**GameTime - Detailed Design**

**Submitted by: Amit Donel, Ofek Weiss, Shoam Tomer**

**Part 1: *Architecture***

**MVC Architecture**

The GameTime project follows the Model-View-Controller (MVC) architecture to ensure modularity, scalability, and ease of maintenance.

* **Model**:
  + Manages user profiles, game details, and location data.
  + Handles data retrieval, storage, and processing using PostgreSQL.
* **View**:
  + Implements the user interface using Flutter for both mobile and web platforms. (Web - Optional)
  + Displays game details, player profiles, and maps.
* **Controller**:
  + Acts as a bridge between the Model and View.
  + Processes user inputs and updates the Model and View accordingly.

**Part 2: *Data Description***

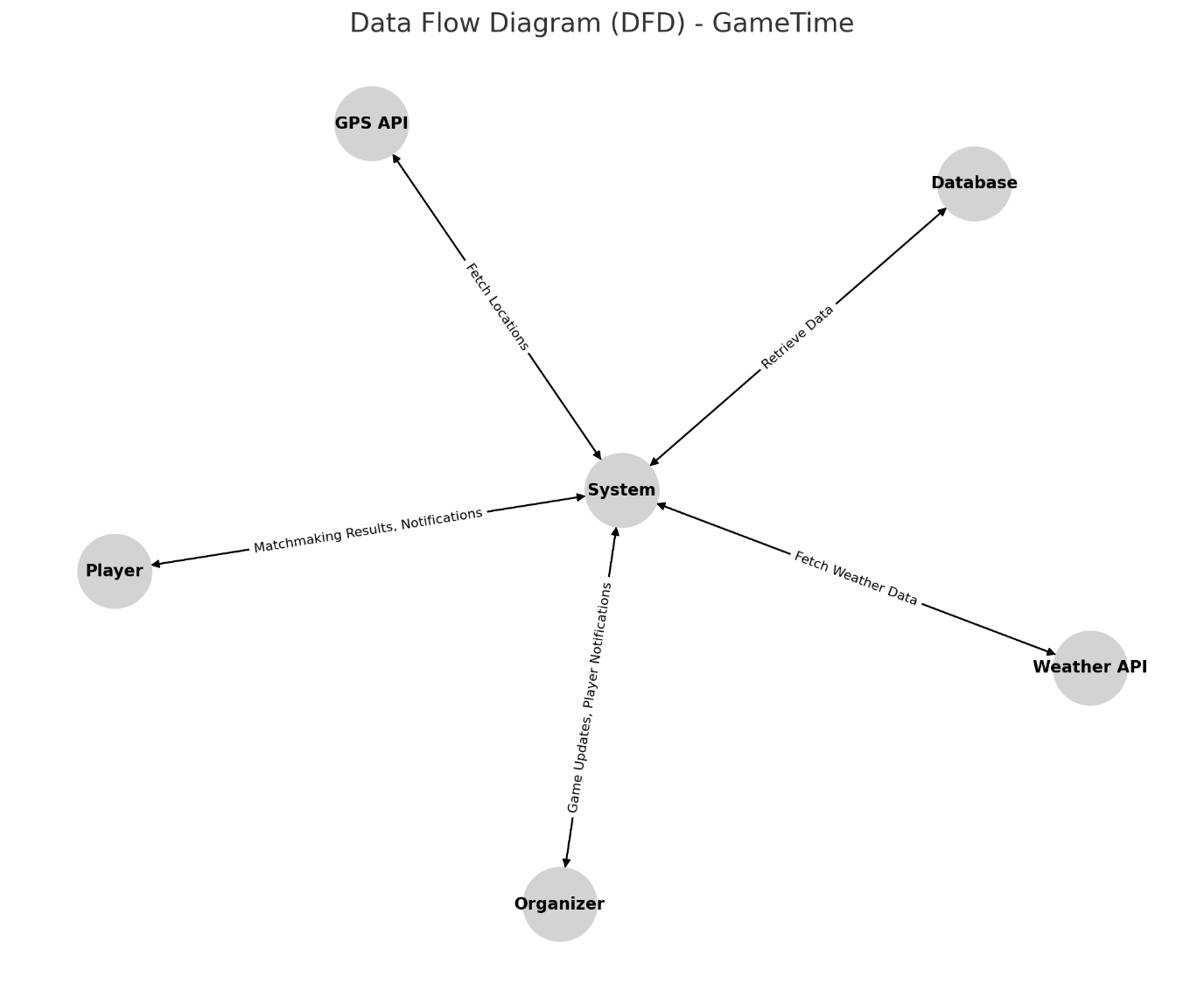
**Database Tables**

1. **User Table**:
   * Fields: user\_id, name, email, password\_hash, skill\_level, preferred\_position, location.
