PROG2002 – Web Development II

Assignment 2: Use Case (A Dynamic Website)

Student ID: Last Name: First Name:

**Title of the project:** Charity Events Management Platform

**Introduction/Motivation**

The increasing demand for transparent and accessible charitable giving in modern society has created a significant need for digital platforms that effectively connect charitable organizations with potential donors and participants. Traditional methods of promoting charitable events, such as printed flyers and word-of-mouth communication, have proven to be inefficient in reaching a broader audience and fail to provide real-time information updates regarding event details, fundraising progress, and participant registration. This project addresses these limitations by developing a comprehensive web-based platform that serves as a centralized hub for charitable event management and public engagement.

The primary motivation behind this project stems from the recognition that technology can play a transformative role in facilitating philanthropic activities and enhancing community participation in charitable causes. By leveraging modern web development technologies, including RESTful API architecture and dynamic client-side interfaces, this platform aims to streamline the process of event discovery, information dissemination, and participant engagement. The system is designed to benefit multiple stakeholders, including charitable organizations that require efficient tools for event promotion and fundraising management, as well as potential participants who seek convenient access to comprehensive information about upcoming charitable activities in their local communities.

Furthermore, the platform addresses the contemporary expectation for immediate access to information and the ability to make informed decisions based on comprehensive event details, including fundraising goals, current participation levels, and specific causes being supported. This project represents a practical application of client-server web development principles, demonstrating how modern web technologies can be employed to create meaningful social impact while simultaneously showcasing technical proficiency in full-stack web development, database design, and API implementation.

Problem Statement

The current landscape of charitable event management and promotion faces several critical challenges that hinder effective public engagement and organizational efficiency. Charitable organizations often struggle to reach their target audience due to fragmented information channels, limited visibility of their events, and the absence of centralized platforms that aggregate charitable activities across different organizations and causes. This fragmentation results in reduced public awareness, lower participation rates, and ultimately, diminished fundraising outcomes for worthy causes.

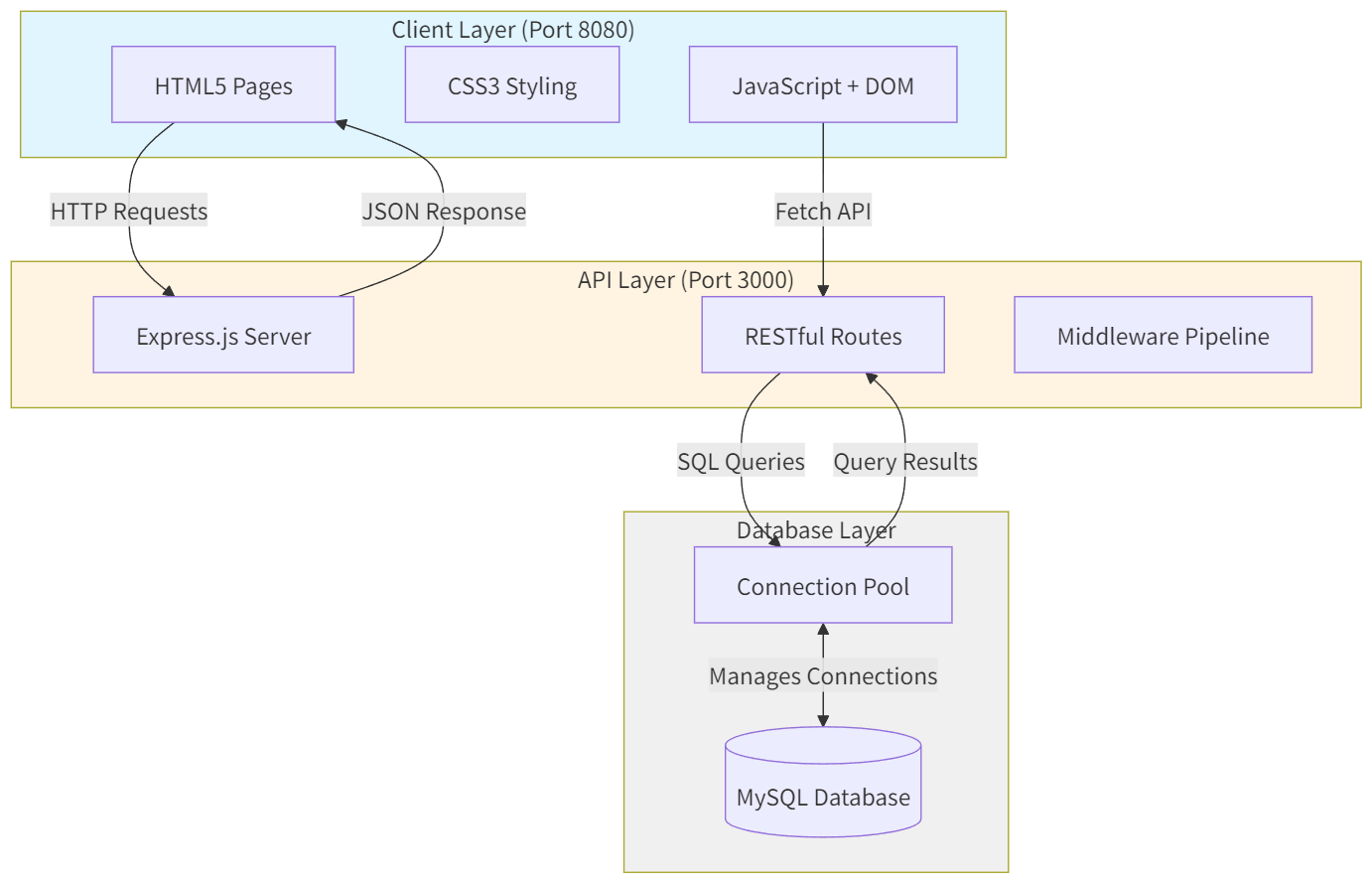
From the perspective of potential participants and donors, the lack of a unified platform presents significant obstacles in discovering charitable events that align with their interests, geographical proximity, and scheduling constraints. Individuals interested in participating in charitable activities must often navigate multiple websites, social media pages, and physical advertisements to locate relevant events, which proves to be time-consuming and frequently leads to missed opportunities for engagement. Additionally, the absence of comprehensive event information, including real-time fundraising progress, participant capacity, and detailed cause descriptions, prevents individuals from making well-informed decisions about their charitable involvement.

