Echo - Our Shareable News | News Aggregator Application for the Progressive Left

Project Description: A web-based news aggregator that uses a community-driven voting system to rank articles by importance.

Tech Stack

Frontend Framework: Next.js
Backend Language and Framework: Python with FastAPI
Database Management System: PostgreSQL
ServerOS: Ubuntu
HostingPlatform: GCP

Functional Requirements

1. **Submit News Articles**
    a. Users can submit news articles via a URL or manual input.
    b. Articles must include a title, description, and category/tagging information.

Enhancements:

    c. Allow users to preview the article (e.g., fetch metadata like title and image from the URL).
    d. Enable moderation (e.g., admin approval or community flagging system) to maintain content quality.

2. **Voting System**
    a. Users can upvote or downvote articles.
    b. Aggregate scores determine popularity and ranking.

Enhancements:

    c. Implement time-decay for votes (e.g., newer articles weigh more heavily).

d. Add "reaction" features (e.g., likes, loves, etc.) to encourage user engagement.

3. **View Most Popular Articles**
    a. A "trending" section showcases the top-voted articles.
    b. Filters for categories, tags, or timeframes (e.g., trending this week, this month).

Enhancements:

c. Add personalized recommendations based on user preferences or voting history.
d. Display a "Most Controversial" section (e.g., articles with a mix of upvotes and downvotes).

4. **Categorization/Tagging**
    a. Articles can be categorized into predefined topics (e.g., politics, technology, sports).
    b. Users or admins can assign tags for better searchability.

Enhancements:

c. Introduce dynamic, user-generated tags with a suggestion system based on existing tags.
d. Use natural language processing (NLP) to auto-suggest categories and tags.

5. **User Profiles**
    a. Users have profiles showing their submitted articles, votes, and activity history.

Enhancements:

b. Add user badges or reputation scores to encourage participation.
c. Enable user preferences for content (e.g., favorite categories or notification settings).

6. **Search Functionality**
    a. Users can search for articles using keywords, categories, or tags.

Enhancements:

b. Include filters for sorting by popularity, recency, or relevance.
c. Add an advanced search feature for Boolean queries (e.g., "technology AND AI").
d. Include typo correction and suggestions for queries.

7. **Notifications**
    a. Users receive notifications for new votes on their articles or replies to their comments.

Enhancements:

b. Add push notifications for trending articles or updates on followed categories.
c. Allow users to customize notification settings (e.g., email, push, in-app).

V2 Functional Requirements

1. **Social Features**
    a. Enable users to comment on articles and reply to others' comments.
    b. Include social sharing buttons for platforms like Twitter or Facebook.
2. **Moderation Tools**
    a. Provide admins with tools for banning users, removing inappropriate content, or handling flagged posts.
    b. Allow community-driven moderation (e.g., flagging articles or comments for review).
3. **Analytics**
    a. Provide users with insights (e.g., how many views, votes, or shares their articles have received).
    b. Add site-wide metrics for admins (e.g., most active users, popular categories).

4. **Dark Mode**
    a. Include a dark mode toggle for a better user experience across devices.
5. **Mobile App**
    a. Build a mobile-friendly version or a native app for on-the-go browsing and interaction.
6. **Accessibility**
    a. Ensure the app meets accessibility standards (e.g., WCAG compliance for screen readers).
7. **Monetization**
    a. Include options for monetization, such as sponsored articles or ad placements.
    b. Allow users to purchase premium features (e.g., ad-free browsing or enhanced profile customization).
8. **Scalability**
    a. Use pagination or infinite scrolling for article feeds.
    b. Employ caching for frequently accessed pages or queries to improve performance.
9. **Security**
    a. Implement user authentication with multi-factor authentication (MFA).
    b. Ensure proper sanitization of user inputs to prevent vulnerabilities (e.g., SQL injection, XSS).
10. **Backup and Recovery**
    a. Set up automated backups to prevent data loss.
    b. Provide a recovery option for accidentally deleted content (e.g., articles, tags).

Non-Functional Requirements

1. High Traffic Loads

- **Existing Requirement**: Handle high traffic loads without significant degradation in performance.

- **Enhancements**:
    - **Load Testing**: Regularly perform load and stress tests using tools like Apache JMeter, Gatling, or Locust to ensure the system can handle peak traffic.
    - **Content Delivery Network (CDN)**: Use a CDN (e.g., Cloudflare, AWS CloudFront) to cache static assets and reduce server load.
    - **Autoscaling**: Implement autoscaling to dynamically add resources during traffic spikes.
    - **Queue Management**: Use message queues (e.g., RabbitMQ, Kafka) for asynchronous tasks like ranking updates or notifications.

