Project

Basic Project outline

DATA REQUIRED TO GENERATE TIMETABLE

To generate a timetable, the following data is required:

- Staff details (name, department, contact information)
- Staff availability (preferred working hours, days available)
- Subjects each staff member can teach
- Classroom details (room number, capacity)
- Allocated subjects for each department and batch
- Constraints (crossover of staff and hours, consecutive classes)

1. **Classroom Availability:**
    - Specify the availability of each classroom, including any special equipment or features.
    - Consider any restrictions on the types of classes that can be held in specific classrooms.
2. **Student Group Information:**
    - Include information about student groups, their sizes, and the courses they are enrolled in.
    - Consider any constraints related to specific groups, such as special accommodations or preferences.
3. **Special Events or Constraints:**
    - Account for any special events, holidays, or constraints that might affect the regular schedule.
    - Include any specific constraints provided by the institution or regulatory requirements.
4. **Break Preferences:**
    - Allow staff members to specify preferences for breaks during the day.
    - Consider rules for mandatory breaks or time off between consecutive classes.

5. **Optimization Criteria:**
    - Define optimization criteria for the algorithm, such as minimizing travel time between classrooms for staff or maximizing room utilization.
6. **Priority Rules:**
    - Establish priority rules for certain classes or staff members that must be accommodated first.
    1. **Fallback Options:**
        - Plan for alternative options in case the initial timetable is not feasible or conflicts with constraints.
        - Implement a mechanism to suggest alternative schedules based on user preferences.
    - Consider any rules related to specific courses or departments.
7. **User Feedback Mechanism:**
    - Provide a way for users to review and provide feedback on the generated timetable.
    - Allow for adjustments or manual modifications based on user input.

CONSTRAINTS TO BE CONSIDERED

- Crossover of staff and hour: Ensure that staff members do not have overlapping schedules.
- Consecutive classes for a staff: Minimize consecutive classes for a staff member to avoid fatigue.

The constraints you've outlined are a good starting point, covering the basics to ensure a functional and practical timetable. However, depending on the specific requirements and policies of the university, you might consider additional constraints to tailor the timetable generation more precisely. Here are some suggestions:

1. **Maximum Teaching Hours per Day:**
    - Define a maximum limit for the number of teaching hours a staff member can have in a single day to prevent burnout.
2. **Maximum Teaching Hours in a Row:**
    - Limit the number of consecutive teaching hours without a break to ensure that staff members have time for rest or preparation.
3. **Preferred Time Windows:**
    - Allow staff members to specify preferred time windows for classes, accommodating their peak performance times.
4. **Subject-Specific Constraints:**
    - Incorporate constraints specific to certain subjects or types of classes, considering any unique requirements or preferences.
5. **Room-Specific Constraints:**
    - Consider constraints related to specific rooms, such as equipment availability, capacity, or restrictions on certain types of classes.
6. **Day-Off Requests:**
    - Allow staff members to request specific days off or indicate days when they prefer not to have classes.

7. **Avoiding Back-to-Back Labs:**
    - If applicable, avoid scheduling laboratory sessions back-to-back to allow time for setup and cleanup.
8. **Balance Workload:**
    - Distribute the workload evenly among staff members, ensuring that no one is significantly overburdened compared to others.
9. **Preferential Treatment for Senior Staff:**
    - Consider any policies that give preference or priority to senior staff members in terms of scheduling.
10. **Ensuring Resource Availability:**
    - Check for resource availability, such as projectors or specialized equipment, to avoid clashes in resource usage.

Remember, the constraints should align with the goals and policies of the university, providing a balance between the preferences of staff members, the efficient use of resources, and the overall needs of the institution.

TECHNICAL IMPLEMENTATION MAP

- **Frontend:**
  - Landing Page
  - Registration and Login (for institution, staff, and class representatives)
  - Campus Incharge Dashboard
  - Teacher Dashboard
  - Class Representative Dashboard
- **Backend:**
  - User Authentication and Authorization
  - Data Validation and Storage
  - Timetable Generation Algorithm
  - Notification System (preferably using WhatsApp API)
  - Database Management
  - Report Generation

Certainly! Let's elaborate on the technical implementation map:

Frontend:

1. **Landing Page:**
   - Design an attractive and informative landing page that introduces the Dynamic Timetable Generator.
   - Include key features, benefits, and a call-to-action for users to register or log in.
2. **Registration and Login:**
   - Create user-friendly registration and login forms for institutions, staff, and class representatives.
   - Implement validation checks for email formats, password strength, and other necessary fields.

- Use secure authentication mechanisms, such as bcrypt for password hashing.

3. **Campus Incharge Dashboard:**
    - Develop a dashboard where campus incharges can:
        - Input and manage campus-specific details.
        - Allocate classrooms to different batches.
        - Allocate staff to subjects and classes.
4. **Teacher Dashboard:**
    - Build a dashboard for teachers that allows them to:
        - View their timetable with class details.
        - Request swaps, substitutions, or leave.
        - Accept or reject swap/substitution requests.
        - Set preferences and availability.
5. **Class Representative Dashboard:**
    - Create a dashboard for class representatives to:
        - View class-specific timetables.
        - Receive notifications about changes.
        - Communicate with teachers regarding class preferences.

Backend:

1. **User Authentication and Authorization:**
    - Implement a secure authentication system with role-based access control (RBAC).
    - Verify user roles (institution, staff, class representative) to determine access levels.
2. **Data Validation and Storage:**
    - Validate and sanitize user inputs to prevent security vulnerabilities (SQL injection, cross-site scripting).

- Store user and timetable data in a relational database (e.g., MySQL, PostgreSQL).

