**PROG2002 – Web Development II**

**Assignment 2: Use Case (A Dynamic Website)**

**Student ID:** 24516947 **Last Name:** Sun **First Name:** Chongbo

**Title of the project:**

Charitable organizations and public welfare websites

**Introduction/Motivation**

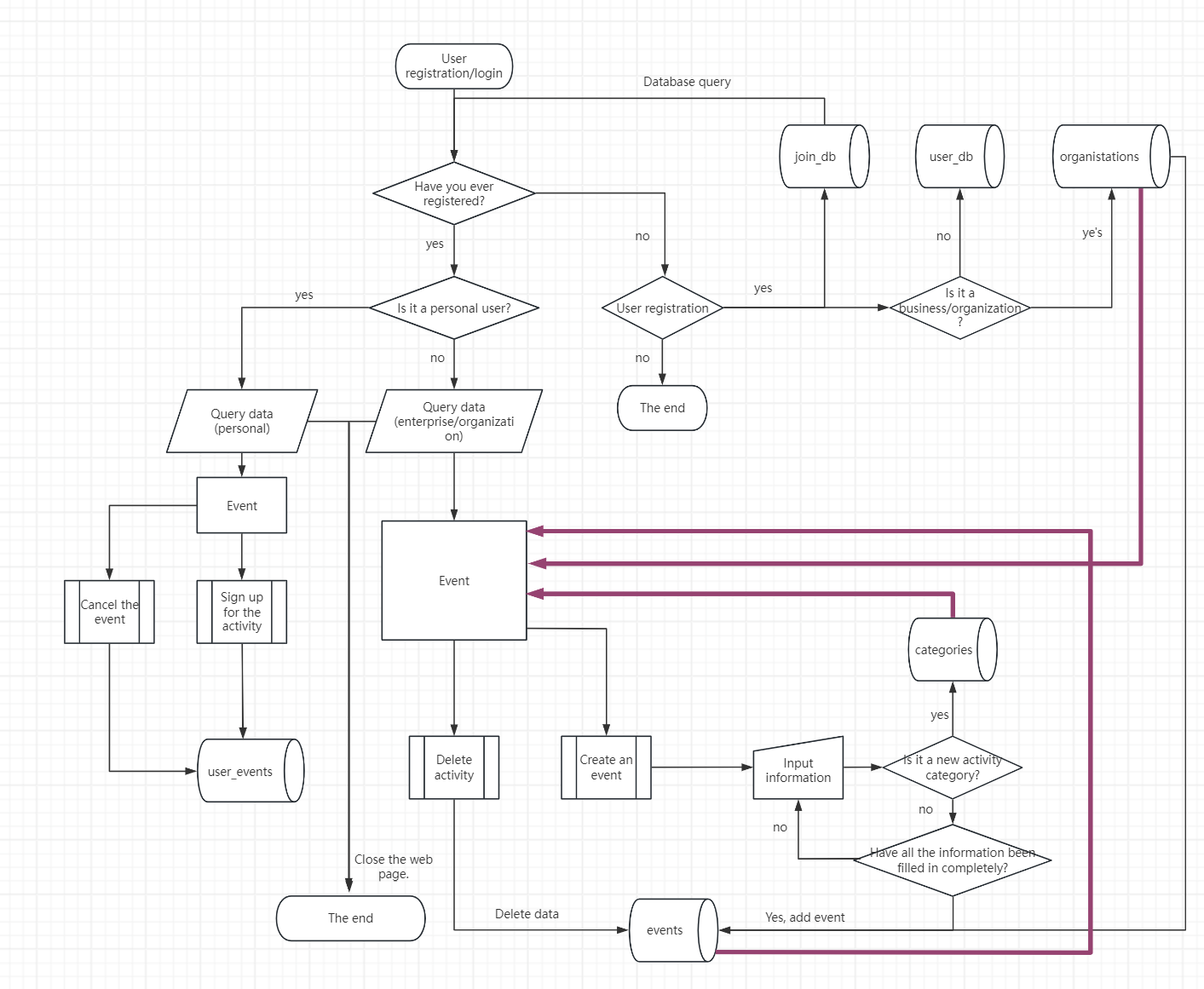
In today's society, charitable organizations are playing an increasingly vital role in addressing social issues and supporting vulnerable groups. However, many face challenges such as limited resources, lack of transparency, and communication barriers with the public. Meanwhile, potential donors and volunteers often struggle to find reliable sources of information about these organizations' activities and impact. This information asymmetry not only limits the social influence of philanthropy but also discourages broader participation in public welfare initiatives.

