates[0].content.parts[0].text;

                            if (textPart) {

                                accumulatedResponseText += textPart;

*// Process text part for potential JSON lines*

                                currentLine += textPart;

                                const jsonLines = currentLine.**split**('\n');

                                currentLine = jsonLines.**pop**() || ""; *// Keep incomplete JSON line*

                                for (const jsonLine of jsonLines) {

                                    if (jsonLine.**trim**()) {

                                        try {

                                            const card = JSON.**parse**(jsonLine.**trim**());

                                            if (card && typeof card.question === 'string' && typeof card.answer === 'string' && flashcards.length < MAX\_CARDS) {

                                                flashcards.**push**(card);

                                                cardsGeneratedInThisStream++;

                                                console.**log**(`Card received: ${flashcards.length}`);

*// Update UI immediately*

**updateCardDisplay**(); *// Update content*

**updateCardCounter**(); *// Update counter*

**updateControls**(); *// Update buttons*

*// Update history in localStorage incrementally or at the end*

                                                if (currentHistoryId !== null) {

**updateHistoryItem**(currentHistoryId, flashcards);

                                                }

                                            }

                                        } catch (e) {

                                            console.**warn**("Could not parse line as JSON card:", jsonLine.**trim**(), e);

                                        }

                                    }

                                }

                            }

                        }

                    } catch (e) {

                        console.**error**("Error parsing SSE data chunk:", dataContent, e);

                    }

                }

            }

             if (flashcards.length >= MAX\_CARDS) {

                 console.**log**("Max card limit reached during stream.");

*// Optionally close the stream early if possible/needed*

                 break;

             }

        }

*// Process any remaining buffer (likely empty for SSE)*

        if (currentLine.**trim**()) {

             try {

                 const card = JSON.**parse**(currentLine.**trim**());

                 if (card && typeof card.question === 'string' && typeof card.answer === 'string' && flashcards.length < MAX\_CARDS) {

                     flashcards.**push**(card);

                     cardsGeneratedInThisStream++;

                     console.**log**(`Card received (final): ${flashcards.length}`);

**updateCardDisplay**();

**updateCardCounter**();

**updateControls**();

                     if (currentHistoryId !== null) {

**updateHistoryItem**(currentHistoryId, flashcards);

                     }

                 }

             } catch (e) {

                 console.**warn**("Could not parse final line as JSON card:", currentLine.**trim**(), e);

             }

        }

*// Add the accumulated model response to chat history \*after\* the stream is done*

        if (accumulatedResponseText) {

            chatHistory.**push**({ role: "model", parts: [{ text: accumulatedResponseText }] });

        }

        if (cardsGeneratedInThisStream === 0 && flashcards.length === 0) {

             uploadStatus.textContent = 'API generated no valid flashcards from the text.';

        } else {

             uploadStatus.textContent = `Generated ${flashcards.length} flashcards.`;

        }

    } catch (error) {

        console.**error**("Streaming API Call failed:", error);

        uploadStatus.textContent = `Streaming Error: ${error.message}`;

*// Remove the user message we optimistically added if the call failed*

        if (chatHistory[chatHistory.length - 1] === currentUserMessage) {

            chatHistory.**pop**();

        }

    }

}

*// --- Flashcard Display and Navigation ---*

function **updateCardDisplay**() {

    if (flashcards.length === 0) {

        flashcardQuestion.textContent = 'No flashcards generated or loaded.';

        flashcardAnswer.textContent = '';

        flashcardSection.style.display = 'none';

    } else {

        flashcardSection.style.display = 'block';

        currentCardIndex = Math.**max**(0, Math.**min**(currentCardIndex, flashcards.length - 1));

        const card = flashcards[currentCardIndex];

        flashcardQuestion.textContent = card.question;

        flashcardAnswer.textContent = card.answer;

        flashcardElement.classList.**remove**('is-flipped');

    }

**updateControls**();

**updateCardCounter**();

}

function **flipCard**() {

    if (flashcards.length > 0) {

        flashcardElement.classList.**toggle**('is-flipped');

    }

}

function **showNextCard**() {

    if (currentCardIndex < flashcards.length - 1) {

        currentCardIndex++;

