A REPORT ON

"AI News Aggregator"

SUBMITTED IN PARTIAL FULFILLMENT FOR AWARD DEGREE OF

BCA – MEDIA & INFORMATION TECHNOLOGY
(MINOR PROJECT)

UNIV ROLL NO. 23BCAM27
IInd Year (SEMESTER-4)

UNDER THE GUIDANCE OF
MR. NITISH KUMAR SIR

CENTRE FOR MEDIA & ENTERTAINMENT

Submitted To:
Mr. Nitish Kumar

Submitted By:
Soumyadyuti Dey

CONTENTS

.

CANDIDATE'S DECLARATION

Certified that this project report entitled **"AI News Aggregator"** submitted by **Soumyadyuti Dey (Your Roll No.)**, student of BCA-MIT, The NorthCap University, Gurgaon in the partial fulfillment of the requirements for the award of the degree of **Bachelor of Computer Applications (MIT)** is a record of the candidate's own work carried out during the project period.

This report has not been submitted to any other university or institution for the award of any degree.

(Soumyadyuti Dey)
Date: 28/04/25

SUPERVISOR'S CERTIFICATE

This is to certify that the project entitled **"AI News Aggregator"** is the bona fide work of **Soumyadyuti Dey**, submitted in partial fulfillment of the requirements for the degree of **Bachelor of Computer Applications (Media & Information Technology)** at The NorthCap University, under my guidance and supervision during **Jan'25 – May'25**.

Supervisor Name: (Mr. Nitish Kumar)
Date: 28/04/25

ABSTRACT

The AI News Aggregator is a web-based application built to personalize and simplify the news consumption experience. Leveraging technologies such as Flask, HTML/CSS, JavaScript, and the NewsAPI, the application delivers a dynamic and responsive interface where users can filter news based on their chosen topics. The system utilizes keyword-based preference matching and stores reading history and saved articles. Additionally, a Text-to-Speech (TTS) system allows users to listen to articles, creating a hands-free, accessible experience.

This project integrates essential full-stack web development concepts, API handling, user-centric design, and personalized content delivery. It serves as a robust example of a scalable and extensible information aggregator with real-world applications.

ACKNOWLEDGEMENT

I sincerely thank my project supervisor **Mr. Nitish Kumar** for his constant support and insightful feedback throughout the course of this project. I am also grateful to **The NorthCap University** for providing a conducive learning environment.

I extend heartfelt thanks to my parents and friends for their encouragement, and to everyone who helped me stay motivated and focused during this journey.

Soumyadyuti Dey
Date: 28/04/25

CHAPTER 1: About the Organization

1.1 Organization Structure

The project is part of a digital innovation initiative within a tech-focused organization that prioritizes artificial intelligence, automation, and modern UI/UX development.

1.2 Core Area

Core areas include Web Development, Artificial Intelligence, and API Integration.

1.3 Specializations Available

- AI & Automation
- Web Application Development
- Natural Language Processing (NLP)
- UI/UX and Front-End Engineering

1.4 Specialization Opted

- Personalized AI News Aggregator
- Front-end + Back-end Integration
- NewsAPI and Text-to-Speech APIs
- Local/Cloud Storage for Preferences

CHAPTER 2: System Overview

2.1 Introduction

The AI News Aggregator is designed to streamline news consumption by delivering articles

that align with user-specified interests. It offers keyword-based filtering, article history, a TTS engine, and interactive UI for a comprehensive reading experience.

2.2 Features of the System

- Personalized News Feeds
- Keyword Preference System
- Text-to-Speech Functionality
- Article Save & History Tracking
- Responsive and Modern Web Design

2.3 Problem Definition

Existing news platforms often overwhelm users with irrelevant articles. This system solves the problem by curating content through keyword preferences and enhancing accessibility with a TTS engine.

CHAPTER 3: Software and Hardware Requirements

3.1 Software Requirements

- Python 3.x
- Flask
- HTML/CSS/JavaScript
- NewsAPI (External API)
- gTTS for TTS feature

3.2 Hardware Requirements

- Minimum 4GB RAM
- Internet Connection
- Modern Web Browser

3.3 Operating System Environment

- Windows / macOS / Linux
- Python must be installed and configured

- Compatible with mobile and desktop browsers

CHAPTER 4: System Analysis

4.1 Software Requirement Analysis

The system utilizes:

- API Integration (NewsAPI)
- RESTful request handling with Flask
- Text-to-Speech library for reading news aloud
- Local storage or database for saving preferences

