Personalized News Dashboard

Date - 19TH May 2024
Title - Personalized News Dashboard
Name - Ayush Kumar

Personalized News Dashboard

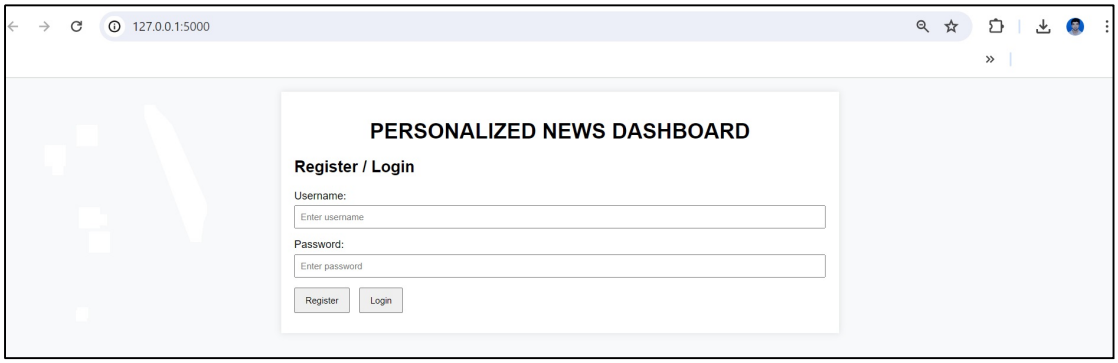


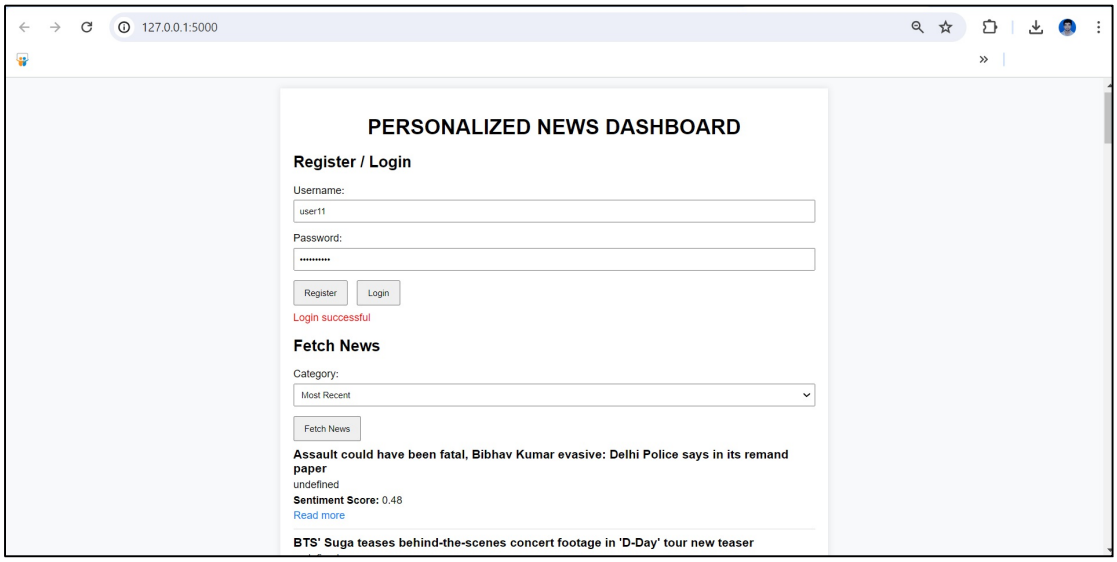
Table of Contents

1. System Design

The System Design section outlines the high-level architecture, data flow, and components of the Personalized News Dashboard project. It covers the overall structure of the system and how different parts interact with each other to provide the desired functionality.

1.1 Overview

The Personalized News Dashboard is a web application that allows users to register, log in, and fetch news articles based on different categories. The system uses JWT for authentication and provides sentiment scores for news headlines.



1.2 Architecture

The application employs a three-tier architecture: the frontend (HTML, CSS, JavaScript), the backend (Flask in Python), and the database (SQLite). Authentication is managed using JWT tokens, while external RSS feeds from Times of India supply the news data.

- Frontend: The frontend, created with HTML, CSS, and JavaScript, comprises forms for user registration and login, and a section to fetch and display news articles. The design ensures a user-friendly interface for interaction.

❖ Index.HTML

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>PERSONALIZED NEWS DASHBOARD</title>
    <!-- Link to the external CSS file -->
    <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>PERSONALIZED NEWS DASHBOARD</h1>
        <!-- Authentication section for registration and login -->
        <div id="auth-section">
            <h2>Register / Login</h2>
            <div class="form-group">
                <label for="username">Username:</label>
                <input type="text" id="username" placeholder="Enter username">
            </div>
            <div class="form-group">
                <label for="password">Password:</label>
```

```html
                <input type="password" id="password" placeholder="Enter password">
            </div>
            <button id="registerBtn">Register</button>
            <button id="loginBtn">Login</button>
            <div id="auth-message"></div>
        </div>

        <!-- News section for fetching and displaying news articles -->
        <div id="news-section" style="display:none;">
            <h2>Fetch News</h2>
            <div class="form-group">
                <label for="category">Category:</label>
                <select id="category">
                    <option value="india">India</option>
                    <option value="sports">Sports</option>
                    <option value="entertainment">Entertainment</option>
                    <option value="science">Science</option>
                    <option value="world">World</option>
                    <option value="technology">Technology</option>
                    <option value="top-stories">Top Stories</option>
                    <option value="most-recent">Most Recent</option>
                    <option value="business">Business</option>
                    <option value="us">US</option>
                    <option value="cricket">Cricket</option>
                    <option value="life-style">Life Style</option>
                    <option value="astrology">Astrology</option>
                    <option value="nri">NRI</option>
                    <option value="environment">Environment</option>
                    <option value="education">Education</option>
                </select>
            </div>
            <button id="fetchNewsBtn">Fetch News</button>
            <div id="news-results"></div>
        </div>
    </div>
    <!-- Link to the external JavaScript file -->
    <script src="{{ url_for('static', filename='js/scripts.js') }}"></script>
</body>
</html>
```

❖ Styles.CSS

```css
/* Set the default font family, margin, padding, and background color for the body */
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f8f9fa;
}
/* Style for the container div to center it and add padding, background color, and shadow */
.container {
    max-width: 800px;
    margin: 20px auto;
    padding: 20px;
    background-color: #ffffff;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
/* Center align the main heading */
h1 {
    text-align: center;
}
/* Style for form groups to add bottom margin */
.form-group {
    margin-bottom: 15px;
}
/* Style for labels to display them as block elements with a bottom margin */
label {
    display: block;
    margin-bottom: 5px;
```

```
}
/* Style for input and select elements to make them full width and add padding */
input, select {
    width: 100%;
    padding: 8px;
    box-sizing: border-box;
}
/* Style for buttons to add padding, right margin, and change cursor to pointer */
button {
    padding: 10px 15px;
    margin-right: 10px;
    cursor: pointer;
}
/* Style for authentication message to add top margin and set text color to red */
#auth-message {
    margin-top: 10px;
    color: red;
}
/* Style for individual news items to add bottom border and padding */
.news-item {
    border-bottom: 1px solid #ddd;
    padding: 10px 0;
}
/* Style for news item headings to remove margin */
.news-item h3 {
    margin: 0;
}
/* Style for news item paragraphs to add top and bottom margin */
.news-item p {
    margin: 5px 0;
}
/* Style for news item links to set color and remove underline */
.news-item a {
    color: #007bff;
    text-decoration: none;
}
```

❖ Scripts.JS

```
document.addEventListener('DOMContentLoaded', () => {
    // Get references to DOM elements
    const registerBtn = document.getElementById('registerBtn');
    const loginBtn = document.getElementById('loginBtn');
    const fetchNewsBtn = document.getElementById('fetchNewsBtn');
    const authMessage = document.getElementById('auth-message');
    const newsSection = document.getElementById('news-section');
    const newsResults = document.getElementById('news-results');
    let accessToken = '';
    // Add event listener for the register button
    registerBtn.addEventListener('click', () => {
        const username = document.getElementById('username').value;
        const password = document.getElementById('password').value;
        // Send a POST request to the /register endpoint
        fetch('/register', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ username, password })
        })
        .then(response => response.json())
        .then(data => {
            // Display the response message
            authMessage.textContent = data.message;
        });
    });
    // Add event listener for the login button
    loginBtn.addEventListener('click', () => {
        const username = document.getElementById('username').value;
```