**updateCardDisplay**();

*// Generate more cards when approaching the end*

*// Check if we need file data (only needed for the \*first\* call in a session)*

        if (flashcards.length < MAX\_CARDS && currentCardIndex >= flashcards.length - 2 && !isGenerating) {

            isGenerating = true;

            generateBtn.disabled = true;

*// Subsequent calls don't need file data, rely on chat history*

**generateFlashcards**(null, null, null, CARDS\_PER\_BATCH) *// Pass nulls for file info*

                .**catch**(err => {

                    uploadStatus.textContent = `Error generating next batch: ${err.message}`;

                })

                .**finally**(() => {

                    isGenerating = false;

                    generateBtn.disabled = false;

**updateControls**();

                });

        }

    }

}

function **showPreviousCard**() {

    if (currentCardIndex > 0) {

        currentCardIndex--;

**updateCardDisplay**();

    }

}

function **updateControls**() {

    prevCardBtn.disabled = currentCardIndex === 0;

    nextCardBtn.disabled = currentCardIndex >= flashcards.length - 1 || flashcards.length === 0;

    generateBtn.disabled = isGenerating || !fileInput.files[0];

}

function **updateCardCounter**() {

    cardCounter.textContent = flashcards.length > 0 ?

        `${currentCardIndex + 1} / ${flashcards.length}` : '0 / 0';

    if (flashcards.length >= MAX\_CARDS) {

        cardCounter.textContent += ' (Max)';

    }

}

*// --- Settings ---*

function **toggleSettingsPopover**() {

    const isDisplayed = settingsPopover.style.display === 'block';

    settingsPopover.style.display = isDisplayed ? 'none' : 'block';

    if (!isDisplayed) {

        apiKeyInput.value = apiKey;

    }

}

function **saveSettings**() {

    apiKey = apiKeyInput.value.**trim**();

    if (apiKey) {

        localStorage.**setItem**('geminiApiKey', apiKey);

        console.**log**("API Key saved to localStorage.");

        uploadStatus.textContent = 'API Key saved.';

    } else {

        localStorage.**removeItem**('geminiApiKey');

        console.**log**("API Key removed from localStorage.");

        uploadStatus.textContent = 'API Key removed.';

    }

**setTimeout**(() => {

        if (uploadStatus.textContent.**includes**('API Key'))

            uploadStatus.textContent = '';

    }, 3000);

**toggleSettingsPopover**();

}

*// --- History Management ---*

function **loadHistory**() {

    try {

        const storedHistory = localStorage.**getItem**('flashcardHistory');

        history = storedHistory ? JSON.**parse**(storedHistory) : [];

        if (!Array.**isArray**(history)) {

            console.**error**("History in localStorage is not an array. Resetting.");

            history = [];

            localStorage.**removeItem**('flashcardHistory');

        }

    } catch (e) {

        console.**error**("Failed to parse history from localStorage:", e);

        history = [];

        localStorage.**removeItem**('flashcardHistory');

    }

**displayHistory**();

}

function **saveHistoryItem**(filename, cards) {

    if (!filename || !cards || cards.length === 0) return;

    const timestamp = Date.**now**();

    const historyItem = {

        id: timestamp,

        filename: filename,

        date: new **Date**().**toLocaleDateString**(),

        time: new **Date**().**toLocaleTimeString**(),

        cardCount: cards.length,

        cards: [...cards] *// Clone the array to avoid reference issues*

    };

*// Add to the beginning*

    history.**unshift**(historyItem);

*// Limit history to 10 items*

    if (history.length > 10) {

        history = history.**slice**(0, 10);

    }

*// Save to localStorage*

    try {

        localStorage.**setItem**('flashcardHistory', JSON.**stringify**(history));

    } catch (e) {

        console.**error**("Failed to save history to localStorage:", e);

**alert**("Could not save to history due to storage limits. Try clearing old items.");

    }

    currentHistoryId = timestamp;