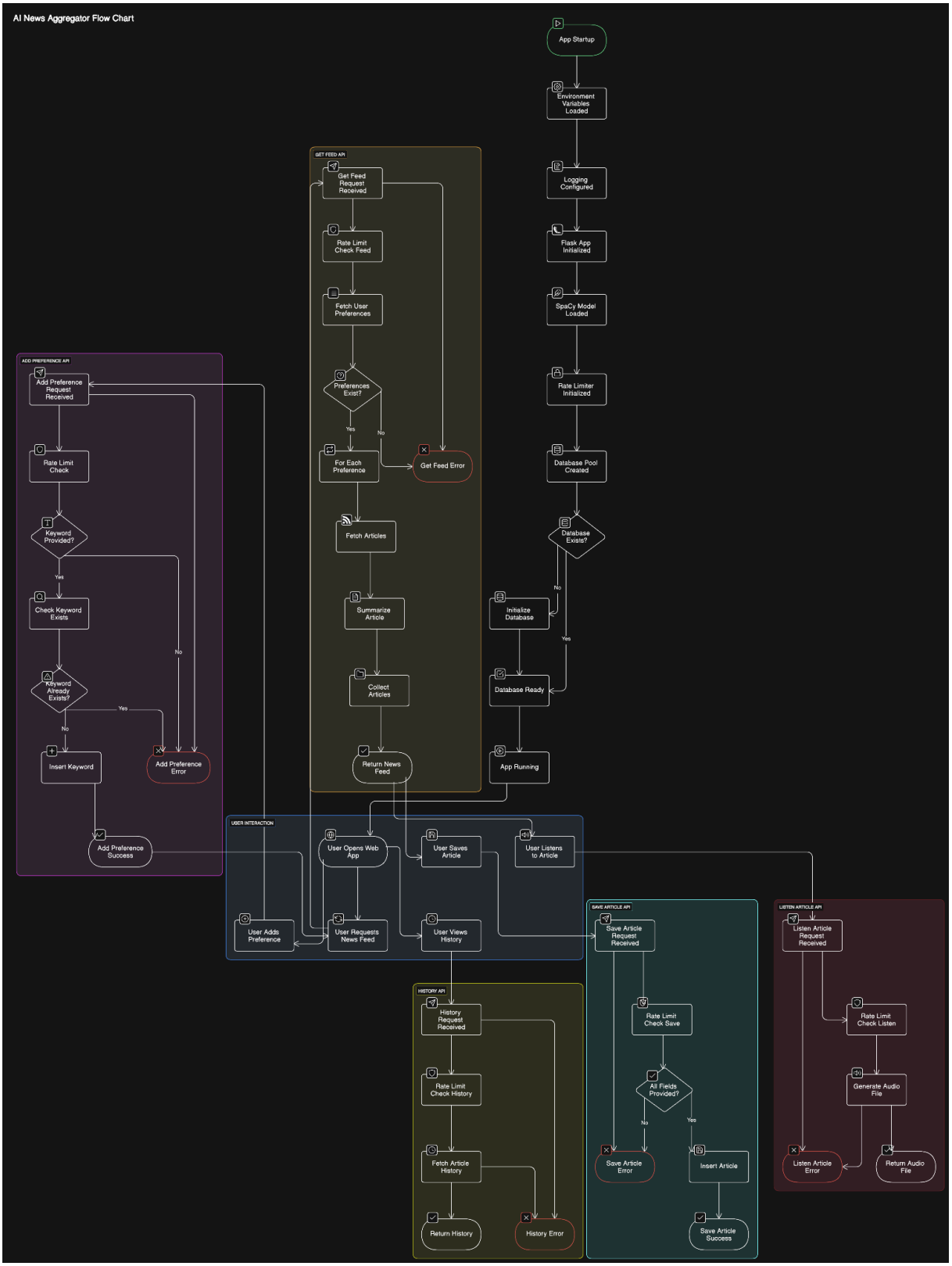
4.2 Feasibility Study

- **Technical**: Feasible using open-source tools
- **Economic**: Cost-effective, low-resource usage
- **Operational**: Simple UI ensures ease of use and adaptability

---

CHAPTER 5: System Design

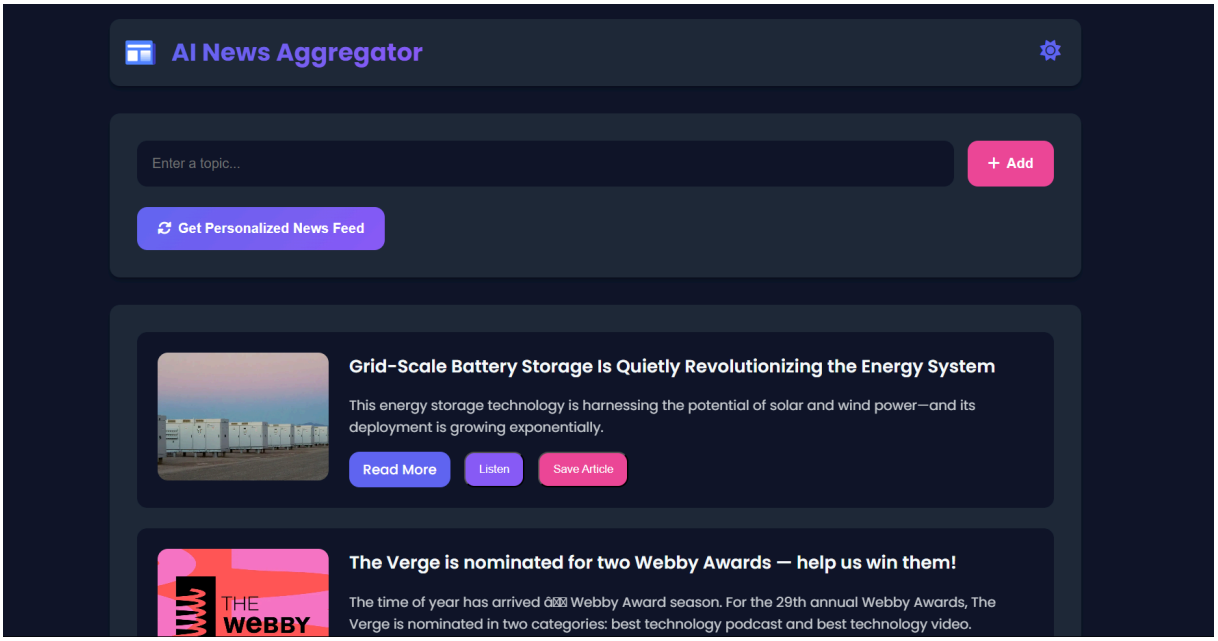
- **Architecture Diagram**: Frontend, Backend, API Layer



