**Website Visitor Classification**

**Challenge Statement**: Develop a tool that accepts a website URL as input, scrapes its content, and classifies users based on their interests or industry. The solution should dynamically generate questions and multiple-choice options to facilitate user categorization. This requires seamless integration of frontend and backend components, emphasizing creativity and problem-solving to deliver a robust, user-centric application. The goal is to build a scalable and innovative solution that enhances user engagement and categorization accuracy.

**Accomplishment Roadmap**

**Project Directory:**

backend/

├── app.py # Main Flask app with API endpoints

├── requirements.txt # Backend dependencies

├── config.py # Configuration settings

├── services/

│ ├── scraper.py # Web scraping logic

│ └── classifier.py # Classification logic

└── tests/

└── test\_app.py # Tests for Flask API endpoints

frontend/

├── public/

│ └── index.html # Main HTML file

├── src/

│ ├── components/

│ │ ├── InputForm.js # URL input form component

│ │ ├── Questions.js # Displays questions and options

│ ├── redux/

│ │ ├── actions.js # Redux actions

│ │ ├── reducer.js # Redux reducer

│ │ └── store.js # Redux store configuration

│ ├── App.js # Main app component

│ ├── index.js # Entry point

│ └── styles.css # CSS styles

└── package.json # Frontend dependencies

**Tools Used:**

Here’s a list of **development tools**, **testing tools**, and **other tools** used for this project:

**Development and Testing Tools:**

1. **Visual Studio Code (VSCode)**:
   * Used for writing and editing code for both the frontend (React) and backend (Python).
   * Equipped with features like IntelliSense, debugging, and extensions for enhancing development workflow.
2. **Node.js**:
   * A JavaScript runtime used for running the frontend React application and managing package dependencies.
3. **npm** (Node Package Manager):
   * Used for managing dependencies and packages for the React frontend application.
4. **React**:
   * JavaScript library for building the user interface, handling user interactions, and managing the application state using the **Context API**.
5. **Python**:
   * Used for backend processing, including content scraping, classification, and question generation logic.
6. **Flask**:
   * A Python web framework used for developing the backend API endpoints, handling HTTP requests, and serving the content.
7. **HTML/CSS**:
   * Used for designing and styling the structure of the user interface.
8. **JavaScript**:
   * Used for the client-side logic, dynamic interaction with the backend, and updating the UI.
9. **Postman**: Used for testing API endpoints, ensuring correct responses, and verifying the functionality of the backend (Flask API).
10. **Pytest** (for Python Testing): Testing framework used for unit testing the Python backend code, ensuring the business logic (content scraping, question generation) works as expected.

**Version Control:**

1. **Git**:
   * Used for version control and managing codebase changes.
2. **GitHub**:
   * Repository hosting platform for storing the project code, version tracking, and collaboration.
   * Facilitates code sharing and public access for the project.

**Optional (For Future Use with Databases):**

1. **PostgreSQL / MongoDB** (if database integration is added):
   * Databases for storing user profiles, responses, and categorization data.
2. **Redis** (if caching is implemented):
   * Used for caching frequently accessed data to speed up the application and improve performance.

The main focus has been on **React** for the frontend, **Python/Flask** for the backend, and **Postman** for testing the API. Additionally, **VSCode** was the primary development environment, and GitHub was used for version control.

**Implementation:**

1. **Integration of Frontend and Backend Components**

**Technologies Used:**

* **Frontend**: React.js, Context API, JSX
* **Backend**: Python (Flask), Machine Learning (scikit-learn, trained models)
* **Data Communication**: RESTful APIs (via Axios or fetch in React)

**Skills and Process:**

* **Frontend**:
  + **React.js** was used to build a dynamic and interactive user interface.
  + **Context API** manages state (questions, options, selections) across the app, enabling data flow from backend to frontend.
  + Components are broken down logically for reusability (e.g., Questions and Options components).
* **Backend**:
  + A **Flask API** serves as a bridge between the frontend and the machine learning model.
  + **Content Classification**: Scraped content is processed using **ML models** trained on different topics to classify user intent and provide relevant questions.
  + **API Communication**: React.js frontend makes API calls to Flask backend, which serves classified content and generates AI-driven questions dynamically.

**Impact:**

* The integration ensures smooth data flow and provides a dynamic user experience.
* Real-time question generation based on content ensures personalization and relevance for the user.