**Problem Statement**

Existing charity information platforms have exposed a series of significant operational issues that severely impact their efficiency and user experience. The fragmented information presentation and chronologically delayed updates create substantial barriers for the public seeking up-to-date, accurate event details, making it difficult to obtain timely support. The absence of a unified activity management system and efficient registration platform results in inefficiency for organizers, hindering resource coordination and event planning effectiveness. Donors and volunteers struggle to track their contributions or monitor participation progress through the platform, with this lack of transparency significantly dampening engagement. Communication channels between charities and supporters remain inadequate, with delayed and incomplete information sharing failing to build trust. The platform's weak data infrastructure lacks reliable metrics to demonstrate social impact, leading to public skepticism about project outcomes and undermining the sustainable development of charitable initiatives.

**Solution**

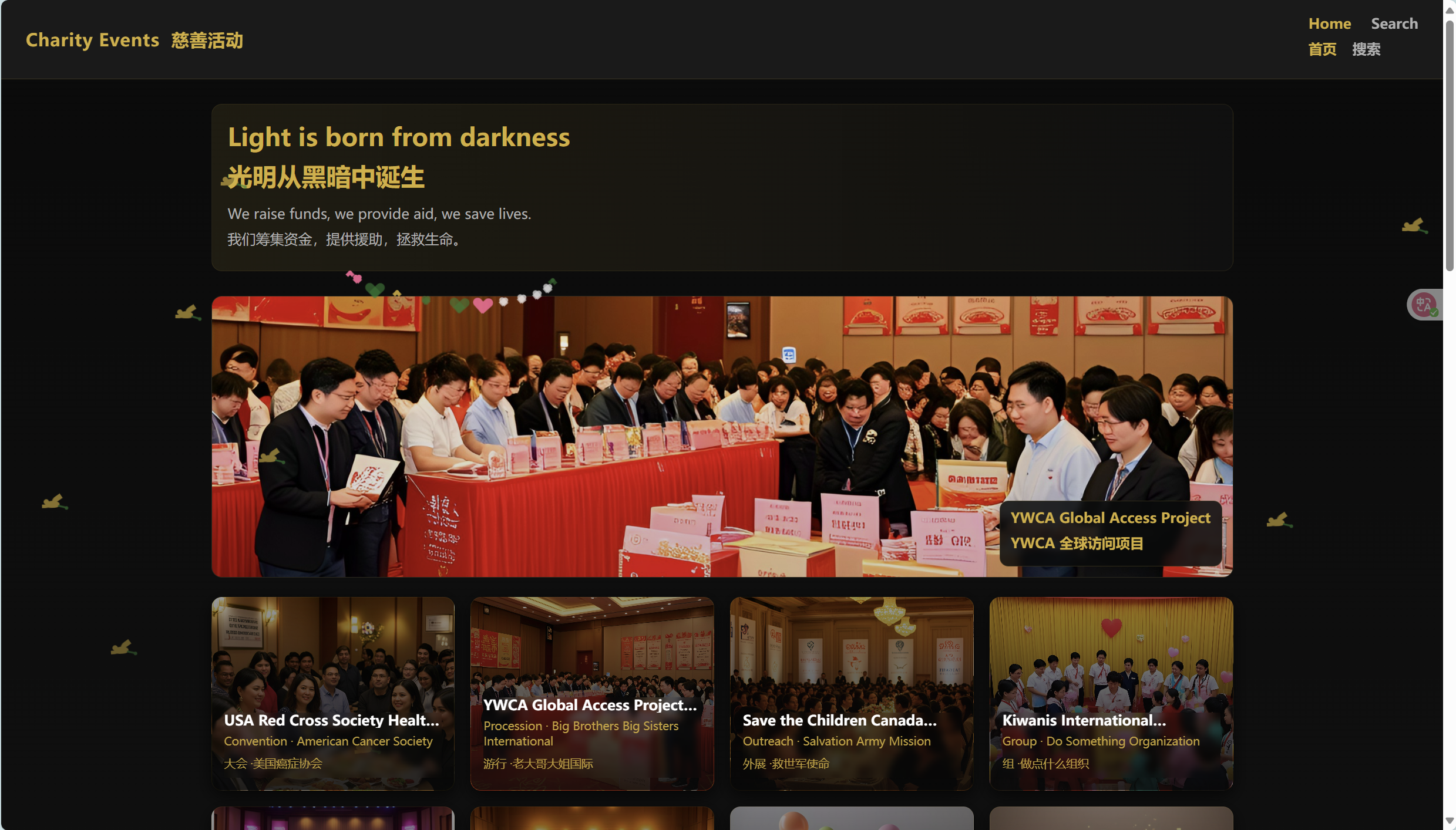
Front-end Architecture: Utilizes responsive web design to ensure optimal experience across desktop and mobile devices. Built with modern JavaScript frameworks for dynamic user interfaces, featuring intuitive browsing, search, and filtering capabilities. Back-end Architecture: Establishes API servers to provide standardized data interfaces. Designs relational databases for storing organizational records, events, and participation data. Implements secure user authentication and authorization systems. Technical Advantages: Front-end and back-end separation architecture enhances system maintainability and scalability. Standardized API interfaces support future feature expansion. Real-time data synchronization ensures information accuracy. Secure data transmission protects user privacy.