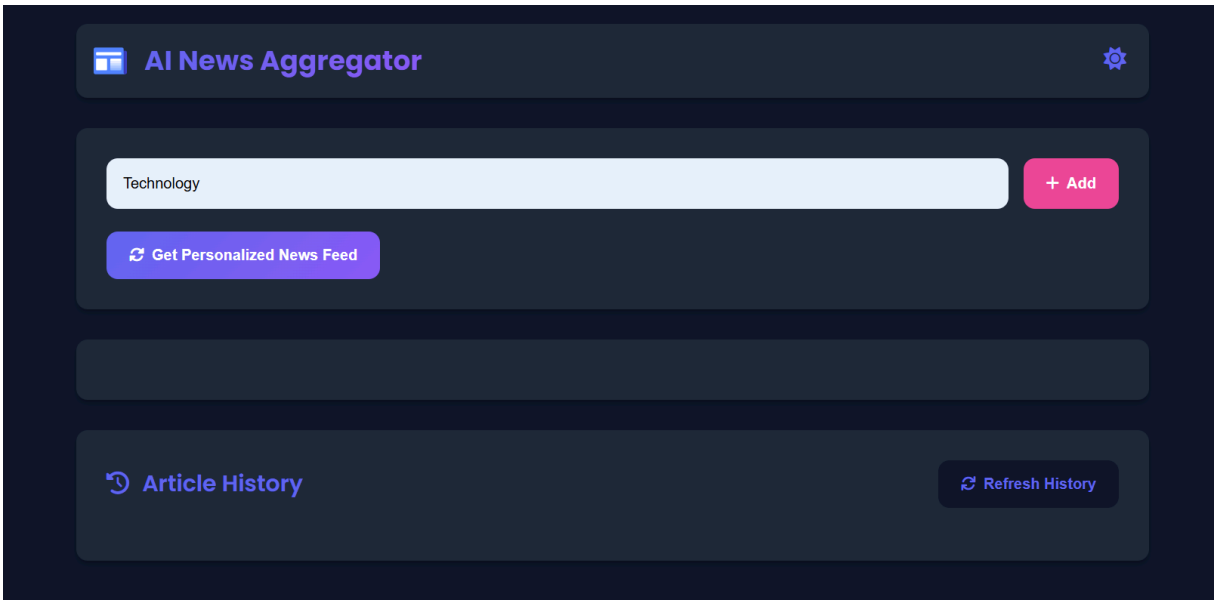
CHAPTER 6: Detailed Design

6.1 Snapshots and Interpretations

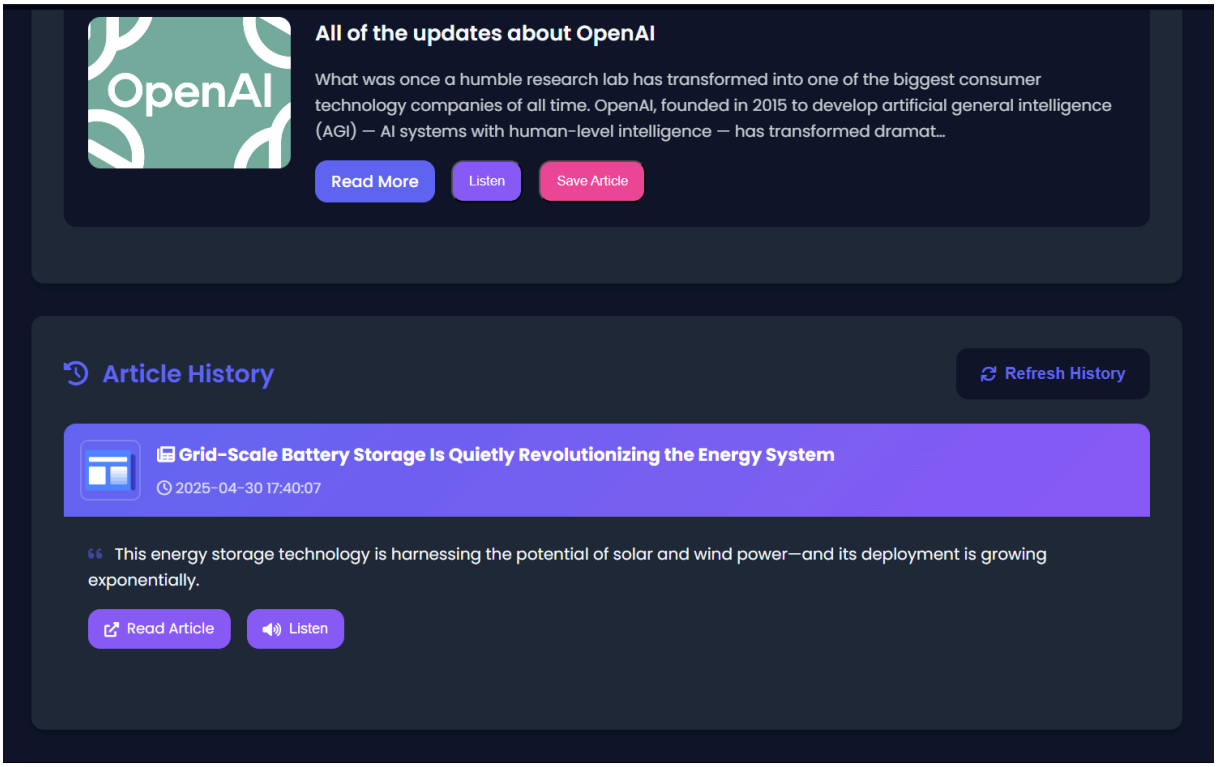
- **Home Page**: Shows top headlines and personalized feed



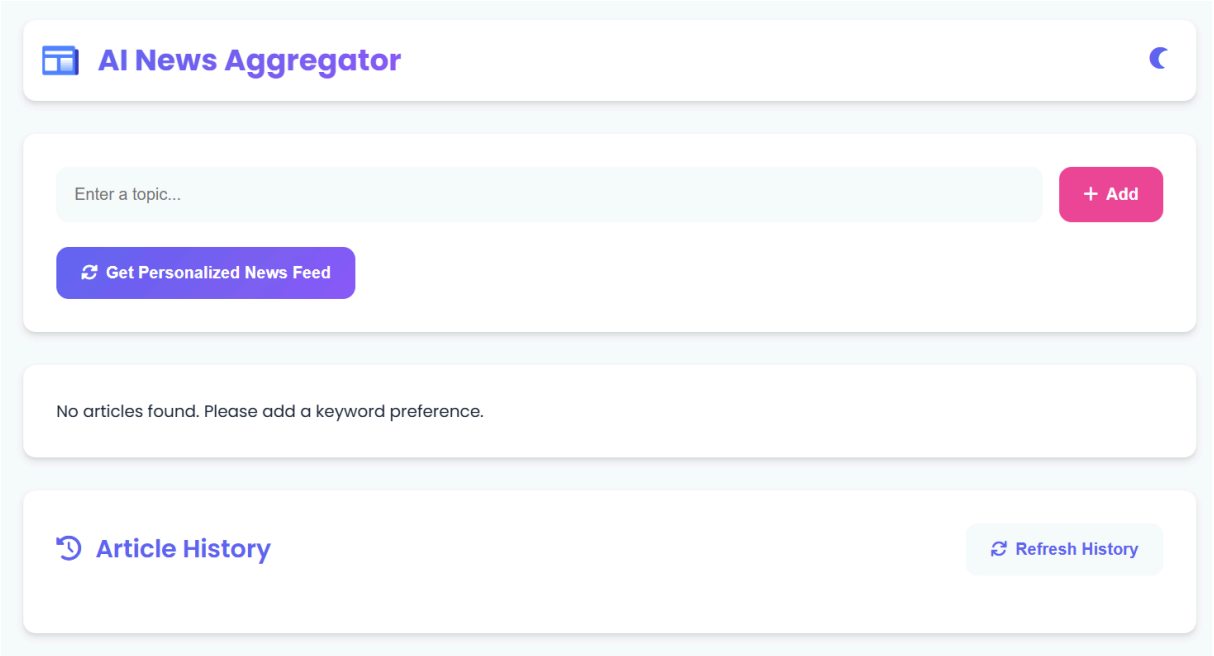
- **Add Keyword Section**: User inputs interests (e.g., "AI", "Technology")



- **Saved Articles Section**: Access to saved stories



- Dark & Light Mode



CHAPTER 7: Code Snippets

- GitHub Link (Click Here)