```
        const password = document.getElementById('password').value;
        // Send a POST request to the /login endpoint
        fetch('/login', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ username, password })
        })
        .then(response => response.json())
        .then(data => {
            if (data.access_token) {
                // Store the access token and display success message
                accessToken = data.access_token;
                authMessage.textContent = "Login successful";
                newsSection.style.display = 'block';
            } else {
                // Display error message
                authMessage.textContent = data.message;
            }
        });
    });
    // Add event listener for the fetch news button
    fetchNewsBtn.addEventListener('click', () => {
        const category = document.getElementById('category').value;

        // Send a GET request to the /news/<category> endpoint
        fetch(`/news/${category}`, {
            method: 'GET',
            headers: {
                'Authorization': `Bearer ${accessToken}`
            }
        })
        .then(response => response.json())
        .then(data => {
            newsResults.innerHTML = '';
            if (data.message) {
                // Display error message if present
                newsResults.textContent = data.message;
            } else {
                // Display fetched news articles
                data.forEach(news => {
                    const newsItem = document.createElement('div');
                    newsItem.classList.add('news-item');
                    newsItem.innerHTML = `
                        <h3>${news.heading}</h3>
                        <p>${news.summary}</p>
                        <p><strong>Sentiment Score:</strong> ${news.sentiment_score}</p>
                        <a href="${news.link}" target="_blank">Read more</a>
                    `;
                    newsResults.appendChild(newsItem);
                });
            }
        });
    });
});
```

- Backend: The backend, developed in Flask, handles routes through blueprints for user authentication and news fetching. It includes utility functions for extracting news details and calculating sentiment scores using NLTK.

❖ __init__.py

```python
from flask import Flask  # Import Flask class to create the app
from flask_sqlalchemy import SQLAlchemy  # Import SQLAlchemy for database interactions
from flask_bcrypt import Bcrypt  # Import Bcrypt for password hashing
from flask_jwt_extended import JWTManager  # Import JWTManager for JWT token handling
from app.config import Config  # Import Config class for application configurations
from flask_cors import CORS  # Import CORS to handle Cross-Origin Resource Sharing
# Instantiate extensions
db: SQLAlchemy = SQLAlchemy()  # Initialize SQLAlchemy object for database interactions
bcrypt: Bcrypt = Bcrypt()  # Initialize Bcrypt object for password hashing
jwt: JWTManager = JWTManager()  # Initialize JWTManager object for handling JWT tokens
def create_app() -> Flask:
    """
    Function to create the Flask app with the required configurations.
    Returns:
        Flask: The Flask app instance.
    """
    app: Flask = Flask(__name__)  # Create the Flask app instance
    CORS(app)  # Enable Cross-Origin Resource Sharing for the app
    app.config.from_object(Config)  # Load configurations from Config class
    db.init_app(app)  # Initialize the app with SQLAlchemy
    bcrypt.init_app(app)  # Initialize the app with Bcrypt
    jwt.init_app(app)  # Initialize the app with JWTManager
    from app.routes import auth_blueprint, news_blueprint  # Import blueprints for routes
    # Register blueprints for authentication and news routes
    app.register_blueprint(auth_blueprint)
    app.register_blueprint(news_blueprint)
    return app  # Return the app instance
```

❖ Config.py

```python
import os  # Import the os module to interact with the operating system's environment variables
class Config:
    """
    Class to store the configurations of the Flask app
    """
    # Secret key for Flask application, used for session management and other security purposes
    # If the environment variable "SECRET_KEY" is not set, it defaults to "your_secret_key"
    SECRET_KEY = os.environ.get("SECRET_KEY") or "your_secret_key"
    # Database URI for SQLAlchemy, which tells SQLAlchemy what database to connect to
    # If the environment variable "DATABASE_URL" is not set, it defaults to a local SQLite database
"users.db"
    SQLALCHEMY_DATABASE_URI = os.environ.get("DATABASE_URL") or "sqlite:///users.db"
    # Disables the modification tracking system of SQLAlchemy, which is unnecessary and can add overhead
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    # Secret key for JWT (JSON Web Tokens), used to sign the tokens
    # If the environment variable "JWT_SECRET_KEY" is not set, it defaults to "your_jwt_secret_key"
    JWT_SECRET_KEY = os.environ.get("JWT_SECRET_KEY") or "your_jwt_secret_key"
```

❖ Models.py

```python
from app import db  # Import the SQLAlchemy instance from the app module
class User(db.Model):
    """
    Class to represent the User model in the database
    """
    # Primary key field for the User model, which uniquely identifies each user
    id = db.Column(db.Integer, primary_key=True)
    # Column to store the username of the user
    # - db.String(150): The maximum length of the username is 150 characters
    # - unique=True: Ensures that no two users can have the same username
    # - nullable=False: The username field cannot be empty (it is a required field)
    username = db.Column(db.String(150), unique=True, nullable=False)
    # Column to store the password of the user
    # - db.String(150): The maximum length of the password is 150 characters
    # - nullable=False: The password field cannot be empty (it is a required field)
    password = db.Column(db.String(150), nullable=False)
```

❖ Routes.py

```python
import feedparser
from app import db, bcrypt
from app.models import User
from flask import Blueprint, request, jsonify, Response, render_template, send_from_directory
from app.utils import extract_news_details, calculate_sentiment_score
from flask_jwt_extended import create_access_token, jwt_required, get_jwt_identity
from typing import Dict, List, Union, Any, Optional
# Define blueprints for authentication and news routes
auth_blueprint: Blueprint = Blueprint("auth", __name__)
news_blueprint: Blueprint = Blueprint("news", __name__)
@auth_blueprint.route("/", methods=["GET"])
def home() -> Response:
    """
    Serve the index.html page.
    Returns:
        Response: The rendered index.html page.
    """
    return render_template("index.html")
@auth_blueprint.route("/test", methods=["GET"])
def test() -> Response:
    """
    Function to test the API.
    Returns:
        Response: JSON response with a message and HTTP status code 200.
    """
    return jsonify(message="Test successful"), 200
@auth_blueprint.route("/register", methods=["POST"])
def register() -> Response:
    """
    Endpoint to register a new user.
    Expects:
        JSON payload with "username" and "password" fields.
    Returns:
        Response: JSON response with a message and HTTP status code.
    """
    data: Optional[Dict[str, Any]] = request.get_json()
    if not data or not data.get("username") or not data.get("password"):
        # Return an error response if username or password is missing
        return jsonify(message="Missing username or password"), 400
    # Hash the password for secure storage
    hashed_password: str = bcrypt.generate_password_hash(data["password"]).decode("utf-8")
```