2. Data Integrity

- **Existing Requirement**: Ensure the integrity and accuracy of data stored within the database.
- **Enhancements**:
    - **Transactional Database**: Use a database with ACID compliance (e.g., PostgreSQL, MySQL) for critical operations like voting and comments.
    - **Validation**: Implement robust input validation at both the frontend and backend to prevent corrupt or malformed data.
    - **Backup and Restore**: Set up automated backups with point-in-time recovery to prevent data loss in case of failures.

3. Security

- **Existing Requirement**: Meet strict security standards to protect sensitive user information.
- **Enhancements**:
    - **Encryption**:
        - Use TLS (SSL) for encrypting data in transit.
        - Encrypt sensitive data at rest, such as user passwords (using strong hashing algorithms like bcrypt).
    - **Authentication & Authorization**:
        - Implement secure user authentication, such as OAuth 2.0 or OpenID Connect.

        - Use role-based access control (RBAC) to restrict access to sensitive features or data.
    - **Vulnerability Scanning**: Regularly scan for vulnerabilities using tools like OWASP ZAP or Burp Suite.
    - **Rate Limiting**: Prevent abuse or DDoS attacks by rate-limiting API requests.
    - **Security Headers**: Use HTTP security headers (e.g., Content Security Policy, X-Frame-Options).

4. Scalability

- **Existing Requirement**: Built using scalable technologies, allowing for efficient expansion as the user base grows.
- **Enhancements**:
    - **Microservices Architecture**: Break down the application into modular services to scale individual components independently.
    - **Database Scaling**:
        - Use read replicas to distribute database read queries.
        - Consider sharding for very large datasets.
    - **Caching**:
        - Cache frequently accessed data using Redis or Memcached to reduce load on the database.

5. User Experience

- **Existing Requirement**: Provide an intuitive and visually appealing user experience.
- **Enhancements**:
    - **Responsive Design**: Ensure the UI is optimized for all devices and screen sizes.
    - **Performance Optimization**: Minimize UI load times using techniques like lazy loading, image optimization, and efficient use of JavaScript.
    - **Accessibility**: Comply with accessibility standards (e.g., WCAG 2.1) to make the application usable for people with disabilities.

6. Performance Metrics

- **Existing Requirement**:
  - Respond to requests within 500ms.
  - Update rankings regularly, ideally every few seconds.
- **Enhancements**:
  - **Monitoring Tools**: Use tools like Prometheus, Grafana, or New Relic to monitor response times and system health.
  - **Asynchronous Processing**: Use asynchronous updates for ranking calculations to avoid blocking user requests.
  - **Database Indexing**: Optimize database queries with appropriate indexing for frequent operations like voting or fetching rankings.

7. Availability

- **Existing Requirement**: Achieve 99.98% uptime (maximum downtime: ~1.44 hours/year).
- **Enhancements**:
  - **Redundant Infrastructure**: Use multi-zone or multi-region deployments to mitigate failures in a single data center.
  - **Health Checks and Failover**: Implement automated health checks and failover mechanisms for servers and databases.
  - **Disaster Recovery Plan**: Define and test a comprehensive disaster recovery strategy to handle major failures.

Suggested Additions

1. **Logging and Auditing**:
   a. Implement detailed logging for all system activities to help with debugging and tracking malicious behavior.
   b. Maintain an audit trail for critical operations like vote changes or user role updates.
2. **APIs and Extensibility**:
   a. Ensure the backend APIs are RESTful and well-documented, enabling integration with third-party systems or mobile apps.
3. **Energy Efficiency**:

    a. Optimize server usage and consider green hosting providers to reduce the environmental impact of running the application.
4. **Compliance**:
    a. Ensure compliance with data protection laws like GDPR (EU) or CCPA (California) if you collect user data.
5. **Alerts and Notifications**:
    a. Set up automated alerts for downtime, performance degradation, or security threats to respond quickly.

Backend: 1. Create a new file called main.py in the backend directory. This will serve as the entrypoint for our Flask-like FastAPI application. 2. Open the main.py file and import the FastAPI library. 3. Create a new FastAPI object and define routes for our API. 4. Save the file and close it.

Frontend: 1. Navigate to the frontend directory and create a new file called index.js. 2. Open the index.js file and import the necessary libraries (e.g. react-dom, react). 3. Create a new React component and render it to the DOM. 4. Save the file and close it.

Database 1. Navigate to the database directory and create a new file called schema.sql. 2. Open the schema.sql file and define the tables for our PostgreSQL database. 3. Save the file and close it.

Docs 1. Navigate to the docs directory and create a new file called README.md. 2. Open the README.md file and document the purpose of the echo app, the technology stack, and instructions for installation and deployment. 3. Save the file and close it.