Python :

```
import
```

```python
import sqlite3
import requests
import spacy
import pyttsx3
from gtts import gTTS
from flask import Flask, render_template, request, jsonify,
send_file
from datetime import datetime, timedelta
from functools import lru_cache
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address
import threading
from queue import Queue
import logging
from dotenv import load_dotenv

# Load environment variables
load_dotenv()

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Initialize Flask app and spaCy model
app = Flask(__name__)
nlp = spacy.load("en_core_web_sm")

# Initialize rate limiter
limiter = Limiter(
    app=app,
    key_func=get_remote_address,
    default_limits=["200 per day", "50 per hour"]
)

# Get API key from environment variable
NEWS_API_KEY = os.getenv("NEWS_API_KEY",
"0b631bbba8cf44a7b0643628fb4da859")

# Database connection pool
class DatabasePool:
    def __init__(self, max_connections=5):
        self.max_connections = max_connections
        self.connections = Queue(maxsize=max_connections)
        self.lock = threading.Lock()

        # Ensure database directory exists
```

```
os.makedirs(os.path.dirname("ai_news_aggregator/news_aggregator.db"), exist_ok=True)

        for _ in range(max_connections):
            conn = sqlite3.connect("ai_news_aggregator/news_aggregator.db", check_same_thread=False)
            conn.row_factory = sqlite3.Row
            self.connections.put(conn)

        # Initialize database
        self.initialize_database()

    def get_connection(self):
        return self.connections.get()

    def return_connection(self, conn):
        self.connections.put(conn)

    def initialize_database(self):
        try:
            conn = self.get_connection()
            cursor = conn.cursor()

            # Check if article_history table exists and has the keyword column
            cursor.execute("PRAGMA table_info(article_history)")
            columns = [column[1] for column in cursor.fetchall()]

            if 'keyword' not in columns:
                # Add the keyword column if it doesn't exist
                cursor.execute('ALTER TABLE article_history ADD COLUMN keyword TEXT')
                logger.info("Added keyword column to article_history table")

            # Check if user_preferences table exists
            cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='user_preferences'")
            if not cursor.fetchone():
                # Create user_preferences table if it doesn't exist
                cursor.execute('''
                    CREATE TABLE user_preferences (
                        id INTEGER PRIMARY KEY
```

```
AUTOINCREMENT,
                        keyword TEXT NOT NULL UNIQUE,
                        created_at DATETIME DEFAULT
CURRENT_TIMESTAMP
                    )
                ''')
                logger.info("Created user_preferences
table")

            # Create or update indexes
            cursor.execute('''
                CREATE INDEX IF NOT EXISTS
idx_article_history_timestamp
                ON article_history(timestamp)
            ''')

            conn.commit()
            logger.info("Database schema updated
successfully")
        except sqlite3.Error as e:
            logger.error(f"Error updating database schema:
{e}")
            raise
        finally:
            self.return_connection(conn)

db_pool = DatabasePool()

def init_db():
    try:
        conn = db_pool.get_connection()
        cursor = conn.cursor()

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS user_preferences (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                keyword TEXT NOT NULL UNIQUE,
                created_at DATETIME DEFAULT
CURRENT_TIMESTAMP
            )
        ''')

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS article_history (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                title TEXT NOT NULL,
                url TEXT NOT NULL,
```

```
                summary TEXT,
                timestamp DATETIME DEFAULT
CURRENT_TIMESTAMP,
                keyword TEXT
            )
        ''')

        cursor.execute('''
            CREATE INDEX IF NOT EXISTS
idx_article_history_timestamp
            ON article_history(timestamp)
        ''')

        conn.commit()
        db_pool.return_connection(conn)
        logger.info("Database initialized successfully")
    except sqlite3.Error as e:
        logger.error(f"Error initializing database: {e}")
        raise

@lru_cache(maxsize=100)
def execute_query(query, params=()):
    try:
        conn = db_pool.get_connection()
        cursor = conn.cursor()
        cursor.execute(query, params)
        conn.commit()
        db_pool.return_connection(conn)
        return True
    except sqlite3.Error as e:
        logger.error(f"Database error in execute_query:
{e}")
        raise

@lru_cache(maxsize=100)
def fetch_data(query, params=()):
    try:
        conn = db_pool.get_connection()
        cursor = conn.cursor()
        cursor.execute(query, params)
        result = cursor.fetchall()
        db_pool.return_connection(conn)
        return result
    except sqlite3.Error as e:
        logger.error(f"Database error in fetch_data: {e}")
        raise
```

```python
# Cache API responses for 5 minutes
@lru_cache(maxsize=100)
def fetch_articles(keyword):
    try:
        url =
f"https://newsapi.org/v2/everything?q={keyword}&language=en
&apiKey={NEWS_API_KEY}"
        response = requests.get(url, timeout=10)
        response.raise_for_status()

        articles = response.json().get("articles", [])
        news_feed = []
        for article in articles[:5]:
            title = article.get("title", "No Title")
            link = article.get("url", "#")
            summary =
summarize_article(article.get("description", "No
Description"))
            image_url = article.get("urlToImage", None)
            news_feed.append({
                "title": title,
                "url": link,
                "summary": summary,
                "image_url": image_url
            })

        return news_feed
    except requests.exceptions.RequestException as e:
        logger.error(f"Error fetching articles: {e}")
        return []

def summarize_article(text):
    if not text:
        return "No content to summarize."

    # Use a more efficient summarization approach
    doc = nlp(text)
    sentences = [sent.text for sent in doc.sents]

    # Calculate sentence importance based on length and
content
    important_sentences = []
    for sent in sentences[:3]:  # Consider first 3 sentences
        if len(sent.split()) > 5:  # Only include sentences
with more than 5 words
            important_sentences.append(sent)
```

```
    return " ".join(important_sentences[:2])

@app.route("/")
def index():
    return render_template("index.html")

@app.route("/add_preference", methods=["POST"])
@limiter.limit("10 per minute")
def add_preference():
    try:
        keyword = request.form.get("keyword") or
(request.json.get("keyword") if request.is_json else None)

        if not keyword:
            return jsonify({"error": "Keyword is
required."}), 400

        existing = fetch_data(
            "SELECT keyword FROM user_preferences WHERE
LOWER(keyword) = LOWER(?)",
            (keyword,)
        )

        if existing:
            return jsonify({"error": f"'{keyword}' is
already in your preferences."}), 400

        execute_query(
            "INSERT INTO user_preferences (keyword) VALUES
(?)",
            (keyword,)
        )

        return jsonify({"message": f"'{keyword}' added
successfully to preferences!"})

    except Exception as e:
        logger.error(f"Error in add_preference: {e}")
        return jsonify({"error": "An unexpected error
occurred."}), 500

@app.route("/get_feed", methods=["GET"])
@limiter.limit("30 per minute")
def get_feed():
    try:
        keywords = fetch_data("SELECT keyword FROM
user_preferences")
```

```
        news_feed = []

        # Use threading to fetch articles concurrently
        threads = []
        results = []

        def fetch_articles_thread(keyword):
            articles = fetch_articles(keyword[0])
            results.extend(articles)

        for keyword in keywords:
            thread =
threading.Thread(target=fetch_articles_thread,
args=(keyword,))
            threads.append(thread)
            thread.start()

        for thread in threads:
            thread.join()

        return jsonify(results)
    except Exception as e:
        logger.error(f"Error fetching feed: {e}")
        return jsonify({"error": "Failed to fetch feed."}),
500

@app.route("/save_article", methods=["POST"])
@limiter.limit("20 per minute")
def save_article():
    try:
        data = request.json
        title = data.get("title")
        url = data.get("url")
        summary = data.get("summary")
        keyword = data.get("keyword", "")  # Make keyword
optional with default empty string

        if not all([title, url, summary]):
            return jsonify({"error": "Title, URL, and
summary are required."}), 400

        execute_query(
            "INSERT INTO article_history (title, url,
summary, keyword) VALUES (?, ?, ?, ?)",
            (title, url, summary, keyword)
        )
        return jsonify({"message": "Article saved
```

```
successfully!"})
    except Exception as e:
        logger.error(f"Error saving article: {e}")
        return jsonify({"error": "Failed to save
article."}), 500

@app.route("/history", methods=["GET"])
@limiter.limit("30 per minute")
def history():
    try:
        history = fetch_data(
            "SELECT title, url, summary, timestamp, keyword
FROM article_history ORDER BY timestamp DESC LIMIT 100"
        )
        return jsonify([{
            "title": row[0],
            "url": row[1],
            "summary": row[2],
            "timestamp": row[3],
            "keyword": row[4]
        } for row in history])
    except Exception as e:
        logger.error(f"Error fetching history: {e}")
        return jsonify({"error": "Failed to fetch
history."}), 500

@app.route("/listen_article", methods=["POST"])
@limiter.limit("10 per minute")
def listen_article():
    try:
        data = request.json
        title = data.get("title", "Untitled Article")
        summary = data.get("summary", "No summary
available.")

        content = f"Title: {title}. Summary: {summary}"
        temp_dir = os.path.join(os.path.dirname(__file__),
"temp")
        os.makedirs(temp_dir, exist_ok=True)
        audio_file = os.path.join(temp_dir,
"article_audio.mp3")

        tts = gTTS(text=content, lang='en')
        tts.save(audio_file)

        return send_file(
            audio_file,
```

```
            mimetype="audio/mpeg",
            as_attachment=True,
            download_name="article_audio.mp3"
        )
    except Exception as e:
        logger.error(f"Error generating audio: {e}")
        return jsonify({"error": "Failed to generate
audio"}), 500

if __name__ == "__main__":
    # Ensure database is initialized
    try:
        db_pool.initialize_database()
        logger.info("Application started with initialized
database")
    except Exception as e:
        logger.error(f"Failed to initialize database: {e}")
        raise

    app.run(debug=True, threaded=True)
```