```python
    # Create a new user instance
    new_user: User = User(username=data["username"], password=hashed_password)
    # Add and commit the new user to the database
    db.session.add(new_user)
    db.session.commit()
    return jsonify(message="User registered successfully"), 201
@auth_blueprint.route("/login", methods=["POST"])
def login() -> Response:
    """
    Endpoint to log in a user.
    Expects:
        JSON payload with "username" and "password" fields.
    Returns:
        Response: JSON response with an access token and HTTP status code, or an error message.
    """
    data: Optional[Dict[str, Any]] = request.get_json()
    if not data or not data.get("username") or not data.get("password"):
        # Return an error response if username or password is missing
        return jsonify(message="Missing username or password"), 400
    # Query the database for the user with the provided username
    user: Optional[User] = User.query.filter_by(username=data["username"]).first()
    # Check if user exists and password matches
    if user and bcrypt.check_password_hash(user.password, data["password"]):
        # Create an access token for the user
        access_token: str = create_access_token(identity=user.id)
        return jsonify(access_token=access_token), 200
    return jsonify(message="Invalid credentials"), 401
@news_blueprint.route("/news/<string:category>", methods=["GET"])
@jwt_required()
def get_news(category: str) -> Response:
    """
    Endpoint to get news articles based on category.
    Args:
        category (str): The news category to fetch articles for.
    Returns:
        Response: JSON response with the list of news articles and their sentiment scores, or an error
message.
    """
    # Dictionary mapping categories to their respective RSS feed URLs
    urls: Dict[str, str] = {
        "india": "https://timesofindia.indiatimes.com/rssfeeds/-2128936835.cms",
        "sports": "https://timesofindia.indiatimes.com/rssfeeds/4719148.cms",
        "entertainment": "http://timesofindia.indiatimes.com/rssfeeds/1081479906.cms",
        "science": "https://timesofindia.indiatimes.com/rssfeeds/-2128672765.cms",
        "world": "http://timesofindia.indiatimes.com/rssfeeds/296589292.cms",
        "technology": "http://timesofindia.indiatimes.com/rssfeeds/66949542.cms",
        "top-stories": "http://timesofindia.indiatimes.com/rssfeedstopstories.cms",
        "most-recent": "http://timesofindia.indiatimes.com/rssfeedmostrecent.cms",
        "business": "http://timesofindia.indiatimes.com/rssfeeds/1898055.cms",
        "us": "https://timesofindia.indiatimes.com/rssfeeds_us/72258322.cms",
        "cricket": "http://timesofindia.indiatimes.com/rssfeeds/54829575.cms",
        "life-style": "http://timesofindia.indiatimes.com/rssfeeds/2886704.cms",
        "astrology": "https://timesofindia.indiatimes.com/rssfeeds/65857041.cms",
        "nri": "http://timesofindia.indiatimes.com/rssfeeds/7098551.cms",
        "environment": "http://timesofindia.indiatimes.com/rssfeeds/2647163.cms",
        "education": "http://timesofindia.indiatimes.com/rssfeeds/913168846.cms",
    }
    if category in urls:
        # Parse the RSS feed for the given category
        feed: feedparser.FeedParserDict = feedparser.parse(urls[category])
        # Extract news details from the feed entries
        extracted_news: List[Dict[str, Any]] = extract_news_details(feed.entries)
        # Calculate sentiment score for each news heading
        for news in extracted_news:
            news["sentiment_score"] = calculate_sentiment_score(news["heading"])
        return jsonify(extracted_news), 200
    return jsonify(message="Category not found"), 404
@auth_blueprint.route("/sentimentScore", methods=["POST"])
def sentimentScore() -> Response:
    """
    Endpoint to calculate the sentiment score of a news headline.
    Expects:
        JSON payload with "heading" field.
```

```
    Returns:
        Response: JSON response with the sentiment score and HTTP status code.
    """
    data: Optional[Dict[str, Any]] = request.get_json()
    if not data or not data.get("heading"):
        # Return an error response if heading is missing
        return jsonify(message="Missing heading"), 400
    heading: str = data["heading"]
    # Calculate sentiment score for the provided heading
    print(f"Sentiment score for heading: {heading} is {calculate_sentiment_score(heading)}")
    return jsonify(sentiment_score=calculate_sentiment_score(heading)), 200
@auth_blueprint.route('/static/<path:path>')
def send_static(path: str) -> Response:
    """
    Serve static files.
    Args:
        path (str): The path to the static file.
    Returns:
        Response: The static file.
    """
    return send_from_directory('static', path)
```

❖ Utils.py

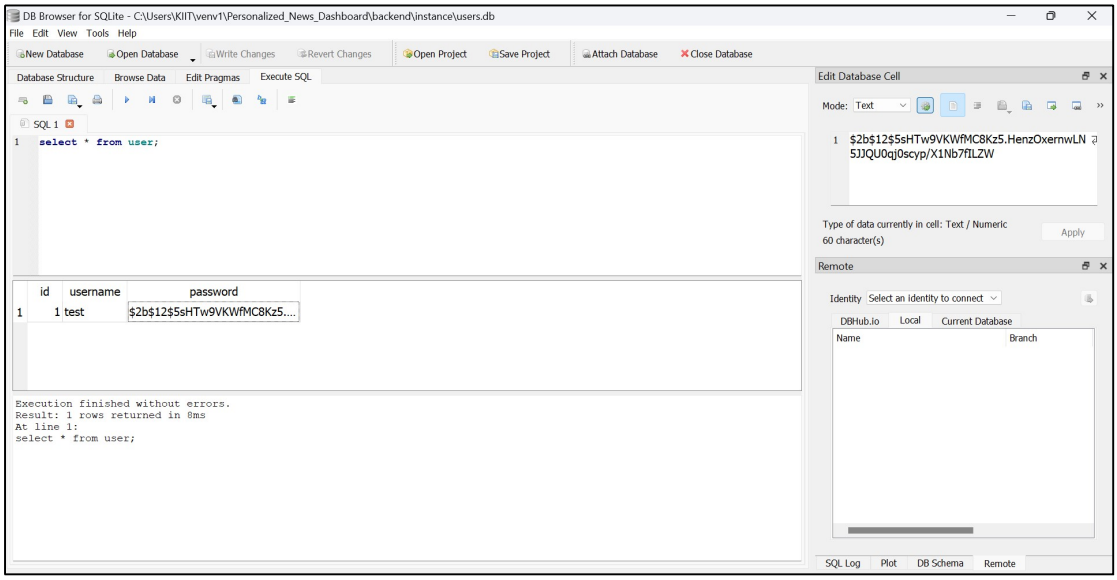
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from typing import List, Dict, Any
# Download the VADER lexicon for sentiment analysis
nltk.download("vader_lexicon")
def extract_news_details(feed: List[Dict[str, Any]]) -> List[Dict[str, str]]:
    """
    Extracts news details from the given feed.
    Args:
        feed (List[Dict[str, Any]]): The feed containing news items.
    Returns:
        List[Dict[str, str]]: A list of dictionaries with news details including heading, link, and image.
    """
    news_details: List[Dict[str, str]] = []
    for item in feed:
        if isinstance(item, dict):
            # Extract the heading and link from the item
            heading: str = item.get("title", "")
            link: str = item.get("link", "")
            image: str = ""
            # Extract image link if available
            links: List[Dict[str, str]] = item.get("links", [])
            if isinstance(links, list):
                for link_item in links:
                    if link_item.get("type", "").startswith("image"):
                        image = link_item.get("href", "")
                        break
            # Create a dictionary with the news details
            news: Dict[str, str] = {"heading": heading, "link": link, "image": image}
            news_details.append(news)
    return news_details
def calculate_sentiment_score(heading: str) -> float:
    """
    Calculates the sentiment score of a given heading.
    Args:
        heading (str): The heading for which to calculate the sentiment score.
    Returns:
        float: The calculated sentiment score, scaled and rounded to two decimal places.
    """
    # Initialize the SentimentIntensityAnalyzer
    sid: SentimentIntensityAnalyzer = SentimentIntensityAnalyzer()
    # Get the sentiment scores for the heading
    sentiment_score: float = sid.polarity_scores(heading)["compound"]
    # Scale and round the compound score to fit within the range 0 to 5
```

```
    return round((sentiment_score + 1) * 2.5, 2)
```

❖ Run.py

```
from flask import Flask
from app import create_app, db
# Create the Flask application using the factory function
app: Flask = create_app()
if __name__ == "__main__":
    # Run the application
    with app.app_context():
        """
        Create all database tables within the application context.
        This ensures that the tables are created in the context of the current Flask application.
        """
        db.create_all()
    # Start the Flask application in debug mode
    app.run(debug=True)
```

- Database: SQLite serves as the database backend, offering reliability and performance in storing user data securely. The schema design, particularly the User table, efficiently manages user authentication, contributing to the application's robustness and data integrity.