3. **Timetable Generation Algorithm:**
    - Develop the timetable generation algorithm using the specified constraints and optimization criteria.
    - Ensure the algorithm is scalable and able to handle a large number of users and classes.
4. **Notification System (preferably using WhatsApp API):**
    - Integrate the WhatsApp API to send notifications to staff and class representatives.
    - Implement a notification system that alerts users about timetable changes, swap/substitution requests, and other important updates.
5. **Database Management:**
    - Create a well-structured database schema that stores user details, timetable data, and other relevant information.
    - Optimize database queries for efficient data retrieval.
6. **Report Generation:**
    - Design a reporting system that allows administrators to generate reports on various aspects of the timetable.
    - Include features to export reports in common formats (PDF, CSV) for easy sharing and analysis.

By implementing these components, you create a robust system that ensures a seamless user experience while effectively managing the complexities of timetable generation and communication within the university community.

DATABASE DESIGN:

- **Tables:**
  - Users (user_id, username, password, role)
  - Staff (staff_id, name, department, contact_info)
  - Subjects (subject_id, subject_name)
  - Timetable (timetable_id, day, hour, class, staff_id, subject_id)
  - Notifications (notification_id, sender_id, receiver_id, message, status)

The proposed database design is a good foundation for the Dynamic Timetable Generator. However, the completeness and effectiveness of the model depend on several factors, including the specific requirements of your application and the complexity of interactions among different entities. Here are some considerations and potential enhancements:

1. **Additional Attributes:**
   - Consider adding more attributes to entities based on specific requirements. For example, in the `Staff` table, you might include attributes like `max_hours_per_day` or `preferred_days_off`.
2. **Relationships:**
   - Ensure that relationships between tables are well-defined. For instance, establish foreign key relationships between the `Timetable` table and the `Users`, `Staff`, and `Subjects` tables.
3. **Normalization:**
   - Normalize the database schema to minimize redundancy and improve data integrity. This involves breaking down tables into smaller, related tables to eliminate data duplication.
4. **Constraints and Triggers:**
   - Implement database constraints to enforce data integrity rules. For example, ensure that a staff member cannot be assigned to teach a class during a time they specified as unavailable.

- Use triggers to automate certain actions based on changes in the database, such as updating a status in the `Notifications` table when a notification is read.

5. **Indexes:**
   - Consider adding indexes to columns that are frequently used in search operations. Indexing can significantly improve query performance.
6. **User Preferences:**
   - If the system needs to store more complex user preferences, consider creating additional tables or JSON fields to accommodate this information.

Here's an enhanced version:

Tables:

- *Users (user_id, username,

password, role, institution_id, created_at, updated_at)**

- *Additional Attributes:* institution_id for associating users with specific institutions, timestamps for record management.
- **Staff (staff_id, user_id, name, department, contact_info, max_hours_per_day, preferred_days_off, created_at, updated_at)**
  - *Additional Attributes:* max_hours_per_day for setting the maximum teaching hours, preferred_days_off for indicating preferred days off, timestamps for record management.
  - *Relationships:* user_id as a foreign key referencing Users.
- **Subjects (subject_id, subject_name, created_at, updated_at)**
  - *Attributes:* timestamps for record management.
- **Timetable (timetable_id, day, hour, class, staff_id, subject_id, created_at, updated_at)**
  - *Attributes:* timestamps for record management.

        - *Relationships:* staff_id as a foreign key referencing Staff, subject_id as a foreign key referencing Subjects.
    - **Notifications (notification_id, sender_id, receiver_id, message, status, created_at, updated_at)**
        - *Attributes:* timestamps for record management.
        - *Relationships:* sender_id and receiver_id as foreign keys referencing Users.

    This enhanced model takes into account additional attributes, normalization, and relationships to create a more comprehensive and maintainable database structure. Remember to adapt the design based on the specific needs and requirements of your application.

AGILE PROJECT PLAN

- **Sprint 1 (January):**
    - Frontend and Backend Setup
    - User Registration and Authentication
    - Database Schema Design
- **Sprint 2 (February):**
    - Staff and Campus Incharge Dashboard
    - Input Data Collection Forms
    - Timetable Generation Algorithm
- **Sprint 3 (March):**
    - Notification System Integration
    - Testing and Debugging
    - Documentation and Report Generation

TIMETABLE GENERATION ALGORITHM(GENETIC ALGORITHM + CSP)

1. **Initialization:**
    - Generate an initial population of timetables randomly.
2. **Evaluation:**
    - Evaluate each timetable based on constraints and preferences.
3. **Selection:**
    - Select the best-fit timetables based on evaluation scores.
4. **Crossover:**
    - Combine schedules of two parents to create new timetables.
5. **Mutation:**
    - Make small random changes to some timetables.
6. **Replacement:**
    - Replace the old population with the new one.
7. **Convergence Check:**
    - Check if the algorithm converges; if not, repeat from step 2.

IMPLEMENTATION FOR BEGINEERS

- **Frontend:**
  - Use HTML, CSS, and JavaScript for the web interface.
  - Utilize a frontend framework like React or Vue.js for dynamic UI.
- **Backend:**
  - Choose a backend language like Python (Django or Flask), Node.js (Express), or Ruby (Ruby on Rails).
  - Use a relational database (e.g., MySQL, PostgreSQL) for data storage.
- **Timetable Algorithm:**
  - Implement the genetic algorithm and CSP using a programming language of choice (e.g., Python).
  - Leverage existing libraries or frameworks that support genetic algorithms.
- **Notification System:**
  - Integrate a messaging API (such as Twilio for WhatsApp) for sending notifications.
- **Agile Tools:**
  - Use project management tools like Trello or Jira for task tracking.
- **Documentation:**
  - Create comprehensive documentation using tools like Markdown or Google Docs.

Remember to continuously test and gather feedback from potential users throughout the development process.