Html :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>AI News Aggregator</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/styles.css') }}">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6
.0.0/css/all.min.css">
    <link rel="icon" type="image/png" href="{{
url_for('static', filename='images/default-thumbnail.jpg')
}}">
</head>
<body>
    <div class="app-container">
        <header class="app-header">
            <div class="logo">
                <img src="{{ url_for('static',
filename='images/default-thumbnail.jpg') }}" alt="AI News
Logo" class="logo-img">
                <h1>AI News Aggregator</h1>
            </div>
```

```
            <div class="theme-toggle">
                <i class="fas fa-moon"></i>
            </div>
        </header>

        <main class="main-content">
            <section class="preferences-section">
                <div class="form-control">
                    <div class="input-group">
                        <input type="text" id="keyword"
name="keyword" placeholder="Enter a topic..." required>
                        <button id="addPreference"
type="button" class="add-btn">
                            <i class="fas fa-plus"></i>
                            <span>Add</span>
                        </button>
                    </div>
                </div>

                <div class="form-control">
                    <button id="fetchFeed" type="button"
class="primary-btn">
                        <i class="fas fa-sync-alt"></i>
                        <span>Get Personalized News
Feed</span>
                    </button>
                </div>
            </section>

            <section class="news-feed-section">
                <div id="newsFeed" class="news-feed"></div>
            </section>

            <section class="history-section">
                <div class="section-header">
                    <h2><i class="fas fa-history"></i>
Article History</h2>
                    <button id="viewHistory" type="button"
class="secondary-btn">
                        <i class="fas fa-sync"></i>
                        <span>Refresh History</span>
                    </button>
                </div>
                <div id="articleHistory"
class="article-history"></div>
            </section>
        </main>
```

```
        <div class="loading-overlay">
            <div class="spinner"></div>
        </div>
    </div>

    <script>
        // Add loading state management
        const loadingOverlay =
document.querySelector('.loading-overlay');

        function showLoading() {
            loadingOverlay.style.display = 'flex';
        }

        function hideLoading() {
            loadingOverlay.style.display = 'none';
        }


document.getElementById('addPreference').addEventListener('
click', async () => {
            showLoading();
            const keywordInput =
document.getElementById('keyword');
            const keyword = keywordInput.value.trim();

            if (!keyword) {
                alert('Please enter a keyword');
                hideLoading();
                return;
            }

            try {
                const response = await
fetch('/add_preference', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json',
                    },
                    body: JSON.stringify({ keyword: keyword
})
                });

                const data = await response.json();

                if (response.ok) {
```

```
                    alert(data.message);
                    keywordInput.value = ''; // Clear the
input field
                } else {
                    alert(data.error || 'Failed to add
preference');
                }
            } catch (error) {
                console.error('Error:', error);
                alert('Failed to add preference');
            }
            hideLoading();
        });


document.getElementById('fetchFeed').addEventListener('clic
k', async () => {
            showLoading();
            const response = await fetch('/get_feed');
            const newsFeed = await response.json();
            const newsFeedDiv =
document.getElementById('newsFeed');
            newsFeedDiv.innerHTML = '';

            if (newsFeed.length > 0) {
                newsFeed.forEach(article => {
                    const articleDiv =
document.createElement('div');
                    articleDiv.classList.add('article');

                    const thumbnailHtml = article.image_url
?
                        `<img src="${article.image_url}"
alt="Article thumbnail" class="article-thumbnail"
onerror="this.src='static/images/default-thumbnail.png'">`
:
                        `<img
src="static/images/default-thumbnail.jpg" alt="Default
thumbnail" class="article-thumbnail">`;

                    articleDiv.innerHTML = `
                        ${thumbnailHtml}
                        <div class="article-content">
                            <h3>${article.title}</h3>
                            <p>${article.summary}</p>
                            <div class="article-actions">
                                <a href="${article.url}"
```

