Task 1: Project Setup and Structure

- **Objective:** Set up the Spring Boot project structure for Soccbuzz.
- **Steps:**
    1. Create a new Spring Boot project using Spring Initializr.
    2. Include dependencies: Spring Web, Spring Data JPA, and any additional dependencies needed for data storage (like H2 or MySQL).
    3. Configure the project in your preferred IDE.

Task 2: Define Domain Models

- **Objective:** Define entities to represent the core data structures of Soccbuzz.
- **Steps:**
    1. Create Java classes for entities such as `Match`, `Team`, `League`, `Schedule`, etc.
    2. Use JPA annotations (`@Entity`, `@Table`, `@ManyToOne`, `@OneToMany`, etc.) to define relationships between entities.
    3. Consider attributes like match date/time, team names, scores, and league details.

Task 3: Implement CRUD APIs

- **Objective:** Develop basic CRUD operations for managing soccer matches and related entities.
- **Steps:**
    1. Create a REST controller (`MatchController` or similar).
    2. Implement methods for:
        - Creating a new match (`POST /matches`)
        - Retrieving a list of all matches (`GET /matches`)
        - Retrieving details of a specific match (`GET /matches/{id}`)
        - Updating match details (`PUT /matches/{id}`)
        - Deleting a match (`DELETE /matches/{id}`)
    3. Use appropriate HTTP methods (`@PostMapping`, `@GetMapping`, etc.) and annotations (`@RequestBody`, `@PathVariable`, etc.).

Task 4: Implement Additional APIs

- **Objective:** Develop APIs for functionalities such as:
    - Filtering matches by league or date.
    - Retrieving schedules.
    - Handling match results and scores updates.

Task 5: Data Validation

- **Objective:** Implement validation for input data to ensure consistency and integrity.
- **Steps:**
    1. Use validation annotations (`@NotNull`, `@Size`, `@Min`, etc.) in entity classes.
    2. Validate input data in controller methods using `@Valid` annotation.
    3. Return appropriate HTTP status codes and error messages for validation failures.

ask 6: Error Handling

- **Objective:** Implement centralized error handling for the APIs.
- **Steps:**
    1. Create an exception handler (`@ControllerAdvice`).
    2. Handle specific exceptions (e.g., `EntityNotFoundException`, `MethodArgumentNotValidException`) and return appropriate error responses.
    3. Consider custom error messages and HTTP status codes (e.g., 404 for not found, 400 for bad request).