**displayHistory**();

}

function **updateHistoryItem**(historyId, updatedCards) {

    if (!historyId || !updatedCards) return;

    const itemIndex = history.**findIndex**(item => item.id === historyId);

    if (itemIndex === -1) return;

    history[itemIndex].cards = [...updatedCards];

    history[itemIndex].cardCount = updatedCards.length;

    try {

        localStorage.**setItem**('flashcardHistory', JSON.**stringify**(history));

    } catch (e) {

        console.**error**("Failed to update history in localStorage:", e);

    }

**displayHistory**();

}

function **displayHistory**() {

    historyList.innerHTML = '';

    if (history.length === 0) {

        const emptyItem = document.**createElement**('li');

        emptyItem.className = 'history-empty';

        emptyItem.textContent = 'No history yet';

        historyList.**appendChild**(emptyItem);

        return;

    }

    history.**forEach**(item => {

        const li = document.**createElement**('li');

        li.className = 'history-item';

        if (item.id === currentHistoryId) {

            li.classList.**add**('active');

        }

        li.dataset.historyId = item.id;

        li.innerHTML = `

            <div class="history-title">${item.filename}</div>

            <div class="history-details">

                ${item.cardCount} cards · ${item.date}

            </div>

        `;

        historyList.**appendChild**(li);

    });

}

function **handleHistoryClick**(event) {

    if (isGenerating) return;

*// Find the li element that was clicked (or its parent if a child was clicked)*

    let target = event.target;

    while (target && target.tagName !== 'LI') {

        if (target === historyList) return; *// Clicked on empty space*

        target = target.parentElement;

    }

    if (!target || !target.dataset.historyId) return;

    const historyId = **parseInt**(target.dataset.historyId, 10);

    const selectedItem = history.**find**(item => item.id === historyId);

    if (selectedItem && selectedItem.cards) {

*// Load the flashcards from history*

        flashcards = [...selectedItem.cards];

        currentCardIndex = 0;

        currentHistoryId = historyId;

        chatHistory = []; *// Reset chat history when loading from saved history*

*// Update UI*

        uploadStatus.textContent = `Loaded ${selectedItem.cardCount} cards from "${selectedItem.filename}"`;

        flashcardSection.style.display = flashcards.length > 0 ? 'block' : 'none';

        fileInput.value = ''; *// Clear file input*

**updateCardDisplay**();

**displayHistory**(); *// Refresh history list to show active item*

    }

}

*// Apply CSS to history items*

function **applyHistoryStyles**() {

    const style = document.**createElement**('style');

    style.textContent = `

        #history-list {

            padding: 8px 0;

        }

        .history-item {

            padding: 12px;

            margin: 8px 0;

            border-radius: 8px;

            cursor: pointer;

            transition: background-color 0.2s;

            border: 1px solid var(--dusk-border);

        }

        .history-item:hover {

            background: rgba(127, 92, 255, 0.1);

        }

        .history-item.active {

            background: rgba(0, 255, 231, 0.1);

            border-color: var(--dusk-accent);

        }

        .history-title {

            font-weight: 500;

            color: var(--dusk-accent);

            margin-bottom: 4px;

            overflow: hidden;

            text-overflow: ellipsis;

            white-space: nowrap;

        }

        .history-details {

            font-size: 0.8rem;

            color: var(--dusk-muted);

        }

        .history-empty {

            text-align: center;

            color: var(--dusk-muted);

            font-style: italic;

            padding: 20px 0;

        }

    `;

    document.head.**appendChild**(style);

}

*// --- Keyboard Controls ---*

document.**addEventListener**('keydown', e => {

  if (settingsPopover.style.display === 'block') return;

  if (e.key === 'ArrowRight' || e.key === 'd') **showNextCard**();

  if (e.key === 'ArrowLeft' || e.key === 'a') **showPreviousCard**();

  if (e.key === ' ' || e.key === 'Enter') **flipCard**();

});

*// --- Swipe Animations & Carousel ---*

let startX = null;

flashcardElement.**addEventListener**('touchstart', e => {

  if (e.touches.length === 1) startX = e.touches[0].clientX;

});

flashcardElement.**addEventListener**('touchend', e => {

  if (startX === null) return;

  let dx = e.changedTouches[0].clientX - startX;

  if (dx > 60) **showPreviousCard**();

  else if (dx < -60) **showNextCard**();

  startX = null;

