Introduction:

Designed to optimize route planning and mitigate risks, our system leverages real-time data and advanced algorithms to ensure safe and efficient flight operations. This documentation provides comprehensive insights into the system's functionalities, API endpoints, deployment instructions, and more.

Technical Documentation:

System Overview:

Our flight navigation enhancement system is built using a modern tech stack, comprising MySQL for data storage, Express.js for backend server development, React.js for frontend user interface, and Node.js for server-side JavaScript execution. The system is deployed using Vercel for frontend deployment and Render for backend deployment, ensuring reliability and scalability.

Technologies Used:

- MySQL: Relational database management system for data storage, offering robustness and data integrity.
- Express.js: Web application framework for Node.js, facilitating backend server development with simplicity and flexibility.
- React.js: JavaScript library for building user interfaces, providing a fast and interactive frontend experience.
- Node.js: JavaScript runtime environment for executing server-side code, enabling efficient handling of concurrent requests.

Dijkstra Algorithm:

- Purpose: Utilized for finding the shortest path in route planning.
- Implementation: Calculates optimal routes considering real-time weather and flight conditions.
- Benefits: Ensures efficient and safe navigation by dynamically adjusting routes based on current data.

Deployment:

- Frontend Deployment: The frontend application is deployed on Vercel, offering seamless deployment and hosting with built-in CI/CD.
- Backend Deployment: The backend server is deployed on Render, providing a scalable and reliable hosting platform with automatic scaling and monitoring capabilities.

Data Flow:

- Data flows from the frontend user interface to the backend server through RESTful API endpoints.
- Backend server processes requests, retrieves data from the MySQL database, and sends responses back to the frontend.
- Real-time data updates and notifications are facilitated through WebSocket connections between the frontend and backend.

Database Schema:
- The MySQL database comprises multiple tables for storing various data entities, including intermediate cities, flight information, weather data etc.
- Tables are normalized to 3nf to minimize redundancy and ensure data consistency, with appropriate indexes and constraints applied for optimized performance.

Monitoring and Logging:
- System performance and error logs are monitored and logged using built-in logging frameworks provided by Express.js and Node.js.

Scalability and Performance:
- The system architecture is designed for horizontal scalability, allowing for seamless handling of increasing user loads and data volumes.
- Performance optimizations, such as caching and query optimization, are implemented to ensure rapid response times and efficient resource utilization.

API Links:
Our flight navigation enhancement system provides a set of RESTful APIs for interacting with various components of the system. This documentation covers the available endpoints, request and response formats, authentication methods, and error handling.

Base URL:
`https://flight-navigation-backend.onrender.com`

Endpoints:

1. Airport Distance API
  - GET https://airportgap.com/api/airports/distance
    - Description: Retrieves the distance between two airports.
    - Request Parameters:
      - `from` (string): The IATA code of the departure airport.
      - `to` (string): The IATA code of the destination airport.

2. Route Planning API
  - GET /api/getRoute
    - Description: Retrieves the optimal route for a flight.
    - Request Parameters:
      - `from` (string): The IATA code of the departure airport.
      - `to` (string): The IATA code of the destination airport.

3. Location API
  - GET /api/getlocation
    - Description: Retrieves the current location of the flight.
    - Request Parameters:

- `flight_id` (string): The unique identifier of the flight.

4. Insert Airport API
  - POST /api/insertAirport
    - Description: Inserts a new airport into the database.
    - Request Body Parameters:
      - `iata` (string): The IATA code of the airport.
      - `name` (string): The name of the airport.
      - `city` (string): The city where the airport is located.
      - `country` (string): The country where the airport is located.

5. Insert Destination Airport API
  - POST /api/insertDestinationAirport
    - Description: Inserts a new destination airport into the database.
    - Request Body Parameters:
      - `iata` (string): The IATA code of the airport.
      - `name` (string): The name of the airport.
      - `city` (string): The city where the airport is located.
      - `country` (string): The country where the airport is located.

6. Flight Status API
  - GET /api/getFlyStatus
    - Description: Retrieves the flight status.
    - Request Parameters:
      - `flight_id` (string): The unique identifier of the flight.

7. Nearest Airport API
  - GET /api/get-nearest-airport
    - Description: Finds the nearest airport based on current location.
    - Request Parameters:
      - `latitude` (float): Current latitude.
      - `longitude` (float): Current longitude.

8. Nearby Airports API
  - GET /api/near
    - Description: Retrieves nearby airports within a specified radius.
    - Request Parameters:
      - `latitude` (float): Latitude.
      - `longitude` (float): Longitude.
      - `radius` (int): Radius in kilometers.

9. Airport Graphs API
  - GET /api/airport-graphs
    - Description: Retrieves graph data for airport connections.

10. External Airport Info API
  - GET https://airport-info.p.rapidapi.com/airport
    - Description: Retrieves detailed airport information from an external API.
    - Request Parameters:
      - `iata` (string): The IATA code of the airport.

11. Flight Fuel API
  - GET /api/flight-fuel
    - Description: Retrieves fuel consumption details for a flight.
    - Request Parameters:
      - `flight_id` (string): The unique identifier of the flight.

  - POST /api/insertFlightFuel
    - Description: Inserts fuel data for a flight into the database.
    - Request Body Parameters:
      - `flight_id` (string): The unique identifier of the flight.
      - `fuel_used` (int): The amount of fuel used.
      - `fuel_remaining` (int): The amount of fuel remaining.

12. External Location Data API
  - GET https://test.api.amadeus.com/v1/reference-data/locations
    - Description: Retrieves location data from an external API.
    - Request Parameters:
      - `keyword` (string): Search keyword.

13. Aircraft API
  - GET /api/aircraft
    - Description: Retrieves details of aircrafts in the system.

14. Routes API
  - GET /api/routes
    - Description: Retrieves available flight routes.

Authentication:
- API Key: Each request must include an API key in the headers for authentication.
  - Header: `Authorization: Bearer YOUR_API_KEY`

Rate Limiting and Usage Policies:
- Each API key is subject to rate limiting to prevent abuse.
- The default rate limit is 1000 requests per hour.

Error Handling:
- 400 Bad Request: The request is invalid or malformed.

- 401 Unauthorized: The API key is missing or invalid.
- 404 Not Found: The requested resource does not exist.
- 500 Internal Server Error: An unexpected error occurred on the server.