The technical challenges inherent in developing such a platform include the design and implementation of a robust database schema capable of managing complex relationships between events, organizations, and categories while maintaining data integrity and supporting efficient query operations. Furthermore, the system must provide a seamless integration between the server-side data management layer and the client-side presentation layer through well-designed RESTful APIs that facilitate reliable data retrieval and manipulation. The platform must also address user experience considerations, ensuring that the interface remains intuitive and responsive while handling dynamic content updates and supporting various search and filtering operations that enable users to efficiently locate events matching their specific criteria.

**Solution**

The solution implemented in this project comprises a three-tier web application architecture that effectively separates concerns between data persistence, business logic, and presentation layers. This architectural approach ensures maintainability, scalability, and adherence to modern web development best practices while facilitating efficient communication between the client and server components of the system.

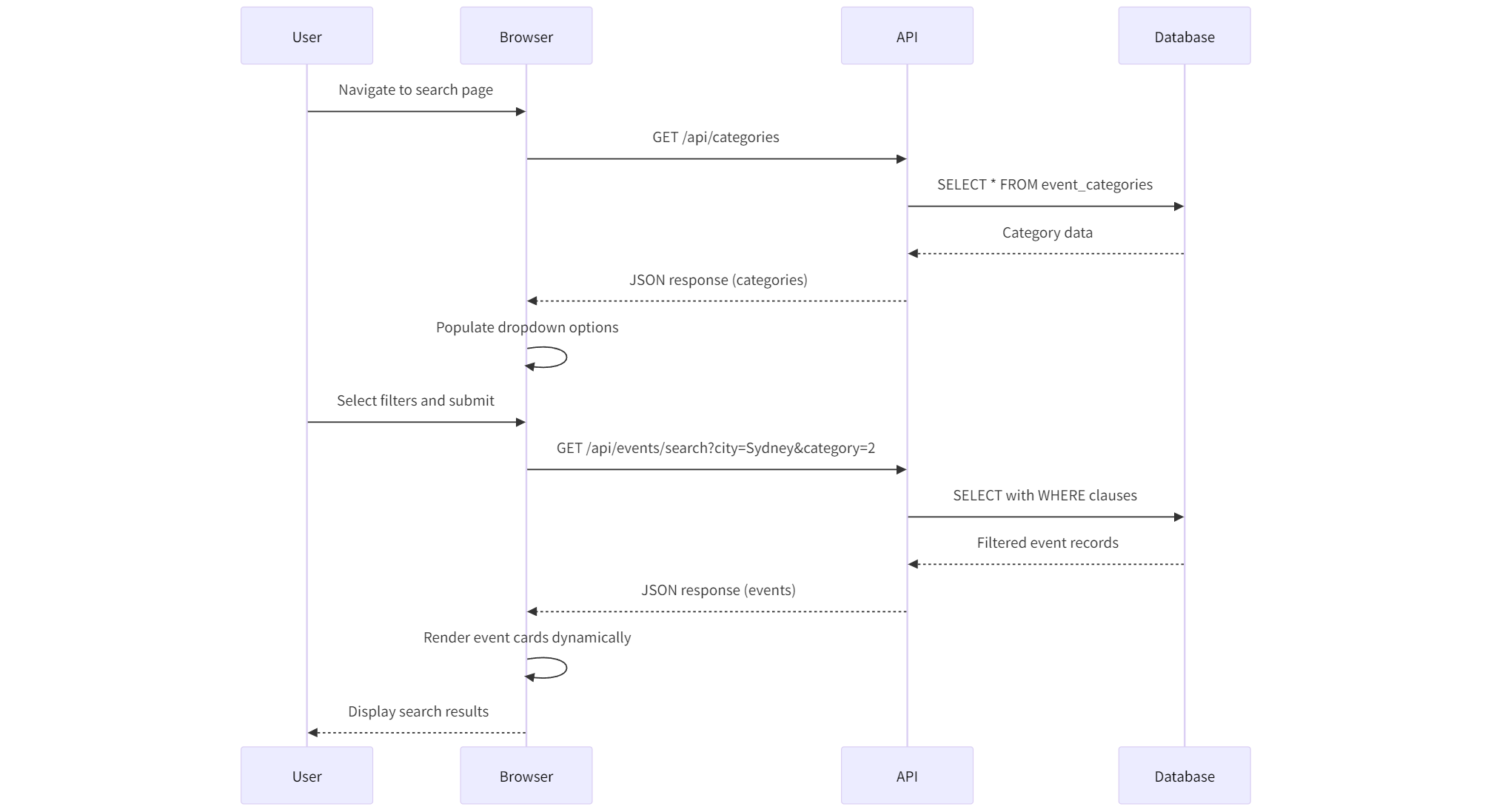
System Architecture and Technology Stack



The application employs MySQL as the relational database management system, chosen for its robust support for complex queries, transaction management, and referential integrity constraints essential for maintaining consistent relationships between charitable organizations, event categories, and individual events. The database layer is accessed through a Node.js-based server application utilizing the Express framework, which provides a lightweight yet powerful foundation for implementing RESTful API endpoints. The Express middleware architecture facilitates request processing, error handling, and database connection management through a connection pool mechanism that optimizes resource utilization and ensures reliable database connectivity.

The client-side implementation leverages pure JavaScript, HTML5, and CSS3 technologies without relying on frontend frameworks such as Angular, in accordance with the assignment requirements. This approach demonstrates proficiency in fundamental web technologies and DOM manipulation techniques while utilizing modern JavaScript features including async/await syntax for handling asynchronous API requests and Promises for managing concurrent operations. The client application communicates with the server exclusively through the RESTful API layer, ensuring a clear separation between the presentation and data access concerns.

Client-Server Communication Concept



The communication between the client and server follows the REST (Representational State Transfer) architectural style, which provides a stateless, cacheable, and uniform interface for data exchange. When a user navigates to any page within the application, the client initiates HTTP GET requests to specific API endpoints corresponding to the required data. For instance, when loading the home page, the client sends a GET request to the /api/events endpoint, which triggers the server to query the database for all current and upcoming events, join the relevant tables to include category and organization information, and return a JSON response containing the complete event dataset.