});

*// Animate card slide*

function **animateCard**(direction) {

  flashcardElement.classList.**remove**('slide-left', 'slide-right');

  void flashcardElement.offsetWidth;

  flashcardElement.classList.**add**(direction === 'left' ? 'slide-left' : 'slide-right');

**setTimeout**(() => flashcardElement.classList.**remove**('slide-left', 'slide-right'), 400);

}

const origShowNext = showNextCard;

**showNextCard** = function() {

**animateCard**('left');

**setTimeout**(origShowNext, 120);

};

const origShowPrev = showPreviousCard;

**showPreviousCard** = function() {

**animateCard**('right');

**setTimeout**(origShowPrev, 120);

};

*// --- Initialize ---*

**initialize**();

**applyHistoryStyles**();

*Style.css*

*/\* Basic Reset \*/*

\* {

    margin: 0;

    padding: 0;

    box-sizing: border-box;

}

:root {

    --dusk-bg: #181a20;

    --dusk-panel: #23242b;

    --dusk-glass: **rgba**(36, 38, 46, 0.85);

    --dusk-glass-blur: 18px;

    --dusk-accent: #00ffe7; */\* Teal \*/*

    --dusk-accent2: #7f5cff; */\* Purple \*/*

    --dusk-text: #e6e6f0;

    --dusk-muted: #7a7a8c;

    --dusk-border: **rgba**(127, 92, 255, 0.25); */\* Slightly more visible border \*/*

    --dusk-shadow: 0 8px 32px 0 **rgba**(0,0,0,0.55); */\* Deeper shadow \*/*

    --animation-speed-fast: 0.3s;

    --animation-speed-medium: 0.7s;

    --animation-speed-slow: 1.2s;

    --easing-curve: **cubic-bezier**(.77,0,.18,1);

}

@keyframes backgroundPan {

    0% { background-position: 0% 50%; }

    50% { background-position: 100% 50%; }

    100% { background-position: 0% 50%; }

}

@keyframes subtleGlow {

    0%, 100% { text-shadow: 0 0 8px **var**(--dusk-accent2), 0 0 16px **rgba**(127, 92, 255, 0.5); }

    50% { text-shadow: 0 0 12px **var**(--dusk-accent2), 0 0 24px **rgba**(127, 92, 255, 0.7); }

}

@keyframes pulseBorder {

     0% { border-color: **var**(--dusk-border); box-shadow: **var**(--dusk-shadow); }

     50% { border-color: **var**(--dusk-accent2); box-shadow: 0 8px 40px 0 **rgba**(127, 92, 255, 0.4); }

     100% { border-color: **var**(--dusk-border); box-shadow: **var**(--dusk-shadow); }

}

body {

    font-family: 'Inter', 'Segoe UI', Arial, sans-serif;

*/\* Animated Gradient Background \*/*

    background: **linear-gradient**(120deg, **var**(--dusk-bg) 0%, **var**(--dusk-panel) 50%, **var**(--dusk-bg) 100%);

    background-size: 200% 200%; */\* Increase size for panning \*/*

    animation: backgroundPan 15s ease infinite; */\* Slow pan animation \*/*

    color: **var**(--dusk-text);

    display: flex;

    min-height: 100vh;

    overflow: hidden;

    letter-spacing: 0.01em;

}

*/\* Glassmorphism Effect \*/*

.glassmorphism {

    background: **var**(--dusk-glass);

    backdrop-filter: **blur**(**var**(--dusk-glass-blur));

    -webkit-backdrop-filter: **blur**(**var**(--dusk-glass-blur));

    border-radius: 18px;

    border: 1.5px solid **var**(--dusk-border);

    box-shadow: **var**(--dusk-shadow);

    padding: 24px;

    transition: box-shadow **var**(--animation-speed-fast) ease, border **var**(--animation-speed-fast) ease, transform **var**(--animation-speed-fast) ease;

}

.glassmorphism:hover {

*/\* Subtle lift on hover for elements using glassmorphism directly \*/*

*/\* transform: translateY(-3px); \*/*

    box-shadow: 0 12px 40px 0 **rgba**(0,0,0,0.65);