```
target="_blank" class="read-more">Read More</a>
                                <button
onclick="listenToArticle('${encodeURIComponent(article.titl
e)}', '${encodeURIComponent(article.summary)}')"
class="listen-btn">Listen</button>
                                <button
onclick="saveArticle('${encodeURIComponent(article.title)}'
, '${encodeURIComponent(article.url)}',
'${encodeURIComponent(article.summary)}')"
class="save-btn">Save Article</button>
                            </div>
                        </div>
                    `;
                    newsFeedDiv.appendChild(articleDiv);
                });
            } else {
                newsFeedDiv.innerHTML = '<p>No articles
found. Please add a keyword preference.</p>';
            }
            hideLoading();
        });


document.getElementById('viewHistory').addEventListener('cl
ick', async () => {
            showLoading();
            const response = await fetch('/history');
            const history = await response.json();
            const articleHistoryDiv =
document.getElementById('articleHistory');
            articleHistoryDiv.innerHTML = '';

            if (history.length > 0) {
                history.forEach(entry => {
                    const entryDiv =
document.createElement('div');
                    entryDiv.classList.add('history-entry');

                    entryDiv.innerHTML = `
                        <div class="history-entry-header">
                            <img src="{{ url_for('static',
filename='images/default-thumbnail.jpg') }}" alt="Article
thumbnail" class="history-thumbnail">
                            <div
class="history-entry-content">
                                <h3><i class="fas
fa-newspaper"></i> ${entry.title}</h3>
```

```
                                <p class="timestamp"><i
class="far fa-clock"></i> ${entry.timestamp}</p>
                            </div>
                        </div>
                        <div class="history-entry-body">
                            <p><i class="fas
fa-quote-left"></i> ${entry.summary}</p>
                            <div
class="history-entry-actions">
                                <a href="${entry.url}"
target="_blank" class="history-btn">
                                    <i class="fas
fa-external-link-alt"></i> Read Article
                                </a>
                                <button
onclick="listenToArticle('${encodeURIComponent(entry.title)
}', '${encodeURIComponent(entry.summary)}')"
class="history-btn">
                                    <i class="fas
fa-volume-up"></i> Listen
                                </button>
                                ${entry.keyword ? `<span
class="keyword-tag"><i class="fas fa-tag"></i>
${entry.keyword}</span>` : ''}
                            </div>
                        </div>
                    `;
                    articleHistoryDiv.appendChild(entryDiv);
                });
            } else {
                articleHistoryDiv.innerHTML = '<div
class="empty-history"><i class="fas fa-inbox"></i><p>No
history available.</p></div>';
            }
            hideLoading();
        });

        async function listenToArticle(title, summary) {
            try {
                const response = await
fetch('/listen_article', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json',
                    },
                    body: JSON.stringify({ title:
decodeURIComponent(title), summary:
```

```
decodeURIComponent(summary) }),
                });

                if (response.ok) {
                    const blob = await response.blob();
                    const audioUrl =
URL.createObjectURL(blob);
                    const audio = new Audio(audioUrl);
                    audio.play();
                } else {
                    alert('Failed to play the article.
Please try again.');
                }
            } catch (error) {
                console.error('Error playing audio:',
error);
                alert('Failed to play the article. Please
try again.');
            }
        }

        // Add this new function for saving articles
        async function saveArticle(title, url, summary) {
            try {
                const response = await
fetch('/save_article', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json',
                    },
                    body: JSON.stringify({
                        title: decodeURIComponent(title),
                        url: decodeURIComponent(url),
                        summary: decodeURIComponent(summary)
                    })
                });

                const data = await response.json();

                if (response.ok) {
                    // Show success message in a more
user-friendly way
                    const successMessage =
document.createElement('div');
                    successMessage.className =
'success-message';
                    successMessage.innerHTML = `
```

```
                        <i class="fas fa-check-circle"></i>
                        <span>Article saved
successfully!</span>
                    `;

document.body.appendChild(successMessage);

                    // Remove the message after 3 seconds
                    setTimeout(() => {
                        successMessage.remove();
                    }, 3000);
                } else {
                    // Show error message in a more
user-friendly way
                    const errorMessage =
document.createElement('div');
                    errorMessage.className =
'error-message';
                    errorMessage.innerHTML = `
                        <i class="fas
fa-exclamation-circle"></i>
                        <span>${data.error || 'Failed to
save article'}</span>
                    `;
                    document.body.appendChild(errorMessage);

                    // Remove the message after 3 seconds
                    setTimeout(() => {
                        errorMessage.remove();
                    }, 3000);
                }
            } catch (error) {
                console.error('Error:', error);
                // Show error message in a more
user-friendly way
                const errorMessage =
document.createElement('div');
                errorMessage.className = 'error-message';
                errorMessage.innerHTML = `
                    <i class="fas
fa-exclamation-circle"></i>
                    <span>Failed to save article. Please try
again.</span>
                `;
                document.body.appendChild(errorMessage);

                // Remove the message after 3 seconds
```

```
                setTimeout(() => {
                    errorMessage.remove();
                }, 3000);
            }
        }

        // Add theme toggle functionality
        const themeToggle =
document.querySelector('.theme-toggle');
        themeToggle.addEventListener('click', () => {
            document.body.classList.toggle('dark-theme');
            const icon = themeToggle.querySelector('i');
            icon.classList.toggle('fa-moon');
            icon.classList.toggle('fa-sun');
        });

        // Add smooth scroll behavior

document.querySelectorAll('a[href^="#"]').forEach(anchor =>
{
            anchor.addEventListener('click', function (e) {
                e.preventDefault();

document.querySelector(this.getAttribute('href')).scrollInt
oView({
                    behavior: 'smooth'
                });
            });
        });
    </script>
</body>
</html>
```

CSS :

```
@import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@
300;400;500;600;700&display=swap');

:root {
  --primary-color: #6366f1;
  --secondary-color: #8b5cf6;
  --accent-color: #ec4899;
  --background-color: #f8fafc;
  --text-color: #1e293b;
  --card-background: #ffffff;
```

```
  --card-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px
4px -1px rgba(0, 0, 0, 0.06);
  --transition-speed: 0.3s;
  --border-radius: 12px;
}

/* Dark theme variables */
.dark-theme {
  --background-color: #0f172a;
  --text-color: #f8fafc;
  --card-background: #1e293b;
  --card-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.2), 0 2px
4px -1px rgba(0, 0, 0, 0.1);
}

* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Poppins', sans-serif;
  background-color: var(--background-color);
  color: var(--text-color);
  line-height: 1.6;
  transition: background-color var(--transition-speed),
color var(--transition-speed);
}

.app-container {
  max-width: 1200px;
  margin: 0 auto;
  padding: 2rem;
}

.app-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 2rem;
  padding: 1rem;
  background: var(--card-background);
  border-radius: var(--border-radius);
  box-shadow: var(--card-shadow);
}
```

```css
.logo {
  display: flex;
  align-items: center;
  gap: 1rem;
}

.logo i {
  font-size: 2rem;
  color: var(--primary-color);
}

.logo h1 {
  font-size: 1.8rem;
  font-weight: 700;
  margin: 0;
  background: linear-gradient(135deg, var(--primary-color),
var(--secondary-color));
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
}

.theme-toggle {
  cursor: pointer;
  padding: 0.5rem;
  border-radius: 50%;
  transition: transform var(--transition-speed);
}