The server processes these requests through defined route handlers that execute parameterized SQL queries against the MySQL database, utilizing the mysql2 library's Promise-based API to handle asynchronous database operations. The query results are transformed into JSON format and wrapped in a standardized response structure that includes success indicators, data payloads, and metadata such as result counts. This structured response format enables the client to consistently process API responses regardless of the specific endpoint being accessed.

Error handling is implemented at multiple layers of the application stack, with the server catching database errors, connection failures, and invalid requests, returning appropriate HTTP status codes and descriptive error messages. The client intercepts these error responses and presents user-friendly error messages through DOM manipulation, ensuring that technical errors do not compromise the user experience. This comprehensive error handling strategy demonstrates an understanding of production-ready application development practices.

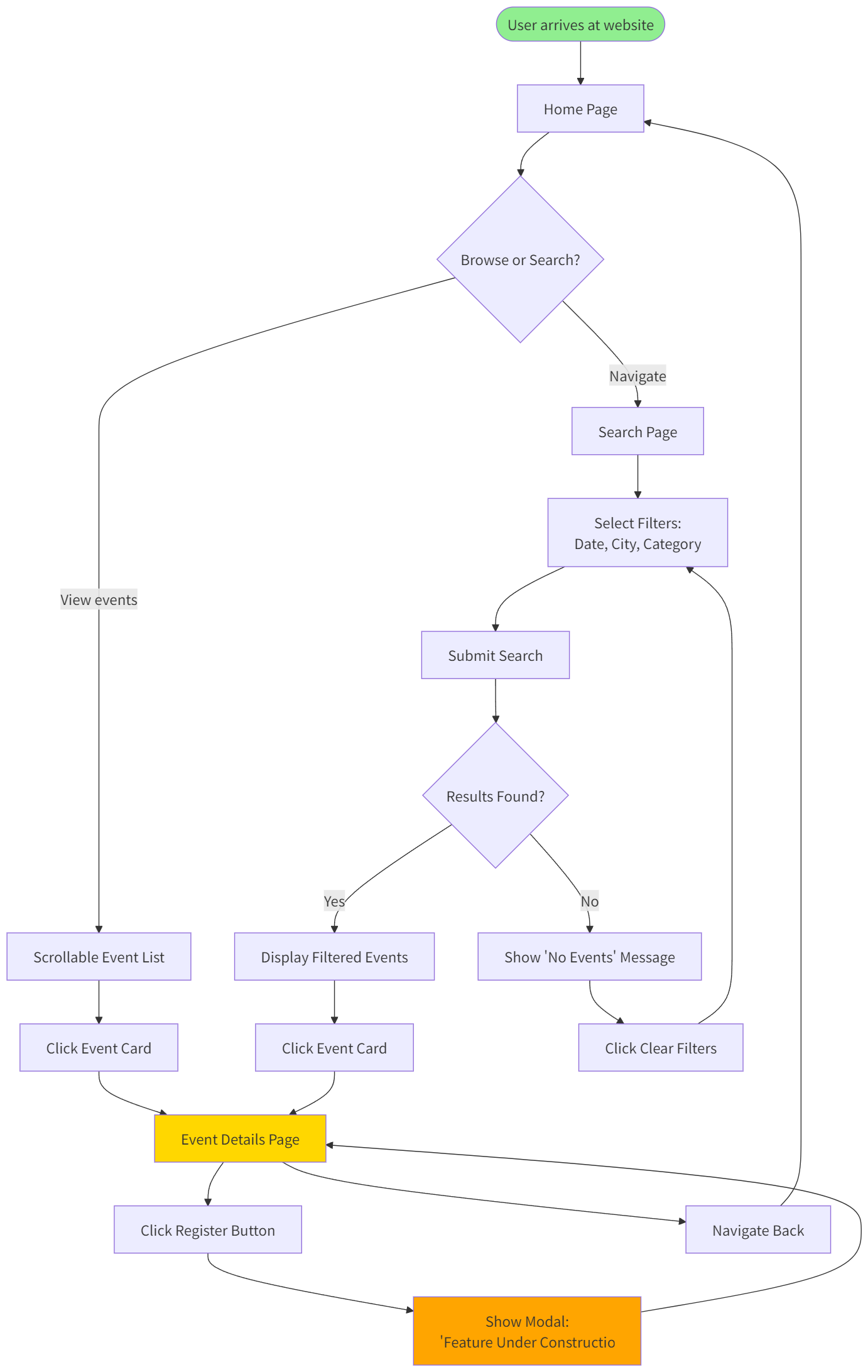
Data Flow and State Management

The application implements a unidirectional data flow pattern where data originates from the database, flows through the API layer, and is consumed by the client for presentation purposes. When users interact with search filters or navigation elements, the client constructs API requests with appropriate query parameters, sends them to the server, awaits the response, and dynamically updates the DOM to reflect the retrieved data. This approach eliminates the need for page reloads and provides a responsive, modern user experience characteristic of single-page application design patterns.

State management within the client is handled through URL query parameters for shareable state (such as event IDs in the details page) and in-memory JavaScript variables for transient state (such as search filter selections). This hybrid approach ensures that users can bookmark specific events or share event URLs while maintaining the simplicity of the implementation without introducing complex state management libraries.

**Web UX**

The user experience design of this application prioritizes intuitive navigation, clear information hierarchy, and responsive interaction patterns that accommodate users with varying levels of technical proficiency. The design process incorporated several key principles and techniques to ensure accessibility and usability across different user demographics and device types.

Navigation and Information Architecture

The application implements a consistent navigation structure present across all pages, featuring a header menu that provides immediate access to the home page, search functionality, and organizational information. This navigation pattern follows established web conventions, reducing cognitive load and enabling users to quickly orient themselves within the application regardless of their entry point. The navigation elements utilize semantic HTML5 tags and maintain consistent visual styling through a cohesive CSS design system that establishes clear visual hierarchies through typography, color, and spacing.