In my professional assessment, the workflow diagram I developed comprehensively addresses most problem-solving phases with logical clarity and detailed implementation steps. However, according to current Phase A2 requirements, we should concentrate on completing the purple-lined sections marked in bold. These components primarily involve the "search" function within the four core database operations——: insert, delete, update, and query. A robust system architecture should begin with meticulous user role identification during login authentication, including accurate classification of users as individual or organizational accounts to enable personalized services. Regarding database design, I recommend establishing a dedicated login database storing only unique user identifiers (if applicable in system design). This approach not only significantly improves webpage loading speeds but also reduces data processing volume per query. During WeChat Mini Program development, I observed that their database access mechanism retrieves all field data during each query – similar to MySQL's row-by-row data retrieval pattern. More importantly, some cloud database providers charge based on the number of columns accessed in a single query. Therefore, by optimizing the database structure and reducing unnecessary field calls, not only can the system performance be improved, but also the operation cost can be reduced to a certain extent. This is undoubtedly an optimization scheme worth considering for projects that pursue high efficiency and cost control.

**Web UX**

In the user experience (UX) design process, I primarily adopted a color scheme featuring black backgrounds paired with white and gold for title text. This high-contrast color combination ensures users can clearly identify required information during browsing, thereby enhancing readability and legibility. Additionally, the top navigation bar maintains the same gold and gray color scheme, which not only maintains visual consistency but also provides clear navigation guidance, enabling users to quickly locate desired functional modules. In event display areas, I introduced innovative design elements using rounded corners and semi-transparent backgrounds. These designs effectively present images and text while delivering a soft, non-intrusive visual experience that creates a comfortable and pleasant atmosphere throughout the page. When hovering over these areas, dynamic pop-up effects are triggered – a dynamic interaction design that adds visual interest while making critical information more prominent. The detail pages also include image zoom functionality, allowing users to click to enlarge visuals for closer examination. To ensure optimal visual experience across different screen resolutions, I specifically considered layout optimization for widescreen displays. To prevent excessive border gaps on overly wide displays, I utilized CSS functions to create small dove-of-freedom patterns that gently descend from the page's top. This design not only fills the empty space but also adds vitality and richness to the overall visual. Additionally, I incorporated a mouse trail effect with an adorable heart shape to complement the descending doves, collectively highlighting the theme—— of charity. These meticulous details enhance user interaction while reinforcing the page's thematic expression. Through such thoughtful design considerations, I strive to deliver both aesthetically pleasing and functional web experiences for users.



**Data Schema**

Entity Details:

1. Organization Table (Organisations)

- Primary Key: id (auto-incrementing integer)

- Core Attributes: Organization name, organization description, contact information, organization address, organization logo

- Timestamp: created\_at (creation time)

- Function: Stores basic information of charitable organizations.

2. Category Table (Categories)

- Primary Key: id (auto-incrementing integer)

- Core Attributes: Category name, category description

- Function: Categorizes and manages activities.