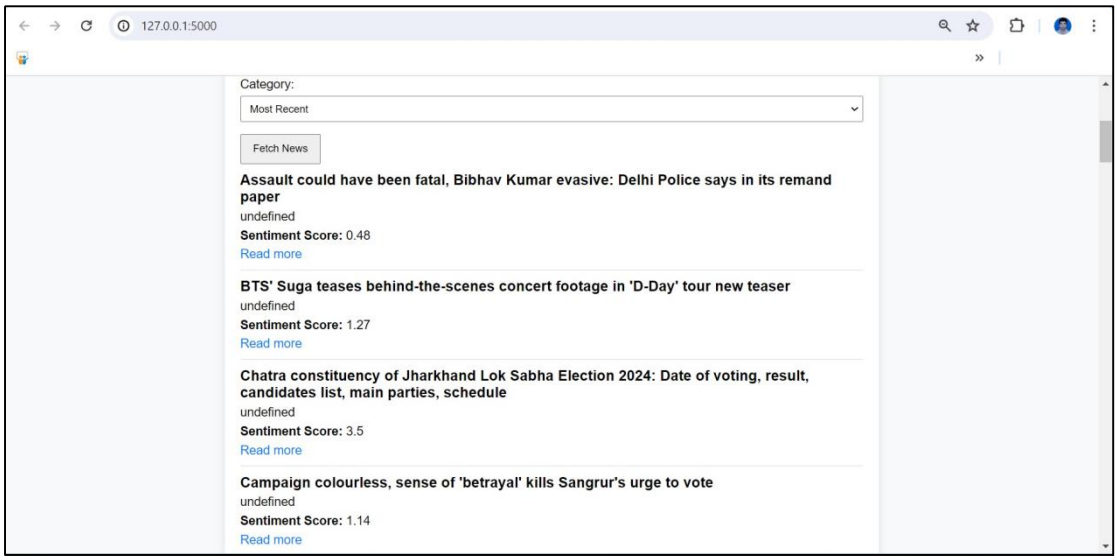


- Authentication: Utilizes JWT (JSON Web Tokens) for secure and stateless user authentication, enabling secure transmission of user identity information between client and server without the need for session management.

```python
# Define blueprints for authentication and news routes
auth_blueprint: Blueprint = Blueprint("auth", __name__)
news_blueprint: Blueprint = Blueprint("news", __name__)
@auth_blueprint.route("/", methods=["GET"])
def home() -> Response:
    """
    Serve the index.html page.
    Returns:
        Response: The rendered index.html page.
    """
    return render_template("index.html")
@auth_blueprint.route("/test", methods=["GET"])
def test() -> Response:
    """
    Function to test the API.
    Returns:
        Response: JSON response with a message and HTTP status code 200.
    """
    return jsonify(message="Test successful"), 200
@auth_blueprint.route("/register", methods=["POST"])
def register() -> Response:
    """
    Endpoint to register a new user.
    Expects:
        JSON payload with "username" and "password" fields.
    Returns:
        Response: JSON response with a message and HTTP status code.
    """
    data: Optional[Dict[str, Any]] = request.get_json()
    if not data or not data.get("username") or not data.get("password"):
        # Return an error response if username or password is missing
        return jsonify(message="Missing username or password"), 400
    # Hash the password for secure storage
    hashed_password: str = bcrypt.generate_password_hash(data["password"]).decode("utf-8")
    # Create a new user instance
    new_user: User = User(username=data["username"], password=hashed_password)
    # Add and commit the new user to the database
    db.session.add(new_user)
    db.session.commit()
    return jsonify(message="User registered successfully"), 201
@auth_blueprint.route("/login", methods=["POST"])
def login() -> Response:
    """
    Endpoint to log in a user.
    Expects:
        JSON payload with "username" and "password" fields.
    Returns:
        Response: JSON response with an access token and HTTP status code, or an error message.
    """
    data: Optional[Dict[str, Any]] = request.get_json()
    if not data or not data.get("username") or not data.get("password"):
        # Return an error response if username or password is missing
        return jsonify(message="Missing username or password"), 400
    # Query the database for the user with the provided username
    user: Optional[User] = User.query.filter_by(username=data["username"]).first()
    # Check if user exists and password matches
    if user and bcrypt.check_password_hash(user.password, data["password"]):
        # Create an access token for the user
        access_token: str = create_access_token(identity=user.id)
        return jsonify(access_token=access_token), 200
    return jsonify(message="Invalid credentials"), 401
@news_blueprint.route("/news/<string:category>", methods=["GET"])
@jwt_required()
```

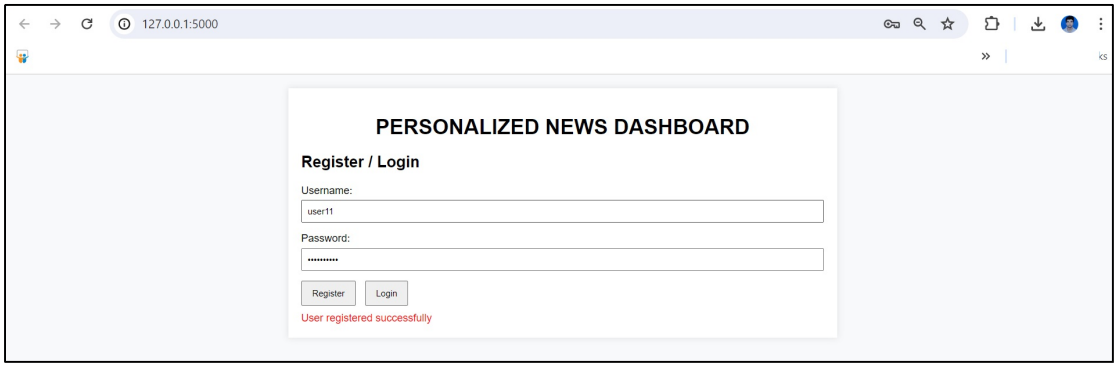
```python
def get_news(category: str) -> Response:
    """
    Endpoint to get news articles based on category.
    Args:
        category (str): The news category to fetch articles for.
    Returns:
        Response: JSON response with the list of news articles and their sentiment scores, or an error
message.
    """
    # Dictionary mapping categories to their respective RSS feed URLs
    urls: Dict[str, str] = {
        "india": "https://timesofindia.indiatimes.com/rssfeeds/-2128936835.cms",
        "sports": "https://timesofindia.indiatimes.com/rssfeeds/4719148.cms",
        "entertainment": "http://timesofindia.indiatimes.com/rssfeeds/1081479906.cms",
        "science": "https://timesofindia.indiatimes.com/rssfeeds/-2128672765.cms",
        "world": "http://timesofindia.indiatimes.com/rssfeeds/296589292.cms",
        "technology": "http://timesofindia.indiatimes.com/rssfeeds/66949542.cms",
        "top-stories": "http://timesofindia.indiatimes.com/rssfeedstopstories.cms",
        "most-recent": "http://timesofindia.indiatimes.com/rssfeedmostrecent.cms",
        "business": "http://timesofindia.indiatimes.com/rssfeeds/1898055.cms",
        "us": "https://timesofindia.indiatimes.com/rssfeeds_us/72258322.cms",
        "cricket": "http://timesofindia.indiatimes.com/rssfeeds/54829575.cms",
        "life-style": "http://timesofindia.indiatimes.com/rssfeeds/2886704.cms",
        "astrology": "https://timesofindia.indiatimes.com/rssfeeds/65857041.cms",
        "nri": "http://timesofindia.indiatimes.com/rssfeeds/7098551.cms",
        "environment": "http://timesofindia.indiatimes.com/rssfeeds/2647163.cms",
        "education": "http://timesofindia.indiatimes.com/rssfeeds/913168846.cms",
    }
    if category in urls:
        # Parse the RSS feed for the given category
        feed: feedparser.FeedParserDict = feedparser.parse(urls[category])
        # Extract news details from the feed entries
        extracted_news: List[Dict[str, Any]] = extract_news_details(feed.entries)
        # Calculate sentiment score for each news heading
        for news in extracted_news:
            news["sentiment_score"] = calculate_sentiment_score(news["heading"])
        return jsonify(extracted_news), 200
    return jsonify(message="Category not found"), 404
@auth_blueprint.route("/sentimentScore", methods=["POST"])
def sentimentScore() -> Response:
    """
    Endpoint to calculate the sentiment score of a news headline.
    Expects:
        JSON payload with "heading" field.
    Returns:
        Response: JSON response with the sentiment score and HTTP status code.
    """
    data: Optional[Dict[str, Any]] = request.get_json()
    if not data or not data.get("heading"):
        # Return an error response if heading is missing
        return jsonify(message="Missing heading"), 400
    heading: str = data["heading"]
    # Calculate sentiment score for the provided heading
    print(f"Sentiment score for heading: {heading} is {calculate_sentiment_score(heading)}")
    return jsonify(sentiment_score=calculate_sentiment_score(heading)), 200
@auth_blueprint.route('/static/<path:path>')
def send_static(path: str) -> Response:
    """
    Serve static files.
    Args:
        path (str): The path to the static file.
    Returns:
        Response: The static file.
    """
    return send_from_directory('static', path)
```

- External Services: Integrates external RSS feeds from Times of India to source news content dynamically, enriching the application's database with up-to-date news articles across various categories.