The information architecture was designed to support two primary user journeys: browsing available events through the chronologically organized home page listing, and searching for specific events using the dedicated search interface with multiple filtering criteria. This dual-path approach accommodates different user preferences and search strategies, recognizing that some users prefer to explore available options while others have specific requirements and wish to narrow results immediately.

### Visual Design and Layout Considerations

The layout employs a card-based design pattern for event listings, which provides a familiar and scannable format for presenting multiple items with associated metadata. Each event card displays essential information including the event name, date, location, category, and associated organization, along with visual elements such as event imagery and status indicators. The card design utilizes CSS Grid and Flexbox layout techniques to ensure responsive adaptation to different viewport sizes, maintaining readability and visual appeal across desktop, tablet, and mobile devices.

Typography choices prioritize legibility, with carefully selected font sizes, line heights, and contrast ratios that meet WCAG accessibility guidelines. The color scheme employs a professional palette that conveys trustworthiness and aligns with the charitable nature of the platform, while status indicators and call-to-action buttons utilize distinct colors to draw attention to important interactive elements.

### Interactive Elements and Feedback Mechanisms

The application provides immediate visual feedback for all user interactions, including hover states on clickable elements, loading indicators during API requests, and clear error messages when operations fail or return empty results. The search functionality demonstrates particular attention to user experience through the implementation of a "Clear Filters" button that resets all form inputs and returns the interface to its initial state, showcasing basic DOM manipulation capabilities while addressing a practical user need.

Form controls within the search interface are carefully selected to match the data type and expected user interaction patterns: date pickers for temporal filtering, dropdown selects for categorical choices with defined options, and text inputs for open-ended location searches. This appropriate mapping of control types to data characteristics reduces user error and streamlines the filtering process.

### Wireframing and Design Iteration

Although formal wireframing tools were not extensively employed in this project, the design process involved sketching layout concepts and evaluating information hierarchy to ensure that the most critical information receives prominent placement. The event details page, for instance, organizes information into logical sections including event overview, detailed description, ticketing information, fundraising progress visualization, and organizational details, allowing users to quickly locate relevant information based on their specific interests.

The decision to display fundraising progress through a visual progress bar rather than numerical values alone demonstrates consideration for cognitive processing efficiency, as visual representations of quantitative data enable faster comprehension compared to textual descriptions. Similarly, the use of icons alongside text labels in event cards leverages dual coding theory to enhance information retention and processing speed.