2. **Game Table**:
   * Fields: game\_id, organizer\_id, location, time, skill\_level, max\_players, current\_players.
3. **Location Table**:
   * Fields: location\_id, latitude, longitude, surface\_type, rain\_cover, shade\_cover.
4. **Player Feedback Table**:
   * Fields: feedback\_id, game\_id, player\_id, rating, comments.
5. **Weather Data Table**:
   * Fields: weather\_id, game\_id, temperature, condition.
6. **Team Allocation Table**:
   * Fields: allocation\_id, game\_id, team, player\_id.

**Part 3: *Graphic Description***

**Data Flow Diagram**

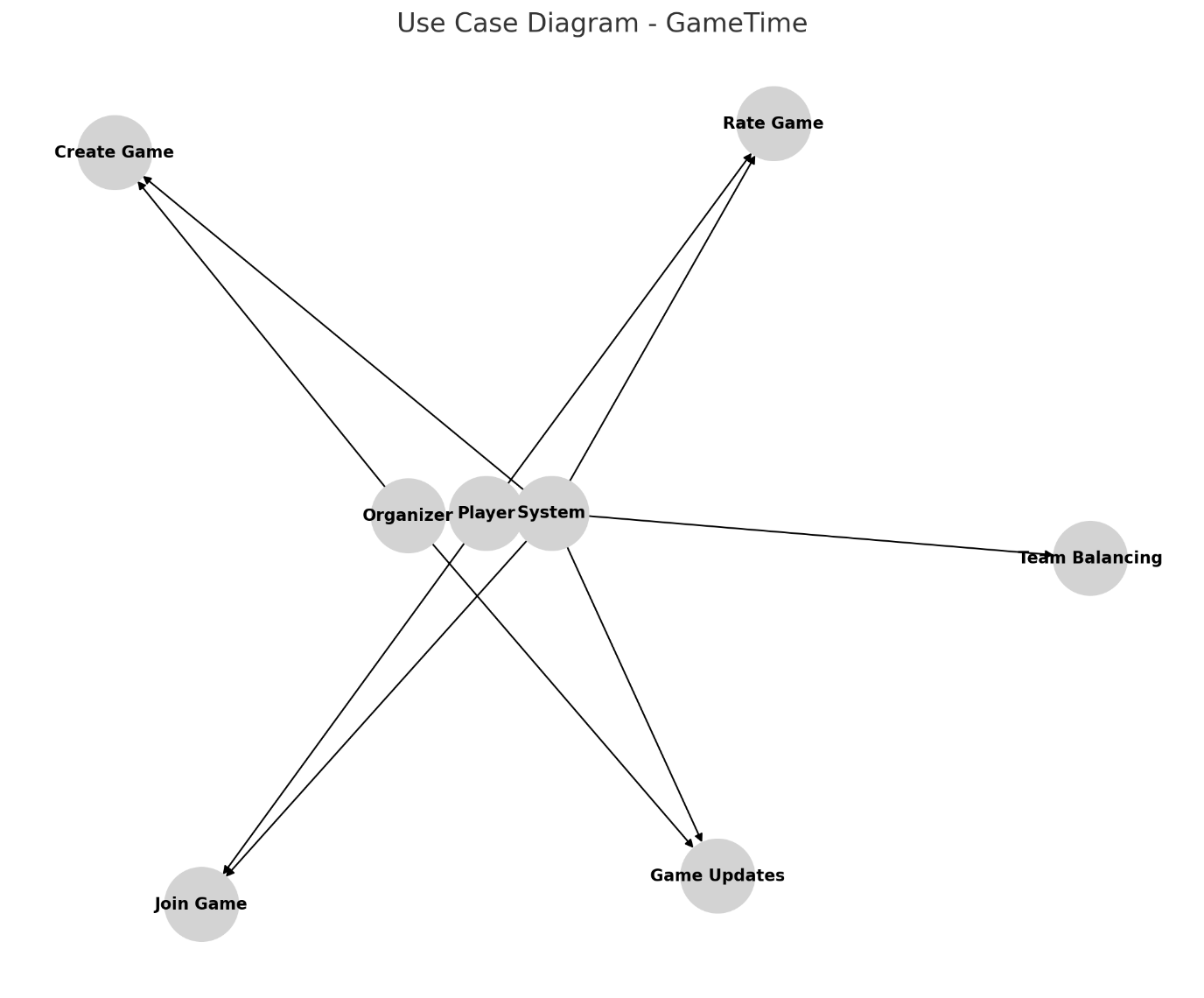
1. **User Interaction**: Players register and create profiles through the View layer.
2. **Game Creation**: Organizers input game details, stored in the database via the Model layer.
3. **Matchmaking**: The Controller processes player data to allocate teams.
4. **Real-Time Updates**: Weather and location data are fetched through APIs.