}

*/\* Layout \*/*

.sidebar {

    width: 270px;

    background: **var**(--dusk-panel);

    padding: 28px 18px 18px 18px;

    display: flex;

    flex-direction: column;

    height: 100vh;

    border-right: 1.5px solid **var**(--dusk-border);

    box-shadow: 4px 0 28px 0 **rgba**(0,0,0,0.25);

    z-index: 2;

    position: relative;

    animation: slideInLeft **var**(--animation-speed-medium) **var**(--easing-curve) 1;

    transition: box-shadow **var**(--animation-speed-fast) ease;

}

.sidebar:hover {

     box-shadow: 6px 0 35px 0 **rgba**(0,0,0,0.35);

}

.sidebar h2 {

    margin-bottom: 24px;

    text-align: center;

    color: **var**(--dusk-accent);

    font-weight: 700;

    letter-spacing: 0.08em;

    font-size: 1.2rem;

    text-shadow: 0 0 8px **var**(--dusk-accent);

    animation: fadeIn **var**(--animation-speed-slow) ease 1;

}

#history-list {

    list-style: none;

    flex-grow: 1;

    overflow-y: auto;

    margin-bottom: 18px;

    scrollbar-width: thin;

    scrollbar-color: **var**(--dusk-accent2) **var**(--dusk-panel);

}

#history-list::-webkit-scrollbar {

    width: 6px;

}

#history-list::-webkit-scrollbar-thumb {

    background: **var**(--dusk-accent2);

    border-radius: 4px;

}

#history-list li {

    padding: 9px 14px; */\* Slightly larger padding \*/*

    margin-bottom: 8px; */\* Slightly more spacing \*/*

    border-radius: 8px; */\* Smoother radius \*/*

    border: 1px solid transparent;

    transition: background-color **var**(--animation-speed-fast) ease,

                border-color **var**(--animation-speed-fast) ease,

                transform **var**(--animation-speed-fast) ease,

                box-shadow **var**(--animation-speed-fast) ease;

    font-size: 0.9rem;

    white-space: nowrap;

    overflow: hidden;

    text-overflow: ellipsis;

    cursor: pointer;

    position: relative; */\* For potential pseudo-elements \*/*

}

#history-list li:hover {

    background-color: **rgba**(127, 92, 255, 0.15); */\* Slightly stronger hover \*/*

    border-color: **var**(--dusk-accent2);

    transform: **translateX**(5px) **scale**(1.02); */\* Add subtle scale \*/*

    box-shadow: 0 2px 8px **rgba**(127, 92, 255, 0.2);

}

#history-list li.active {

    background: **linear-gradient**(90deg, **var**(--dusk-accent2) 0%, **var**(--dusk-accent) 100%);

    color: **var**(--dusk-panel);

    font-weight: 600;

    border-color: transparent; */\* Remove border when active \*/*

    box-shadow: 0 4px 15px **rgba**(127, 92, 255, 0.4);

    transform: **translateX**(5px) **scale**(1.03); */\* Slightly more pronounced active state \*/*

}

.settings-button {

    margin-top: auto;

    padding: 12px 0;

    background: **linear-gradient**(90deg, **var**(--dusk-accent2) 0%, **var**(--dusk-accent) 100%);

    background-size: 200% 100%; */\* For hover animation \*/*

    border: none;

    color: #fff;

    border-radius: 10px;

    font-weight: 600;

    font-size: 1rem;

    cursor: pointer;

    box-shadow: 0 4px 18px 0 **rgba**(0,0,0,0.25);

    transition: background-position **var**(--animation-speed-fast) ease,

                transform **var**(--animation-speed-fast) ease,

                box-shadow **var**(--animation-speed-fast) ease;

}

.settings-button:hover {

    background-position: right center; */\* Animate gradient \*/*

    transform: **translateY**(-2px) **scale**(1.03); */\* Lift effect \*/*

    box-shadow: 0 6px 25px 0 **rgba**(0, 255, 231, 0.3); */\* Accent glow \*/*

}

.settings-button:active {

    transform: **scale**(0.98);

    transition-duration: 0.1s;