The application implements a consistent navigation structure present across all pages, featuring a header menu that provides immediate access to the home page, search functionality, and organizational information. This navigation pattern follows established web conventions, reducing cognitive load and enabling users to quickly orient themselves within the application regardless of their entry point. The navigation elements utilize semantic HTML5 tags and maintain consistent visual styling through a cohesive CSS design system that establishes clear visual hierarchies through typography, color, and spacing.

The information architecture was designed to support two primary user journeys: browsing available events through the chronologically organized home page listing, and searching for specific events using the dedicated search interface with multiple filtering criteria. This dual-path approach accommodates different user preferences and search strategies, recognizing that some users prefer to explore available options while others have specific requirements and wish to narrow results immediately.

Visual Design and Layout Considerations

The layout employs a card-based design pattern for event listings, which provides a familiar and scannable format for presenting multiple items with associated metadata. Each event card displays essential information including the event name, date, location, category, and associated organization, along with visual elements such as event imagery and status indicators. The card design utilizes CSS Grid and Flexbox layout techniques to ensure responsive adaptation to different viewport sizes, maintaining readability and visual appeal across desktop, tablet, and mobile devices.

Typography choices prioritize legibility, with carefully selected font sizes, line heights, and contrast ratios that meet WCAG accessibility guidelines. The color scheme employs a professional palette that conveys trustworthiness and aligns with the charitable nature of the platform, while status indicators and call-to-action buttons utilize distinct colors to draw attention to important interactive elements.

Interactive Elements and Feedback Mechanisms

The application provides immediate visual feedback for all user interactions, including hover states on clickable elements, loading indicators during API requests, and clear error messages when operations fail or return empty results. The search functionality demonstrates particular attention to user experience through the implementation of a "Clear Filters" button that resets all form inputs and returns the interface to its initial state, showcasing basic DOM manipulation capabilities while addressing a practical user need.

Form controls within the search interface are carefully selected to match the data type and expected user interaction patterns: date pickers for temporal filtering, dropdown selects for categorical choices with defined options, and text inputs for open-ended location searches. This appropriate mapping of control types to data characteristics reduces user error and streamlines the filtering process.

Wireframing and Design Iteration

Although formal wireframing tools were not extensively employed in this project, the design process involved sketching layout concepts and evaluating information hierarchy to ensure that the most critical information receives prominent placement. The event details page, for instance, organizes information into logical sections including event overview, detailed description, ticketing information, fundraising progress visualization, and organizational details, allowing users to quickly locate relevant information based on their specific interests.