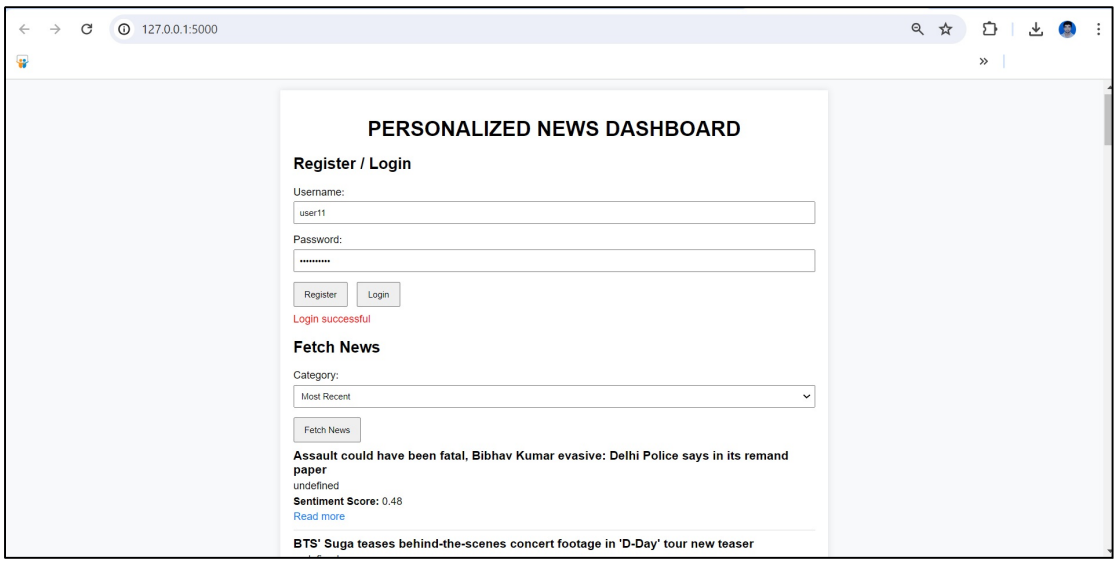
1.3 Data Flow

- User Registration/Login:
  - The user registers or logs in through the frontend.
  - The frontend sends an HTTP request to the backend.
  - The backend processes the request and interacts with the database.
  - The backend returns a response to the frontend.



- Fetching News
  - The user selects a news category and requests news articles.
  - The frontend sends an HTTP request to the backend with the selected category.

- The backend fetches news articles from the RSS feed, calculates sentiment scores, and returns the articles to the frontend.



2. Implementation Details

Implementation Details of the Personalized News Dashboard project include backend development with Flask for user authentication and news fetching, frontend design using HTML/CSS/JavaScript for user interaction, and integration with external RSS feeds from Times of India for dynamic news content. Additionally, SQLite is utilized as the database for storing user credentials and Flask JWT Extended for secure authentication token generation and validation.

2.1 Backend

The backend is built using Flask and consists of several components:

- Blueprints: The application uses blueprints to organize routes.

```
# Define blueprints for authentication and news routes
auth_blueprint: Blueprint = Blueprint("auth", __name__)
news_blueprint: Blueprint = Blueprint("news", __name__)
```

- auth_blueprint: Handles user authentication (registration and login).

```
@auth_blueprint.route("/", methods=["GET"])
def home() -> Response:
    """
    Serve the index.html page.
```

```python
    Returns:
        Response: The rendered index.html page.
    """
    return render_template("index.html")
@auth_blueprint.route("/test", methods=["GET"])
def test() -> Response:
    """
    Function to test the API.
    Returns:
        Response: JSON response with a message and HTTP status code 200.
    """
    return jsonify(message="Test successful"), 200
@auth_blueprint.route("/register", methods=["POST"])
def register() -> Response:
    """
    Endpoint to register a new user.
    Expects:
        JSON payload with "username" and "password" fields.
    Returns:
        Response: JSON response with a message and HTTP status code.
    """
    data: Optional[Dict[str, Any]] = request.get_json()
    if not data or not data.get("username") or not data.get("password"):
        # Return an error response if username or password is missing
        return jsonify(message="Missing username or password"), 400
    # Hash the password for secure storage
    hashed_password: str = bcrypt.generate_password_hash(data["password"]).decode("utf-8")
    # Create a new user instance
    new_user: User = User(username=data["username"], password=hashed_password)
    # Add and commit the new user to the database
    db.session.add(new_user)
    db.session.commit()
    return jsonify(message="User registered successfully"), 201
@auth_blueprint.route("/login", methods=["POST"])
def login() -> Response:
    """
    Endpoint to log in a user.
    Expects:
        JSON payload with "username" and "password" fields.
    Returns:
        Response: JSON response with an access token and HTTP status code, or an error message.
    """
    data: Optional[Dict[str, Any]] = request.get_json()
    if not data or not data.get("username") or not data.get("password"):
        # Return an error response if username or password is missing
        return jsonify(message="Missing username or password"), 400
    # Query the database for the user with the provided username
    user: Optional[User] = User.query.filter_by(username=data["username"]).first()
    # Check if user exists and password matches
    if user and bcrypt.check_password_hash(user.password, data["password"]):
        # Create an access token for the user
        access_token: str = create_access_token(identity=user.id)
        return jsonify(access_token=access_token), 200
    return jsonify(message="Invalid credentials"), 401
```

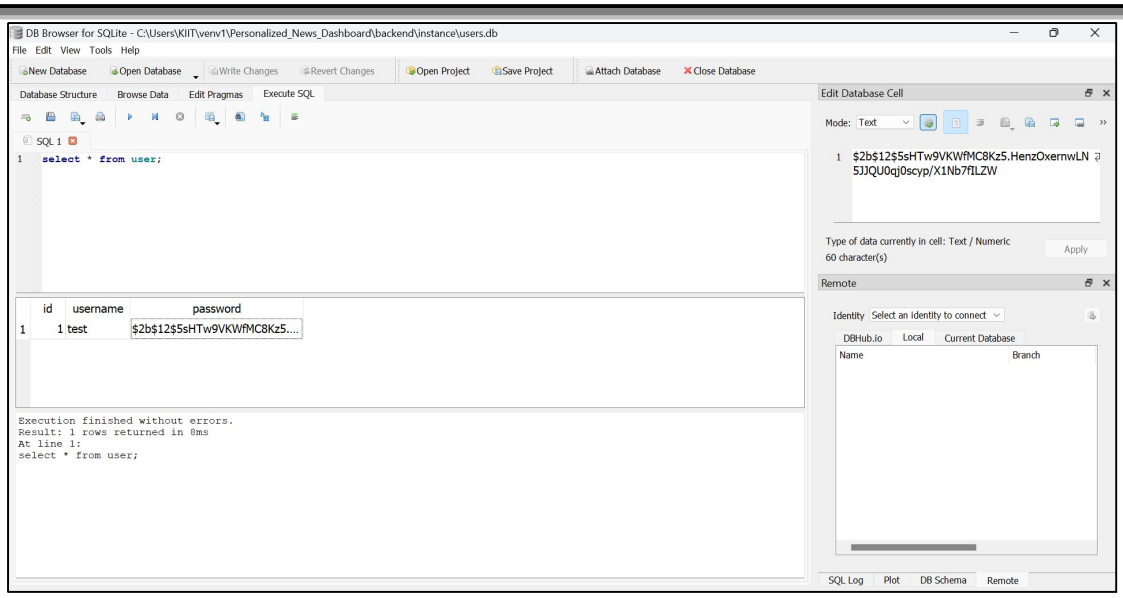
```python
@auth_blueprint.route("/sentimentScore", methods=["POST"])
def sentimentScore() -> Response:
    """
    Endpoint to calculate the sentiment score of a news headline.
    Expects:
        JSON payload with "heading" field.
    Returns:
        Response: JSON response with the sentiment score and HTTP status code.
    """
    data: Optional[Dict[str, Any]] = request.get_json()
    if not data or not data.get("heading"):
        # Return an error response if heading is missing
        return jsonify(message="Missing heading"), 400
    heading: str = data["heading"]
    # Calculate sentiment score for the provided heading
```

```
    print(f"Sentiment score for heading: {heading} is {calculate_sentiment_score(heading)}")
    return jsonify(sentiment_score=calculate_sentiment_score(heading)), 200
@auth_blueprint.route('/static/<path:path>')
def send_static(path: str) -> Response:
    """
    Serve static files.
    Args:
        path (str): The path to the static file.
    Returns:
        Response: The static file.
    """
    return send_from_directory('static', path)
```