}

.main-content {

    flex-grow: 1;

    padding: 48px 20px 20px 20px; */\* Add side padding \*/*

    overflow-y: auto;

    height: 100vh;

    display: flex;

    flex-direction: column;

    align-items: center;

    background: none;

    position: relative;

    scrollbar-width: thin;

    scrollbar-color: **var**(--dusk-accent2) **var**(--dusk-panel);

}

.main-content::-webkit-scrollbar {

    width: 8px;

}

.main-content::-webkit-scrollbar-thumb {

    background: **var**(--dusk-accent2);

    border-radius: 4px;

}

.main-content h1 {

    margin-bottom: 36px;

    text-align: center;

    font-size: 2.4rem; */\* Slightly larger \*/*

    font-weight: 800;

    color: **var**(--dusk-text); */\* Base text color \*/*

    letter-spacing: 0.12em;

*/\* Use animation for the glow \*/*

    animation: fadeIn **var**(--animation-speed-slow) **var**(--easing-curve) 1,

               subtleGlow 4s ease-in-out infinite; */\* Add subtle glow animation \*/*

}

.input-section {

    margin-bottom: 36px;

    text-align: center;

    width: 90%;

    max-width: 650px;

*/\* Apply glassmorphism \*/*

    background: **var**(--dusk-glass);

    backdrop-filter: **blur**(**var**(--dusk-glass-blur));

    -webkit-backdrop-filter: **blur**(**var**(--dusk-glass-blur));

    border-radius: 18px;

    border: 1.5px solid **var**(--dusk-border);

    box-shadow: **var**(--dusk-shadow);

    padding: 28px; */\* Increased padding \*/*

    animation: fadeIn **var**(--animation-speed-slow) **var**(--easing-curve) 1,

               pulseBorder 6s ease-in-out infinite alternate; */\* Add pulsing border \*/*

    transition: box-shadow **var**(--animation-speed-fast) ease, border **var**(--animation-speed-fast) ease;

}

.input-section:hover {

     border-color: **var**(--dusk-accent); */\* Change border color on hover \*/*

     box-shadow: 0 10px 40px 0 **rgba**(0, 255, 231, 0.2); */\* Accent glow on hover \*/*

}

.input-section input[type="file"] {

    display: block;

    margin: 18px auto 12px auto;

    color: **var**(--dusk-text);

    background: **var**(--dusk-panel);

    border-radius: 8px;

    border: 1.5px solid **var**(--dusk-border); */\* Match border style \*/*

    padding: 10px 14px; */\* More padding \*/*

    font-size: 1rem;

    cursor: pointer;

    transition: border-color **var**(--animation-speed-fast) ease, box-shadow **var**(--animation-speed-fast) ease;

}

.input-section input[type="file"]:hover {

    border-color: **var**(--dusk-accent);

}

*/\* Basic button styling \*/*

button {

    padding: 12px 24px; */\* More horizontal padding \*/*

    background: **linear-gradient**(90deg, **var**(--dusk-accent2) 0%, **var**(--dusk-accent) 100%);

    background-size: 200% 100%; */\* For hover animation \*/*

    border: none;

    color: #fff;

    border-radius: 10px;

    font-weight: 600;

    font-size: 1rem;

    cursor: pointer;

    margin: 8px 6px;

    box-shadow: 0 4px 18px 0 **rgba**(0,0,0,0.25);

    transition: background-position **var**(--animation-speed-fast) ease,

                transform **var**(--animation-speed-fast) ease,

                box-shadow **var**(--animation-speed-fast) ease;

}

button:hover:not(:disabled) {

    background-position: right center; */\* Animate gradient \*/*

    transform: **translateY**(-2px) **scale**(1.03); */\* Lift effect \*/*

    box-shadow: 0 6px 25px 0 **rgba**(0, 255, 231, 0.3); */\* Accent glow \*/*

}

button:active:not(:disabled) {

    transform: **scale**(0.98);

    transition-duration: 0.1s;

}

button:disabled {

    background: **var**(--dusk-muted);

    cursor: not-allowed;

    color: #aaa;

    box-shadow: none;

    transform: none;