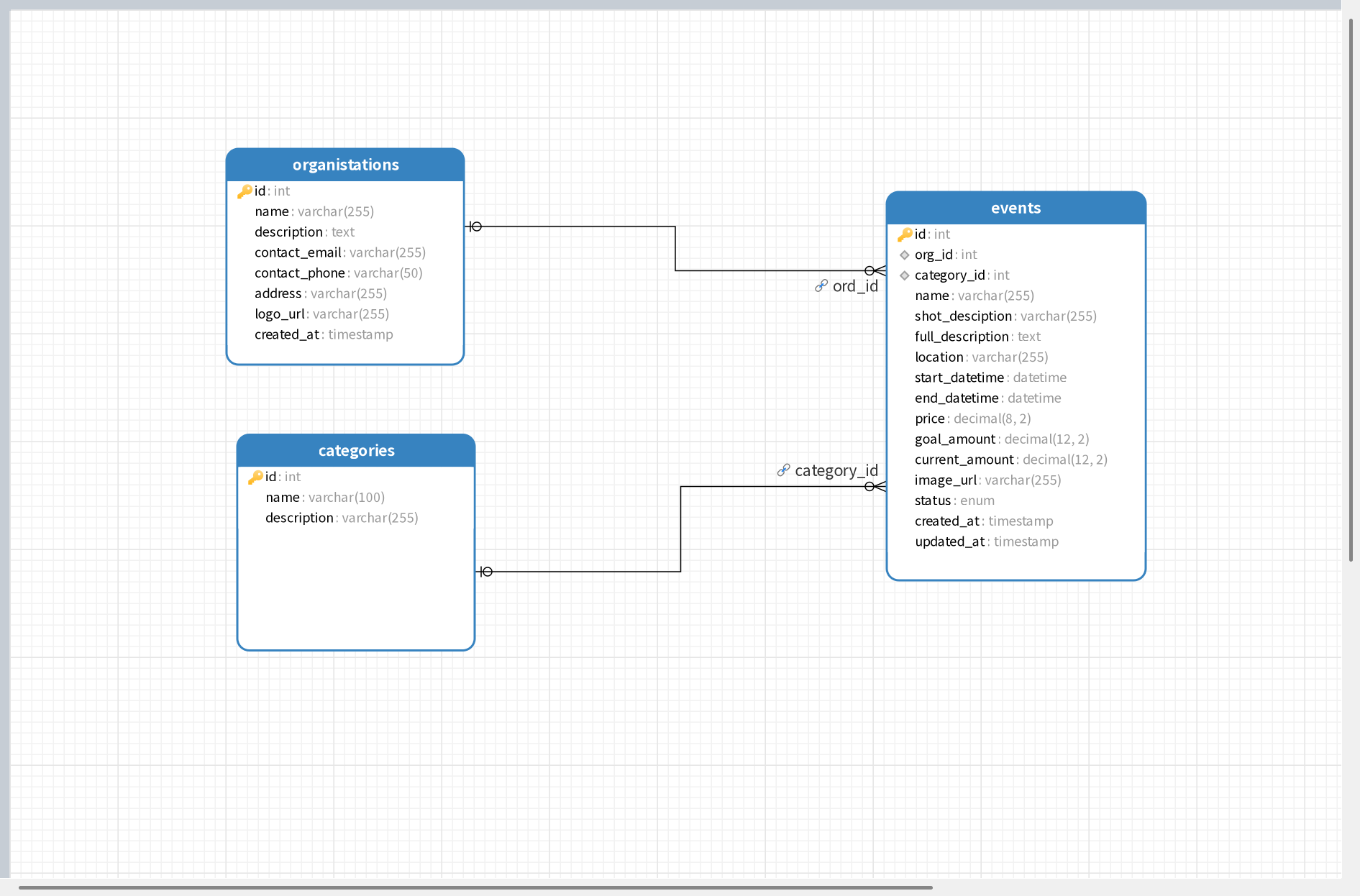
3. Event Table (Events)

- Primary Key: id (auto-incrementing integer)

- Foreign Key Relationships: org\_id → related\_organisations.id, category\_id → related\_categories.id

- Core Attributes: event name, event description, event time, event location, funding information, event status

- Timestamps: created\_at (creation time), updated\_at (updated time).