.theme-toggle:hover {
  transform: rotate(30deg);
}

.theme-toggle i {
  font-size: 1.5rem;
  color: var(--primary-color);
}

.main-content {
  display: grid;
  gap: 2rem;
}

.preferences-section {
  background: var(--card-background);
  padding: 2rem;
  border-radius: var(--border-radius);
  box-shadow: var(--card-shadow);
```

```css
}

.input-group {
  display: flex;
  gap: 1rem;
  margin-bottom: 1.5rem;
}

.input-group input {
  flex: 1;
  padding: 1rem;
  border: 2px solid transparent;
  border-radius: var(--border-radius);
  font-size: 1rem;
  background: var(--background-color);
  color: var(--text-color);
  transition: all var(--transition-speed);
}

.input-group input:focus {
  outline: none;
  border-color: var(--primary-color);
  box-shadow: 0 0 0 3px rgba(99, 102, 241, 0.1);
}

.primary-btn, .secondary-btn, .add-btn {
  display: flex;
  align-items: center;
  gap: 0.5rem;
  padding: 1rem 1.5rem;
  border: none;
  border-radius: var(--border-radius);
  font-size: 1rem;
  font-weight: 600;
  cursor: pointer;
  transition: all var(--transition-speed);
}

.primary-btn {
  background: linear-gradient(135deg, var(--primary-color),
var(--secondary-color));
  color: white;
}

.secondary-btn {
  background: var(--background-color);
  color: var(--primary-color);
```

```
}

.add-btn {
  background: var(--accent-color);
  color: white;
}

.primary-btn:hover, .secondary-btn:hover, .add-btn:hover {
  transform: translateY(-2px);
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
}

.news-feed-section, .history-section {
  background: var(--card-background);
  padding: 2rem;
  border-radius: var(--border-radius);
  box-shadow: var(--card-shadow);
}

.section-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 1.5rem;
}

.section-header h2 {
  font-size: 1.5rem;
  font-weight: 600;
  color: var(--primary-color);
}

.article {
  display: grid;
  grid-template-columns: 200px 1fr;
  gap: 1.5rem;
  padding: 1.5rem;
  margin-bottom: 1.5rem;
  background: var(--background-color);
  border-radius: var(--border-radius);
  transition: transform var(--transition-speed), box-shadow
var(--transition-speed);
}

.article:hover {
  transform: translateY(-4px);
  box-shadow: 0 8px 16px rgba(0, 0, 0, 0.1);
```

```css
}

.article-thumbnail {
  width: 100%;
  height: 150px;
  object-fit: cover;
  border-radius: var(--border-radius);
  transition: transform var(--transition-speed);
}

.article:hover .article-thumbnail {
  transform: scale(1.05);
}

.article-content {
  display: flex;
  flex-direction: column;
  gap: 1rem;
}

.article-content h3 {
  font-size: 1.3rem;
  font-weight: 600;
  color: var(--text-color);
}

.article-content p {
  color: var(--text-color);
  opacity: 0.8;
}

.article-actions {
  display: flex;
  gap: 1rem;
  margin-top: auto;
}

.article-actions button, .article-actions a {
  padding: 0.5rem 1rem;
  border-radius: var(--border-radius);
  font-weight: 500;
  transition: all var(--transition-speed);
}

.read-more {
  background: var(--primary-color);
  color: white;
```

```
  text-decoration: none;
}

.listen-btn {
  background: var(--secondary-color);
  color: white;
}

.save-btn {
  background: var(--accent-color);
  color: white;
}

.article-actions button:hover, .article-actions a:hover {
  transform: translateY(-2px);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

.loading-overlay {
  display: none;
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(0, 0, 0, 0.5);
  justify-content: center;
  align-items: center;
  z-index: 1000;
}

.spinner {
  width: 50px;
  height: 50px;
  border: 4px solid rgba(255, 255, 255, 0.3);
  border-radius: 50%;
  border-top-color: var(--primary-color);
  animation: spin 1s ease-in-out infinite;
}

@keyframes spin {
  to { transform: rotate(360deg); }
}

@keyframes fadeIn {
  from { opacity: 0; transform: translateY(20px); }
  to { opacity: 1; transform: translateY(0); }
```

```css
}

#newsFeed > *, #articleHistory > * {
  animation: fadeIn 0.5s ease-out forwards;
}

@media (max-width: 768px) {
  .app-container {
    padding: 1rem;
  }

  .article {
    grid-template-columns: 1fr;
  }

  .article-thumbnail {
    height: 200px;
  }

  .input-group {
    flex-direction: column;
  }

  .article-actions {
    flex-direction: column;
  }
}

/* Notification Messages */
.success-message, .error-message {
    position: fixed;
    top: 20px;
    right: 20px;
    padding: 1rem;
    border-radius: var(--border-radius);
    display: flex;
    align-items: center;
    gap: 0.5rem;
    animation: slideIn 0.3s ease-out forwards;
    z-index: 1000;
    box-shadow: var(--card-shadow);
}

.success-message {
    background: linear-gradient(135deg, #10b981, #059669);
    color: white;
}
```

```css
.error-message {
    background: linear-gradient(135deg, #ef4444, #dc2626);
    color: white;
}

@keyframes slideIn {
    from {
        transform: translateX(100%);
        opacity: 0;
    }
    to {
        transform: translateX(0);
        opacity: 1;
    }
}

@keyframes slideOut {
    from {
        transform: translateX(0);
        opacity: 1;
    }
    to {
        transform: translateX(100%);
        opacity: 0;
    }
}

.success-message i, .error-message i {
    font-size: 1.2rem;
}

.logo-img {
    width: 40px;
    height: 40px;
    border-radius: 8px;
    object-fit: cover;
}

.history-section {
    background: var(--card-background);
    padding: 2rem;
    border-radius: var(--border-radius);
    box-shadow: var(--card-shadow);
}

.history-entry {
```

```
    background: var(--background-color);
    border-radius: var(--border-radius);
    margin-bottom: 1.5rem;
    overflow: hidden;
    transition: transform var(--transition-speed),
box-shadow var(--transition-speed);
}

.history-entry:hover {
    transform: translateY(-4px);
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.1);
}

.history-entry-header {
    display: flex;
    align-items: center;
    gap: 1rem;
    padding: 1rem;
    background: linear-gradient(135deg,
var(--primary-color), var(--secondary-color));
    color: white;
}

.history-thumbnail {
    width: 60px;
    height: 60px;
    border-radius: 8px;
    object-fit: cover;
    border: 2px solid rgba(255, 255, 255, 0.2);
}

.history-entry-content {
    flex: 1;
}

.history-entry-content h3 {
    margin: 0;
    font-size: 1.1rem;
    color: white;
}

.timestamp {
    margin: 0.5rem 0 0;
    font-size: 0.9rem;
    opacity: 0.8;
}
```