}

*/\* Flashcard Styles \*/*

.flashcard-container {

    width: 90%;

    max-width: 540px;

    text-align: center;

    margin-top: 18px;

    animation: fadeIn **var**(--animation-speed-slow) **var**(--easing-curve) 1;

}

.flashcard {

    min-height: 270px;

    perspective: 1500px; */\* Increased perspective \*/*

    background: none;

    border: none;

    box-shadow: none;

    margin-bottom: 24px;

    cursor: pointer;

    position: relative;

    transition: transform 0.3s **var**(--easing-curve);

}

.flashcard:hover .flashcard-inner:not(.is-flipped) { */\* Only lift when not flipped \*/*

     transform: **translateY**(-8px) **rotateX**(5deg); */\* Add subtle tilt on hover \*/*

     box-shadow: 0 15px 45px 0 **rgba**(0,0,0,0.6);

}

.flashcard-inner {

    position: relative;

    width: 100%;

    height: 270px;

    text-align: center;

    transition: transform **var**(--animation-speed-medium) **var**(--easing-curve), box-shadow **var**(--animation-speed-medium) ease; */\* Smooth transition \*/*

    transform-style: preserve-3d;

    border-radius: 18px;

    box-shadow: **var**(--dusk-shadow);

}

.flashcard.is-flipped .flashcard-inner {

    transform: **rotateY**(180deg);

    box-shadow: **var**(--dusk-shadow); */\* Maintain shadow when flipped \*/*

}

.flashcard-front, .flashcard-back {

    position: absolute;

    width: 100%;

    height: 100%;

    -webkit-backface-visibility: hidden;

    backface-visibility: hidden;

    display: flex;

    align-items: center;

    justify-content: center;

    padding: 32px;

    border-radius: 18px;

    font-size: 1.3rem; */\* Slightly larger text \*/*

    font-weight: 600;

    color: **var**(--dusk-text);

    background: **var**(--dusk-glass); */\* Base glass background \*/*

    border: 1.5px solid **var**(--dusk-border);

    box-shadow: inset 0 0 20px **rgba**(0,0,0,0.3); */\* Inner shadow for depth \*/*

    letter-spacing: 0.04em;

    transition: background **var**(--animation-speed-fast) ease, border-color **var**(--animation-speed-fast) ease;

    overflow: hidden; */\* Ensure content doesn't overflow \*/*

    line-height: 1.5; */\* Improve readability \*/*

}

.flashcard-front {

     background: **linear-gradient**(135deg, **var**(--dusk-glass) 70%, **rgba**(127, 92, 255, 0.3)); */\* Purple accent gradient \*/*

     border-color: **rgba**(127, 92, 255, 0.4);

}

.flashcard-back {

    transform: **rotateY**(180deg);

    background: **linear-gradient**(135deg, **var**(--dusk-glass) 70%, **rgba**(0, 255, 231, 0.3)); */\* Teal accent gradient \*/*

    border-color: **rgba**(0, 255, 231, 0.4);

    color: **var**(--dusk-text);

}

.flashcard-controls {

    display: flex;

    justify-content: space-between;

    align-items: center;

    margin-top: 18px;

    gap: 12px;

    animation: fadeIn 1.5s ease 0.5s 1 backwards; */\* Delayed fade in \*/*

}

#card-counter {

    font-weight: bold;

    color: **var**(--dusk-accent);

    font-size: 1.1rem;

    text-shadow: 0 0 5px **var**(--dusk-accent);

}

*/\* Settings Popover \*/*

@keyframes popoverEnter {

    from { opacity: 0; transform: **translate**(-50%, -50%) **scale**(0.85); }

    to { opacity: 1; transform: **translate**(-50%, -50%) **scale**(1); }

}

.popover {

    position: fixed;

    top: 50%;

    left: 50%;