- news_blueprint: Handles fetching and returning news articles.

```
@news_blueprint.route("/news/<string:category>", methods=["GET"])
@jwt_required()
def get_news(category: str) -> Response:
    """
    Endpoint to get news articles based on category.
    Args:
        category (str): The news category to fetch articles for.
    Returns:
        Response: JSON response with the list of news articles and their sentiment scores, or an error
message.
    """
    # Dictionary mapping categories to their respective RSS feed URLs
    urls: Dict[str, str] = {
        "india": "https://timesofindia.indiatimes.com/rssfeeds/-2128936835.cms",
        "sports": "https://timesofindia.indiatimes.com/rssfeeds/4719148.cms",
        "entertainment": "http://timesofindia.indiatimes.com/rssfeeds/1081479906.cms",
        "science": "https://timesofindia.indiatimes.com/rssfeeds/-2128672765.cms",
        "world": "http://timesofindia.indiatimes.com/rssfeeds/296589292.cms",
        "technology": "http://timesofindia.indiatimes.com/rssfeeds/66949542.cms",
        "top-stories": "http://timesofindia.indiatimes.com/rssfeedstopstories.cms",
        "most-recent": "http://timesofindia.indiatimes.com/rssfeedmostrecent.cms",
        "business": "http://timesofindia.indiatimes.com/rssfeeds/1898055.cms",
        "us": "https://timesofindia.indiatimes.com/rssfeeds_us/72258322.cms",
        "cricket": "http://timesofindia.indiatimes.com/rssfeeds/54829575.cms",
        "life-style": "http://timesofindia.indiatimes.com/rssfeeds/2886704.cms",
        "astrology": "https://timesofindia.indiatimes.com/rssfeeds/65857041.cms",
        "nri": "http://timesofindia.indiatimes.com/rssfeeds/7098551.cms",
        "environment": "http://timesofindia.indiatimes.com/rssfeeds/2647163.cms",
        "education": "http://timesofindia.indiatimes.com/rssfeeds/913168846.cms",
    }
    if category in urls:
        # Parse the RSS feed for the given category
        feed: feedparser.FeedParserDict = feedparser.parse(urls[category])
        # Extract news details from the feed entries
        extracted_news: List[Dict[str, Any]] = extract_news_details(feed.entries)
        # Calculate sentiment score for each news heading
        for news in extracted_news:
            news["sentiment_score"] = calculate_sentiment_score(news["heading"])
        return jsonify(extracted_news), 200
    return jsonify(message="Category not found"), 404
```

- Database Models: The project utilizes SQLite for the database, featuring a User model that stores user credentials including username and hashed password. This model supports user registration and login functionalities



❖ Utilities: The project includes utilities such as extract_news_details for parsing news data from RSS feeds, and calculate_sentiment_score for determining the sentiment of news headlines using NLTK's VADER sentiment analysis tool. These utilities enhance the functionality of the news fetching and presentation features.

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from typing import List, Dict, Any
# Download the VADER lexicon for sentiment analysis
nltk.download("vader_lexicon")
def extract_news_details(feed: List[Dict[str, Any]]) -> List[Dict[str, str]]:
    """
    Extracts news details from the given feed.
    Args:
        feed (List[Dict[str, Any]]): The feed containing news items.
    Returns:
        List[Dict[str, str]]: A list of dictionaries with news details including heading, link, and image.
    """
    news_details: List[Dict[str, str]] = []
    for item in feed:
        if isinstance(item, dict):
            # Extract the heading and link from the item
            heading: str = item.get("title", "")
            link: str = item.get("link", "")
            image: str = ""
            # Extract image link if available
            links: List[Dict[str, str]] = item.get("links", [])
            if isinstance(links, list):
                for link_item in links:
                    if link_item.get("type", "").startswith("image"):
                        image = link_item.get("href", "")
                        break
            # Create a dictionary with the news details
            news: Dict[str, str] = {"heading": heading, "link": link, "image": image}
            news_details.append(news)
    return news_details
def calculate_sentiment_score(heading: str) -> float:
    """
    Calculates the sentiment score of a given heading.
    Args:
        heading (str): The heading for which to calculate the sentiment score.
```

```
    Returns:
        float: The calculated sentiment score, scaled and rounded to two decimal places.
    """
    # Initialize the SentimentIntensityAnalyzer
    sid: SentimentIntensityAnalyzer = SentimentIntensityAnalyzer()
    # Get the sentiment scores for the heading
    sentiment_score: float = sid.polarity_scores(heading)["compound"]
    # Scale and round the compound score to fit within the range 0 to 5
    return round((sentiment_score + 1) * 2.5, 2)
```

2.2 Frontend: The frontend is built using HTML, CSS, and JavaScript modules.

❖ HTML: Defines the structure of the web pages.
index.html: It contains main page with sections for authentication and fetching news.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>PERSONALIZED NEWS DASHBOARD</title>
    <!-- Link to the external CSS file -->
    <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>PERSONALIZED NEWS DASHBOARD</h1>
        <!-- Authentication section for registration and login -->
        <div id="auth-section">
            <h2>Register / Login</h2>
            <div class="form-group">
                <label for="username">Username:</label>
                <input type="text" id="username" placeholder="Enter username">
            </div>
            <div class="form-group">
                <label for="password">Password:</label>
                <input type="password" id="password" placeholder="Enter password">
            </div>
            <button id="registerBtn">Register</button>
            <button id="loginBtn">Login</button>
            <div id="auth-message"></div>
        </div>

        <!-- News section for fetching and displaying news articles -->
        <div id="news-section" style="display:none;">
            <h2>Fetch News</h2>
            <div class="form-group">
                <label for="category">Category:</label>
                <select id="category">
                    <option value="india">India</option>
                    <option value="sports">Sports</option>
                    <option value="entertainment">Entertainment</option>
                    <option value="science">Science</option>
                    <option value="world">World</option>
                    <option value="technology">Technology</option>
                    <option value="top-stories">Top Stories</option>
                    <option value="most-recent">Most Recent</option>
                    <option value="business">Business</option>
                    <option value="us">US</option>
                    <option value="cricket">Cricket</option>
                    <option value="life-style">Life Style</option>
                    <option value="astrology">Astrology</option>
                    <option value="nri">NRI</option>
                    <option value="environment">Environment</option>
                    <option value="education">Education</option>
```

```
                </select>
            </div>
            <button id="fetchNewsBtn">Fetch News</button>
            <div id="news-results"></div>
        </div>
    </div>
    <!-- Link to the external JavaScript file -->
    <script src="{{ url_for('static', filename='js/scripts.js') }}"></script>
</body>
</html>
```