```css
.history-entry-body {
    padding: 1.5rem;
}

.history-entry-body p {
    margin: 0 0 1rem;
    line-height: 1.6;
    color: var(--text-color);
}

.history-entry-actions {
    display: flex;
    gap: 1rem;
    flex-wrap: wrap;
    margin-top: 1rem;
}

.history-btn {
    display: inline-flex;
    align-items: center;
    gap: 0.5rem;
    padding: 0.5rem 1rem;
    border: none;
    border-radius: var(--border-radius);
    background: var(--primary-color);
    color: white;
    font-size: 0.9rem;
    cursor: pointer;
    text-decoration: none;
    transition: all var(--transition-speed);
}

.history-btn:hover {
    transform: translateY(-2px);
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

.keyword-tag {
    display: inline-flex;
    align-items: center;
    gap: 0.5rem;
    padding: 0.5rem 1rem;
    border-radius: var(--border-radius);
    background: var(--accent-color);
    color: white;
    font-size: 0.9rem;
}
```

```
.empty-history {
    text-align: center;
    padding: 3rem;
    color: var(--text-color);
    opacity: 0.7;
}

.empty-history i {
    font-size: 3rem;
    margin-bottom: 1rem;
    color: var(--primary-color);
}

.section-header h2 i {
    margin-right: 0.5rem;
    color: var(--primary-color);
}

.fa-quote-left {
    color: var(--primary-color);
    opacity: 0.5;
    margin-right: 0.5rem;
}

/* Dark theme adjustments */
.dark-theme .history-entry {
    background: var(--card-background);
}

.dark-theme .history-btn {
    background: var(--secondary-color);
}
```

JavaScript :

```
document.addEventListener('DOMContentLoaded', function () {
    const addPreferenceBtn =
document.getElementById('addPreference');
    const fetchFeedBtn =
document.getElementById('fetchFeed');
    const viewHistoryBtn =
document.getElementById('viewHistory');
    const newsFeed = document.getElementById('newsFeed');
    const articleHistory =
document.getElementById('articleHistory');
```

```
    addPreferenceBtn.addEventListener('click',
addPreference);
    fetchFeedBtn.addEventListener('click', fetchNewsFeed);
    viewHistoryBtn.addEventListener('click',
viewArticleHistory);

    async function addPreference(event) {
        event.preventDefault();  // Prevent default button
behavior
        const keyword =
document.getElementById('keyword').value.trim();
        if (!keyword) {
            alert('Please enter a valid keyword.');
            return;
        }

        try {
            const response = await fetch('/add_preference',
{
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify({ keyword }),
            });

            const data = await response.json();
            if (response.ok) {
                alert(data.message);
                document.getElementById('keyword').value =
'';
            } else {
                throw new Error(data.error || 'Failed to add
preference.');
            }
        } catch (error) {
            alert(error.message || 'An error occurred while
adding the preference.');
        }
    }

    async function fetchNewsFeed() {
        try {
            const response = await fetch('/get_feed');
            const newsFeedData = await response.json();
            displayNewsFeed(newsFeedData);
        } catch (error) {
```

```
            alert('An error occurred while fetching the news
feed. Please try again.');
        }
    }

    function displayNewsFeed(feed) {
        newsFeed.innerHTML = '';
        if (feed.length === 0) {
            newsFeed.innerHTML = '<p>No articles found for
your preferences.</p>';
            return;
        }

        feed.forEach(article => {
            const articleElement =
document.createElement('div');
            articleElement.className = 'news-item';

            // Try to fetch the thumbnail either from
article.thumbnail or extract it from article.content
            const thumbnail = article.thumbnail ||
extractImageFromContent(article.content);

            articleElement.innerHTML = `
                <div class="article-thumbnail">
                    <img src="${thumbnail ||
'/static/images/default-thumbnail.jpg'}" alt="Thumbnail"
class="thumbnail-img">
                </div>
                <h3>${article.title}</h3>
                <p>${article.summary}</p>
                <a href="${article.url}"
target="_blank">Read more</a>
                <button class="saveArticle"
data-article='${JSON.stringify(article)}'>Save</button>
            `;
            newsFeed.appendChild(articleElement);
        });


document.querySelectorAll('.saveArticle').forEach(button =>
{
            button.addEventListener('click', saveArticle);
        });
    }

    // Helper function to extract image URL from the article
```

```
content
    function extractImageFromContent(content) {
        const parser = new DOMParser();
        const doc = parser.parseFromString(content,
'text/html');
        const img = doc.querySelector('img');
        return img ? img.src : null;
    }

    async function saveArticle(event) {
        const article =
JSON.parse(event.target.getAttribute('data-article'));
        try {
            const response = await fetch('/save_article', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify(article),
            });

            const data = await response.json();
            if (response.ok) {
                alert(data.message);
            } else {
                throw new Error(data.error || 'Failed to
save article.');
            }
        } catch (error) {
            alert(error.message || 'An error occurred while
saving the article.');
        }
    }

    async function viewArticleHistory() {
        try {
            const response = await fetch('/history');
            const history = await response.json();
            displayArticleHistory(history);
        } catch (error) {
            alert('An error occurred while fetching the
article history. Please try again.');
        }
    }

    function displayArticleHistory(history) {
        articleHistory.innerHTML = '';
```

```
        if (history.length === 0) {
            articleHistory.innerHTML = '<p>No saved articles
found.</p>';
            return;
        }

        history.forEach(article => {
            const articleElement =
document.createElement('div');
            articleElement.className = 'history-item';

            // Try to fetch the thumbnail either from
article.thumbnail or extract it from article.content
            const thumbnail = article.thumbnail ||
extractImageFromContent(article.content);

            articleElement.innerHTML = `
                <div class="article-thumbnail">
                    <img src="${thumbnail ||
'/static/images/default-thumbnail.jpg'}" alt="Thumbnail"
class="thumbnail-img">
                </div>
                <h3>${article.title}</h3>
                <p>${article.summary}</p>
                <a href="${article.url}"
target="_blank">Read more</a>
                <span class="timestamp">Saved on: ${new
Date(article.timestamp).toLocaleString()}</span>
            `;
            articleHistory.appendChild(articleElement);
        });
    }
});
```

CONCLUSION

The AI News Aggregator successfully delivers personalized news feeds with modern web technologies and intuitive UI/UX. With TTS, users can listen to news on the go, improving accessibility. Its modular architecture supports future integration of additional features such as user authentication, AI-powered topic suggestions, and offline reading.

This project showcases how AI, automation, and thoughtful design can enhance digital news consumption and user experience.

REFERENCES

- NewsAPI Documentation
- Flask Documentation
- Python TTS Libraries
- Real Python - Flask Projects
- W3Schools - Responsive Web Design
- StackOverflow Discussions on Flask Routing

GitHub Link (Click Here)