*/\* transform: translate(-50%, -50%); \*/* */\* Handled by animation \*/*

    width: 92%;

    max-width: 420px;

    z-index: 1000;

    padding: 36px 32px 28px 32px;

    border-radius: 18px;

    border: 1.5px solid **var**(--dusk-accent2);

    box-shadow: 0 12px 50px 0 **rgba**(0,0,0,0.6); */\* Stronger shadow for popover \*/*

    background: **var**(--dusk-glass);

    backdrop-filter: **blur**(22px); */\* More blur for popover \*/*

    -webkit-backdrop-filter: **blur**(22px);

    animation: popoverEnter 0.4s **var**(--easing-curve) 1 forwards;

}

.popover h2 {

    margin-bottom: 24px; */\* More space \*/*

    text-align: center;

    color: **var**(--dusk-accent2);

    font-weight: 700;

    text-shadow: 0 0 10px **var**(--dusk-accent2);

}

.popover label {

    display: block;

    margin-bottom: 8px; */\* More space \*/*

    color: **var**(--dusk-accent);

    font-weight: 600; */\* Bolder label \*/*

    font-size: 0.95rem;

}

.popover input[type="password"] {

    width: 100%;

    padding: 14px; */\* More padding \*/*

    margin-bottom: 22px; */\* More space \*/*

    border-radius: 10px; */\* Match button radius \*/*

    border: 1.5px solid **var**(--dusk-accent2);

    background: **var**(--dusk-panel);

    color: **var**(--dusk-text);

    font-size: 1rem;

    transition: border-color **var**(--animation-speed-fast) ease, box-shadow **var**(--animation-speed-fast) ease;

}

.popover button {

    margin-right: 10px;

    margin-top: 10px; */\* Add top margin for spacing \*/*

}

*/\* Simple Animations (More can be added) \*/*

@keyframes fadeIn {

    from { opacity: 0; }

    to { opacity: 1; }

}

@keyframes slideInLeft {

    from { transform: **translateX**(-100px); opacity: 0; } */\* Start further left \*/*

    to { transform: **translateX**(0); opacity: 1; }

}

*/\* Futuristic glowing accent border for focus \*/*

input:focus, button:focus, select:focus, textarea:focus { */\* Apply to more elements \*/*

    outline: none; */\* Remove default outline \*/*

    border-color: **var**(--dusk-accent) **!important**; */\* Force border color \*/*

    box-shadow: 0 0 0 3px **rgba**(0, 255, 231, 0.3), */\* Outer glow \*/*

                0 0 10px **rgba**(0, 255, 231, 0.2), */\* Inner subtle glow \*/*

                inset 0 0 5px **rgba**(0, 255, 231, 0.1); */\* Inset glow \*/*

}

*/\* Specific focus for password input in popover \*/*

.popover input[type="password"]:focus {

     border-color: **var**(--dusk-accent) **!important**;

     box-shadow: 0 0 0 3px **rgba**(0, 255, 231, 0.3),

                 0 0 10px **rgba**(0, 255, 231, 0.2),

                 inset 0 0 5px **rgba**(0, 255, 231, 0.1);

}

*/\* Add a subtle loading/processing indicator possibility \*/*

.is-loading::after {

    content: '';

    position: absolute;

    top: 50%;

    left: 50%;

    width: 20px;

    height: 20px;

    margin-top: -10px;

    margin-left: -10px;

    border: 3px solid **rgba**(255, 255, 255, 0.3);

    border-top-color: **var**(--dusk-accent);

    border-radius: 50%;

    animation: spin 1s linear infinite;

}

@keyframes spin {

    to { transform: **rotate**(360deg); }

}

**Appendix-III Datasheets**

**Information Regarding Student(s)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Student name | Email id | Permanent Address | Phone Number | Placement Details | Photograph |
| Mohamed Faiz V S | 22btrcn173@jainuniverisity.ac.in |  | 9383420741 |  |  |
| Shashank .S | 22btrcn257@jainuniverisity.ac.in |  | 7760269456 |  |  |
| Karthik Jadhav | 22btrcn135@jainuniverisity.ac.in |  | 9074525933 |  |  |
| K S Sirisha | 22btrcn129@jainuniverisity.ac.in |  | 9391239740 |  |  |
| Y Chandana | 22btrcn330@jainuniverisity.ac.in |  | 6305161833 |  |  |

**Photograph Along With Guide**