1. **Web Scraping, Data Extraction, and AI-based Content Generation**

**Technologies Used:**

* **Scraping**: BeautifulSoup, Requests (Python)
* **Data Processing**: Regular Expressions (for cleaning content)
* **AI and ML Models**: scikit-learn, Custom-trained models for content classification
* **Data Storage**: In-memory (temporary, if Redis is added later)

**Skills and Process:**

* **Web Scraping**:
  + Utilized **BeautifulSoup** and **Requests** to scrape content from websites like Wikipedia and other online sources.
  + Cleaned and pre-processed the scraped content to remove unnecessary tags, whitespace, or irrelevant data.
* **Content Classification**:
  + The scraped content is passed through **ML models** (trained with scikit-learn) to classify it into predefined categories (e.g., Health, Finance, Travel).
* **Dynamic Question Generation**:
  + Based on the classified content, dynamic questions are generated using templates with placeholders like {topic}.
  + The **options** are selected from a predefined list based on the category.

**Impact:**

* Automated extraction and classification of content allow the system to generate relevant and personalized questions based on user interactions.
* Reduces manual effort, increases accuracy, and enhances the user experience.

1. **Clean and Well-Structured Code (GitHub Repository)**

**Technologies Used:**

* **Version Control**: Git, GitHub
* **Code Structure**: Modular design, clean code practices (separation of concerns)
* **Front-end Code**: JSX, React Components, Context API
* **Backend Code**: Flask, Python scripts, ML model integration

**Skills and Process:**

* **Code Organization**:
  + The project is structured with clear separations: frontend (React) and backend (Flask) are kept in separate folders.
  + The React frontend is broken down into reusable components (Question, Option, Button, etc.).
* **Clean Code Practices**:
  + Functions and components have single, clear responsibilities, and follow **SOLID principles**.
  + Clear variable names, consistent coding styles, and comments enhance readability and maintainability.
* **Version Control**:
  + The project is version-controlled on **GitHub**, where code changes are tracked, and progress can be easily monitored.

**Impact:**

* Ensures maintainability and scalability of the project, making it easier to expand or modify in the future.
* With GitHub integration, collaboration and version tracking are optimized for team development.

1. **Use of Databases and Relevant Technologies**

**Clarifying the Current Implementation:**

1. Current Focus:
   * The application dynamically generates questions and multiple-choice options based on content classification.
   * It categorizes users based on the content they engage with, which helps personalize the experience.
2. No Database Implementation Yet:
   * Currently, the project doesn't involve a database for user management or storing user interactions (like responses, selections, etc.).
   * The dynamic question generation process works at runtime and doesn't rely on stored user data for categorization.

**How This Current Approach Helps in Categorizing Users:**

**Content-based Categorization:**

* The content classification model (AI-based) classifies a webpage’s content into specific topics (e.g., Health, Finance, Travel).
* Based on this classification, relevant questions and multiple-choice options are generated dynamically for the user.
* User Categorization: The options selected by the user are then used to categorize them into predefined categories (such as Fitness Enthusiasts, Finance Seekers, etc.). This categorization helps tailor the experience.

For example, if a user selects options related to Health and Fitness, they would be categorized under a Fitness Enthusiast group, and the questions they receive would be more focused on that area.

**Impact of Dynamic Content:**

* The categorization happens in real-time, based on the user’s choices from the dynamically generated questions.
* This approach doesn’t require a database for this phase because the selection and categorization logic can be handled on-the-fly in the frontend (React) with state management (Context API).

**How Databases Could Help in Future Iterations:**

* 1. **User Profiles**: Store individual preferences, names, and historical interactions for personalization.
  2. **Historical Data**: Track past responses to improve future question relevance.
  3. **Categorization Storage**: Store final user categorization and provide tailored content.
  4. **Caching with Redis**: Speed up response times by storing frequently accessed data.
  5. **Behavior Analysis**: Analyze patterns to refine categorization and improve user engagement.

**How It Can Be Implemented:**

1. **Database Schema**: Design tables for users, responses, questions, and categories.
2. **Backend API**: Use Flask/Django to manage user data and responses.
3. **CRUD Operations**: Implement APIs to handle user profiles and responses.
4. **Frontend Update:** Modify React to send user choices to backend.
5. **Persistent Tracking:** Store user selections in database for future categorization.