**1. Enhance Configuration and Security**

**The top priority is to improve how configuration and secrets are handled to align with best practices.**

* **Centralize the Secret Key: The Flask app.secret\_key is currently hardcoded in**[**app.py**](code-assist-path:c:\Users\raman\OneDrive\Documents\Raman\17.%20News\3.%20NewsReader\app.py)**. This should be moved into your**[**config.py**](code-assist-path:c:\Users\raman\OneDrive\Documents\Raman\17.%20News\3.%20NewsReader\config.py)**and loaded from an environment variable, just like your other secrets. This is a critical step to avoid exposing sensitive keys in your source code.**
* **Organize Constants: Move the AI MODEL\_NAME from**[**ai\_utils.py**](code-assist-path:c:\Users\raman\OneDrive\Documents\Raman\17.%20News\3.%20NewsReader\ai_utils.py)**into**[**config.py**](code-assist-path:c:\Users\raman\OneDrive\Documents\Raman\17.%20News\3.%20NewsReader\config.py)**to keep all external service configurations in one place.**

**2. Refactor for Better Database Management**

**Your current direct use of sqlite3 works, but adopting an Object-Relational Mapper (ORM) will make your code cleaner, more secure, and easier to maintain as the application grows.**

* **Introduce an ORM: Integrate SQLAlchemy with its Flask extension, Flask-SQLAlchemy. This will allow you to define your database tables as Python classes (models) and interact with them using Python objects instead of writing raw SQL queries. This reduces boilerplate code and helps prevent SQL injection vulnerabilities.**
* **Implement Database Migrations: Replace the custom**[**database\_setup.py**](code-assist-path:c:\Users\raman\OneDrive\Documents\Raman\17.%20News\3.%20NewsReader\database_setup.py)**script with a dedicated migration tool like Flask-Migrate (which uses Alembic). This provides a robust, version-controlled system for managing schema changes, which is essential for long-term maintenance.**

**3. Improve Code Structure and Reusability**

**Refactoring parts of the main application file will reduce code duplication and improve readability.**

* **Create Data Access Helpers: You have several routes that fetch user data from the database. This logic can be centralized into helper functions (e.g., get\_user\_by\_email(email)). This makes the routes cleaner and ensures user data is fetched consistently.**
* **Break Down Large Routes: The display\_news and reset\_password routes in**[**app.py**](code-assist-path:c:\Users\raman\OneDrive\Documents\Raman\17.%20News\3.%20NewsReader\app.py)**contain a significant amount of logic. This logic can be extracted into smaller, single-purpose functions to make the routes themselves easier to read and test.**

**4. Boost Performance and Scalability**

**As your user base grows, the daily newscast generation script could become a bottleneck.**

* **Parallelize Newscast Generation: The**[**newscast\_generator.py**](code-assist-path:c:\Users\raman\OneDrive\Documents\Raman\17.%20News\3.%20NewsReader\newscast_generator.py)**script processes users one by one. For a large number of users, this will be slow. I recommend modifying the script to process users in parallel using Python's multiprocessing module. For a production-grade system, you might consider a task queue like Celery.**
* **Implement Media File Cleanup: Both the main app and the generator script create audio and video files without a mechanism to remove old ones. This will eventually consume a lot of disk space. A cleanup strategy should be implemented, either as a separate scheduled job or as part of the newscast generator, to delete files older than a certain number of days.**
* **Refine Session Management: Storing large data blobs like combined\_summaries in the client-side session cookie can be inefficient and has size limits. Consider switching to a server-side session extension (like Flask-Session) which stores session data on the server and only sends a session ID to the client.**