**Part 4: *API Specification***

**Key APIs**

1. **User Management API**:
   * POST /register: Register a new user.
   * POST /login: User login.
   * GET /profile: Retrieve user profile.
2. **Game Management API**:
   * POST /create\_game: Create a new game.
   * GET /games: Fetch available games based on preferences.
   * POST /join\_game: Join a game.
3. **Weather Integration API**:
   * GET /weather: Fetch weather conditions for a location.
4. **Location Services API**:
   * GET /nearby\_fields: Retrieve nearby football fields based on GPS coordinates.



**Part 5: *Interface Design***

**Key Screens**

1. **Login and Registration**:
   * Input fields for email/password or social login buttons.
2. **Home Screen**:
   * List of available games with search and filter options.
3. **Game Details**:
   * Display game information, join button, and chat option.
4. **Profile Screen**:
   * Editable user profile with preferences and statistics.
5. **Organizer Dashboard**:
   * Tools for game management and player notifications.

**Part 6: *Programming Languages and Tools***

* **Frontend**: Flutter
* **Backend**: Node.js with Express
* **Database**: PostgreSQL and Redis for caching
* **APIs**: Google Maps API, OpenWeatherMap API
* **Testing**: Postman for API testing
* **Version Control**: GitHub

**Part 7: *Algorithms Description***

1. **Team Balancing Algorithm**:
   * Matches players based on skill levels and positions.
   * Uses a weighted scoring system to create balanced teams.
2. **Matchmaking Algorithm**:
   * Filters games based on player preferences and location proximity.
3. **Weather Handling Algorithm**:
   * Analyzes weather data to suggest rescheduling or alternative locations.
4. **Real-Time Notifications**:
   * Implements WebSockets to notify players of game updates.

**Part 8: *Non-Functional Requirements***

1. **Performance**:
   * Response time: <500ms for API calls.
   * Concurrent users: Support for up to 100 users.
2. **Scalability**:
   * Modular architecture to support horizontal scaling.
3. **Security**:
   * Passwords stored as hashed values.
   * Secure communication with HTTPS.
4. **Usability**:
   * Intuitive UI for smooth user navigation.
   * Accessibility features for inclusivity.
5. **Reliability**:
   * 99% uptime with basic redundancy mechanisms.

**Part 9: *Testing Plan***

1. **Unit Tests**:
   * Test individual components like user registration and matchmaking.
2. **Integration Tests**:
   * Validate the interaction between APIs and the database.
3. **System Tests**:
   * End-to-end testing of core functionalities like game creation and team allocation.
4. **User Testing**:
   * Gather feedback from target users to refine features.

By following this detailed design, the GameTime project aims to deliver a robust and user-friendly platform for organizing and managing amateur football games.