**Flow of Operations**

1. **User Interaction**:
   * Users access the web interface through a browser. The frontend renders the schedule using the data provided by the Flask application.
2. **Request Handling**:
   * When a user visits the main route (/), the Flask application processes the request by initializing the Harmony Search algorithm with the nurse scheduling configuration.
3. **Optimization**:
   * The Harmony Search algorithm runs to generate an optimal schedule based on constraints and preferences. This involves:
     + Generating a random initial population of schedules.
     + Iteratively improving schedules through improvisation, evaluation, and updating the harmony memory.
4. **Database Operations**:
   * The optimized schedule is saved to the SQLite database using SQLAlchemy. The database schema includes tables for days, shifts, nurses, and schedules.
5. **Schedule Display**:
   * The final schedule is formatted and rendered on the web page, allowing users to view the allocated shifts and nursing staff for each day.

**Workflow Diagram Description**

1. **User Accesses Web Interface**
   * **Action**: User navigates to the website.
   * **Outcome**: The main page is loaded.
2. **Flask Application Receives Request**
   * **Action**: Flask application receives a GET request for the main route (/).
   * **Outcome**: Flask processes the request and calls the necessary functions.
3. **Initialize Harmony Search Algorithm**
   * **Action**: Flask initializes the Harmony Search algorithm with the nurse scheduling configuration.
   * **Outcome**: Harmony Search object is created.
4. **Generate Initial Schedule**
   * **Action**: Harmony Search generates a random initial population of schedules.
   * **Outcome**: Initial schedules are stored in harmony memory.
5. **Iterative Optimization**
   * **Action**: The algorithm iteratively improves the schedules through:
     + **Improvisation**: Generating new schedules based on existing ones.
     + **Evaluation**: Assessing the quality of each schedule.
     + **Updating Memory**: Replacing the worst schedules with better ones.
   * **Outcome**: The best schedule is determined after a set number of iterations.
6. **Save Optimized Schedule to Database**
   * **Action**: Flask saves the optimized schedule to the SQLite database using SQLAlchemy.
   * **Outcome**: The schedule is stored in the database.
7. **Render Schedule on Web Page**
   * **Action**: Flask retrieves the schedule data from the database.
   * **Outcome**: Schedule data is sent to the front end.
8. **User Views Schedule**
   * **Action**: The web page renders the schedule using HTML, CSS, and JavaScript.
   * **Outcome**: The user can view the optimized schedule.