❖ CSS: Defines the styles for the web pages.
styles.css: It contain styles for the main page, forms, buttons, and news items.

```
/* Set the default font family, margin, padding, and background color for the body */
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f8f9fa;
}
/* Style for the container div to center it and add padding, background color, and shadow */
.container {
    max-width: 800px;
    margin: 20px auto;
    padding: 20px;
    background-color: #ffffff;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
/* Center align the main heading */
h1 {
    text-align: center;
}
/* Style for form groups to add bottom margin */
.form-group {
    margin-bottom: 15px;
}
/* Style for labels to display them as block elements with a bottom margin */
label {
    display: block;
    margin-bottom: 5px;
}
/* Style for input and select elements to make them full width and add padding */
input, select {
    width: 100%;
    padding: 8px;
    box-sizing: border-box;
}
/* Style for buttons to add padding, right margin, and change cursor to pointer */
button {
    padding: 10px 15px;
    margin-right: 10px;
    cursor: pointer;
}
/* Style for authentication message to add top margin and set text color to red */
#auth-message {
    margin-top: 10px;
    color: red;
}
/* Style for individual news items to add bottom border and padding */
.news-item {
    border-bottom: 1px solid #ddd;
    padding: 10px 0;
}
/* Style for news item headings to remove margin */
.news-item h3 {
    margin: 0;
```

```
}
/* Style for news item paragraphs to add top and bottom margin */
.news-item p {
    margin: 5px 0;
}
/* Style for news item links to set color and remove underline */
.news-item a {
    color: #007bff;
    text-decoration: none;
}
```

❖ JavaScript: Adds interactivity to the web pages.

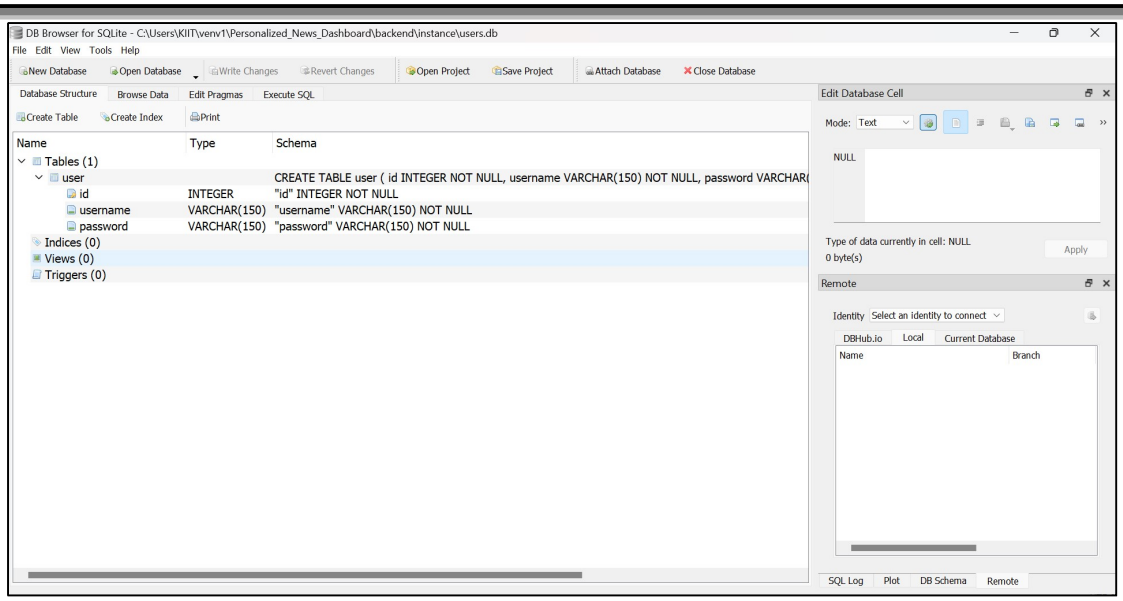
scripts.js: It handles user interactions like registration, login, and fetching news.

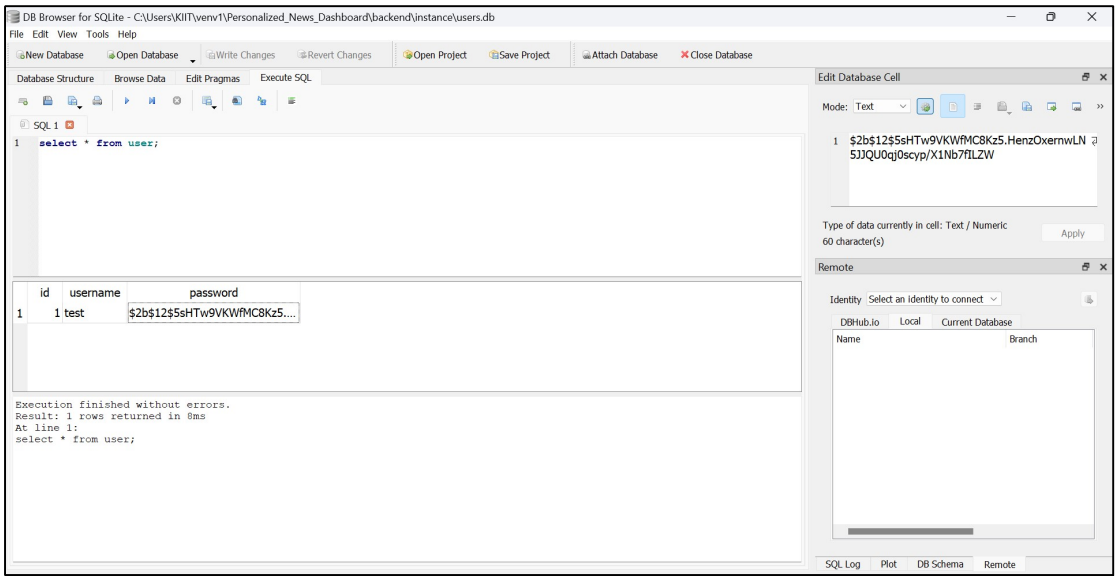
```
document.addEventListener('DOMContentLoaded', () => {
    // Get references to DOM elements
    const registerBtn = document.getElementById('registerBtn');
    const loginBtn = document.getElementById('loginBtn');
    const fetchNewsBtn = document.getElementById('fetchNewsBtn');
    const authMessage = document.getElementById('auth-message');
    const newsSection = document.getElementById('news-section');
    const newsResults = document.getElementById('news-results');
    let accessToken = '';
    // Add event listener for the register button
    registerBtn.addEventListener('click', () => {
        const username = document.getElementById('username').value;
        const password = document.getElementById('password').value;
        // Send a POST request to the /register endpoint
        fetch('/register', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ username, password })
        })
        .then(response => response.json())
        .then(data => {
            // Display the response message
            authMessage.textContent = data.message;
        });
    });
    // Add event listener for the login button
    loginBtn.addEventListener('click', () => {
        const username = document.getElementById('username').value;
        const password = document.getElementById('password').value;
        // Send a POST request to the /login endpoint
        fetch('/login', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ username, password })
        })
        .then(response => response.json())
        .then(data => {
            if (data.access_token) {
                // Store the access token and display success message
                accessToken = data.access_token;
                authMessage.textContent = "Login successful";
                newsSection.style.display = 'block';
            } else {
                // Display error message
                authMessage.textContent = data.message;
            }
        });
    });
```

```
    // Add event listener for the fetch news button
    fetchNewsBtn.addEventListener('click', () => {
        const category = document.getElementById('category').value;

        // Send a GET request to the /news/<category> endpoint
        fetch(`/news/${category}`, {
            method: 'GET',
            headers: {
                'Authorization': `Bearer ${accessToken}`
            }
        })
        .then(response => response.json())
        .then(data => {
            newsResults.innerHTML = '';
            if (data.message) {
                // Display error message if present
                newsResults.textContent = data.message;
            } else {
                // Display fetched news articles
                data.forEach(news => {
                    const newsItem = document.createElement('div');
                    newsItem.classList.add('news-item');
                    newsItem.innerHTML = `
                        <h3>${news.heading}</h3>
                        <p>${news.summary}</p>
                        <p><strong>Sentiment Score:</strong> ${news.sentiment_score}</p>
                        <a href="${news.link}" target="_blank">Read more</a>
                    `;
                    newsResults.appendChild(newsItem);
                });
            }
        });
    });
});
```

2.3 Database : The application uses SQLite as the database. The database schema includes a User table with the following fields:
id: Primary key
username: Username of the user
password: Hashed password of the user





3. Setting Up the Application

To set up the Personalized News Dashboard, I ensured I have the necessary prerequisites like Python, Flask, and NLTK installed, then followed the installation steps to clone the repository, set up a virtual environment, and installed the required dependencies.