By defining the foreign keys org\_id and category\_id in the events table, we establish an inseparable bond between three entities. 1. Organization-Activity Relationship: organisations.id → events.org\_id This represents a classic one-to-many relationship. A charitable organization can create multiple public welfare activities, while each activity is uniquely associated with a specific organization. Business logic implementation: When viewing an "Hope Foundation" organization page, all its initiated activities (e.g., "Hope Foundation Annual Gala", "Hope Foundation Book Donation") are displayed. When browsing specific activities, users can clearly identify the organizing institution and access its detailed profile. Database-level constraints: When inserting new activity records into the events table, the provided org\_id must already exist in the organizations table. Failure to meet this condition triggers database rejection with a foreign key constraint error, preventing phantom activities (those not affiliated with any organization). Due to the ON DELETE RESTRICT clause, records cannot be deleted if existing activities remain under an organization, effectively eliminating orphaned data. 2. Classification-Activity Mapping: categories.id → events.category\_id This represents a many-to-one relationship where a single category (e.g., "Educational Support") may encompass multiple public welfare activities, while each activity typically belongs to only one primary category. System Implementation: Users can filter activities by category on the platform, for example, selecting "Environmental Protection" to display all tagged events (e.g., "Community Cleanup Day", "Afforestation Campaign"). This enables structured management and efficient search of extensive activity data. Database Constraints: Similar to organizational relationships, this ensures every category\_id in the events table corresponds to a valid existing category. This maintains data consistency by preventing activities from being assigned to non-existent categories.

**API design**

* List the main API endpoints of your application (e.g., GET /api/users, POST /api/posts).
* For a single API endpoint, describe its purpose, the data it expects to receive (request body), and the data it will return (response body).
* Explain your choice of HTTP methods (GET, POST, PUT, DELETE, etc.) for each endpoint.

GET /api/events - Use filtering to capture all events

Search the list of charitable events and filter by date, location, category name, and organization name. This is the main endpoint for viewing events.

All parameters are optional and can be combined as desired:

| **parameter name** | **Type** | **Example** | **Explain** |
| --- | --- | --- | --- |
| date | string | "2024-01-15" | Filtering for activities on a specific date (exact match date) |
| location | string | "New York" | Filter by location (fuzzy match, include keywords) |
| category | string | "Education" | Filter by category name (fuzzy match) |

Request example:

GET /api/events?date=2024-01-15&location=New%20York&category=Education

Return example:

[

{

"id": 1,

"org\_id": 5,

"category\_id": 3,

"name": "Annual Charity Gala",

"shot\_desciption": "An evening of fundraising for children's education",

"full\_description": "Join us for an elegant evening...",

"location": "New York Grand Hotel",

"start\_datetime": "2024-01-15T19:00:00.000Z",

"end\_datetime": "2024-01-15T23:00:00.000Z",

"price": 150.00,

"goal\_amount": 50000.00,

"current\_amount": 25000.00,

"image\_url": "/images/gala-2024.jpg",

"status": "active",

"created\_at": "2023-12-01T10:00:00.000Z",

"updated\_at": "2023-12-15T14:30:00.000Z",

// Associated data (obtained through JOIN query)

"category\_name": "Education Support",

"org\_name": "Hope Foundation International"

},

{

"id": 2,

"org\_id": 8,

"category\_id": 2,

"name": "Community Park Cleanup",

// ... Other fields

"category\_name": "Environmental",

"org\_name": "Green Earth Alliance"

}

]

Reasons for Choosing the HTTP GET Method:

1. REST Compliance: The GET method is designed for data retrieval without modifying server states.

2. "Secure Method" Principle in RESTful Architecture: GET adheres to the principle of being a secure method within RESTful architecture.

3. Idempotence: It ensures that identical requests receive consistent responses.

4. Search/Browse Optimization: GET is ideal for browsing and searching scenarios.

5. Cache Efficiency: GET requests can be cached by browsers and CDN servers, enhancing efficiency.

6. Performance Optimization: GET reduces server workload by leveraging caching.

7. Parameter Standardization: GET uses URL parameters for filtering conditions, standardizing the process.

8. Web Standards Compliance: GET facilitates integration with front-end web standards.

9. Bookmark & Share Features: Complete URLs with all search criteria allow for easy bookmarking and sharing of links.