Design Decisions

Flask for Web Framework: Chose Flask due to its simplicity, flexibility, and ease of use for building RESTful APIs. Flask provides just the right amount of functionality needed for this project without unnecessary overhead.
Blueprints for Modularity: Organized routes into Blueprints to keep the codebase modular and maintainable. This allows for better separation of concerns and easier collaboration among team members.
JWT for Authentication: Implemented JWT (JSON Web Tokens) for token-based authentication. JWT provides stateless authentication, allowing scalability and flexibility in handling authentication across multiple services or microservices.
MongoDB for Database: Selected MongoDB as the database due to its flexibility and scalability. MongoDB's document-oriented model suits the needs of a blog application well, allowing for easy storage and retrieval of blog posts.
Hashing Passwords: Used SHA-256 hashing for password storage to enhance security. Storing passwords in hashed form adds an extra layer of protection against data breaches.

Trade-offs Made

Simplicity vs. Features: Prioritized simplicity and ease of understanding in the codebase over adding advanced features. While this approach keeps the codebase clean and maintainable, it may limit the functionality of the application compared to more feature-rich solutions.
Database Choice: Opted for MongoDB for its flexibility and ease of use, but this decision may limit the scalability of the application in the long term compared to more robust relational databases like PostgreSQL.
Error Handling: Error handling is implemented, but it may not cover all edge cases. Additional error handling and validation could be added to improve robustness, but this would increase code complexity.

Additional Features or Improvements

Pagination: Implement pagination for fetching blog posts to improve performance and user experience, especially as the number of posts grows.
Email Verification: Add email verification for user sign-up to ensure the validity of user accounts and reduce the risk of spam or fake accounts.
Logging and Monitoring: Integrate logging and monitoring tools to track application performance, identify errors, and troubleshoot issues more effectively.
Unit Testing: Expand unit test coverage to ensure the reliability and correctness of the codebase, especially for critical components like authentication and data manipulation.
Rate Limiting: Implement rate limiting to prevent abuse or misuse of API endpoints and protect against potential denial-of-service attacks.