3.1 Prerequisites : Before setting up the Personalized News Dashboard,I ensured I have Python 3.x, Flask, and NLTK installed, as well as a functional SQLite or MySQL database for storing user information.

3.2 Installation: To install the Personalized News Dashboard, clone the repository, to your local machine using git clone https://github.com/your-repo/personalized-news-dashboard.git and navigate to the project directory and create and activate a virtual environment using python -m venv venv #source venv/bin/activate # On Windows use venv\Scripts\activate and install the necessary dependencies using pip install -r requirements.txt.

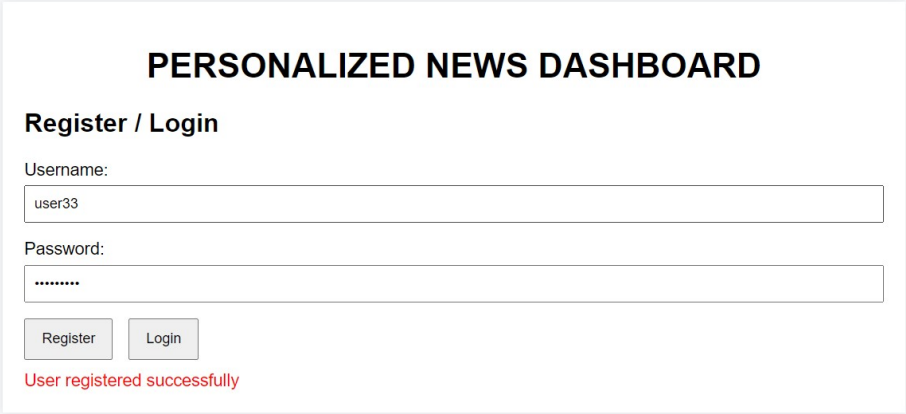
3.3 Running the Application : To run the Personalized News Dashboard, I started the Flask server by executing python run.py in my terminal, then access the application via http://localhost:5000 in my web browser.

4. Operating the Application

To operate the Personalized News Dashboard, I registered and logged in with my credentials, then selected a news category and clicked "Fetch News" to view personalized news articles along with their sentiment scores.

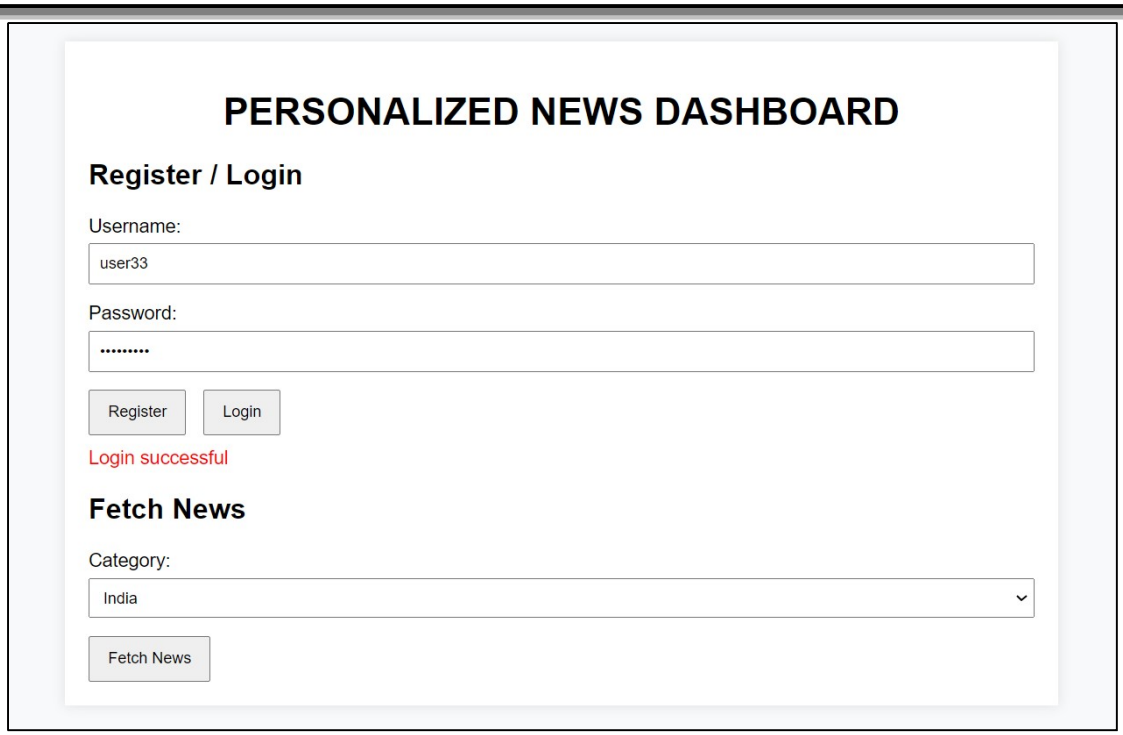
4.1 User Registration and Login

- Register:Enter a username and password in the registration form, then click "Register."
  A message will indicate whether the registration was successful.

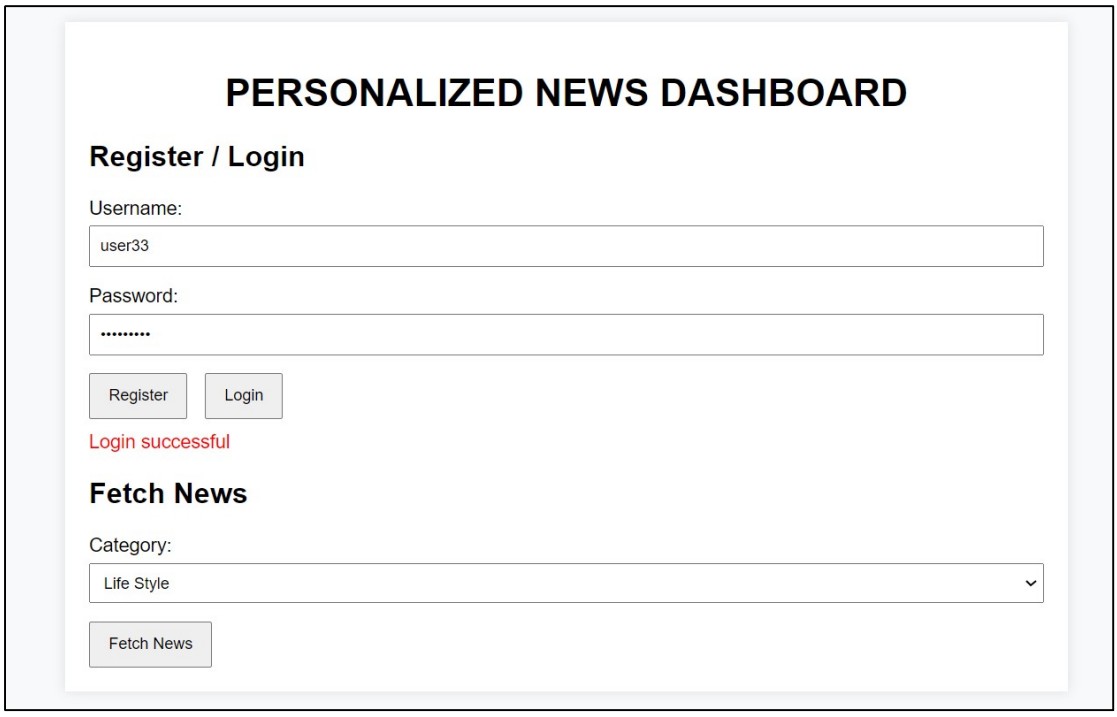


- Login: Enter a username and password in the login form, then click "Login."

On successful login, a message will appear, and the news section will be displayed.



❖ Fetching News: Select a Category:
Choose a news category from the dropdown menu.



❖ Fetch News: Click the "Fetch News" button.
The application will display news articles along with their sentiment scores.