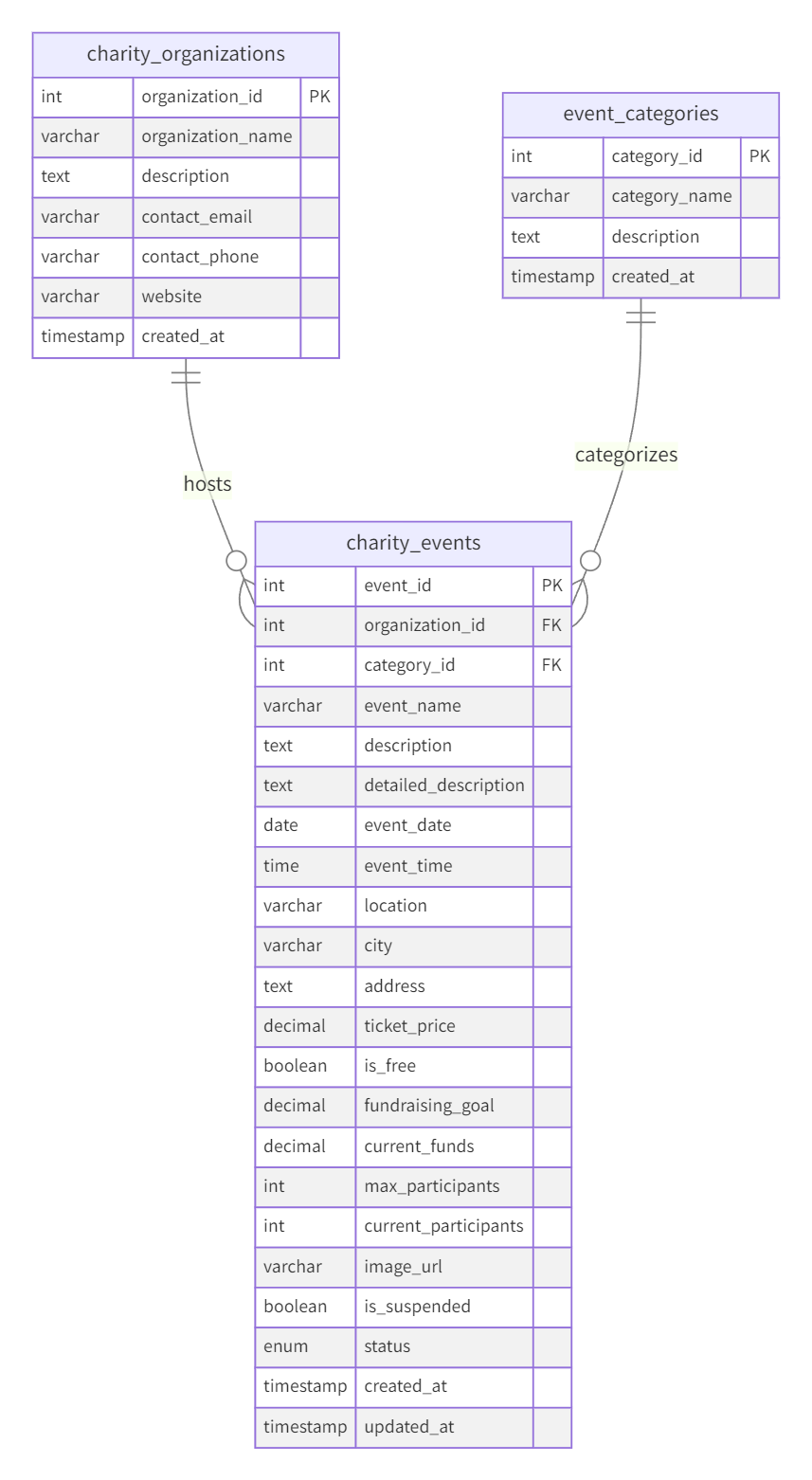
The decision to display fundraising progress through a visual progress bar rather than numerical values alone demonstrates consideration for cognitive processing efficiency, as visual representations of quantitative data enable faster comprehension compared to textual descriptions. Similarly, the use of icons alongside text labels in event cards leverages dual coding theory to enhance information retention and processing speed.

Data Schema

The database schema for this application employs a normalized relational structure designed to eliminate data redundancy while maintaining referential integrity and supporting efficient query operations across multiple related entities. The schema consists of three primary tables that collectively manage the complete information ecosystem for charitable events, organizations, and categorizations.

### Entity-Relationship Overview

The data model is structured around the central charity\_events table, which maintains foreign key relationships with both the charity\_organizations and event\_categories tables, establishing a many-to-one relationship pattern where multiple events can be associated with a single organization and multiple events can belong to a single category, but each event is associated with exactly one organization and one category.



### Table Specifications and Attributes

charity\_organizations Table:  
This table maintains information about the charitable organizations that host events on the platform. The primary key organization\_id uniquely identifies each organization, while descriptive attributes including organization\_name, description, contact\_email, contact\_phone, and website provide comprehensive organizational information necessary for public transparency and contact purposes. The created\_at timestamp records when the organization was registered in the system, supporting audit trail requirements.

event\_categories Table:  
This table implements a classification system for charitable events, enabling filtering and search operations based on event type. The primary key category\_id serves as the unique identifier, while category\_name and description provide human-readable category information. Examples of categories include "Charity Dinner", "Fun Run", "Silent Auction", "Charity Concert", and "Community Fair", each representing distinct event formats with different participant expectations and engagement models.

charity\_events Table:  
This central table contains the comprehensive event information that forms the core of the application's data model. The primary key event\_id uniquely identifies each event, while foreign keys organization\_id and category\_id establish relationships with the organizations and categories tables respectively. Temporal attributes including event\_date, event\_time, and status (enumerated as 'upcoming', 'ongoing', or 'completed') enable time-based filtering and status determination.

Location attributes including location, city, and address support geographical search operations and provide participants with venue information. Financial attributes such as ticket\_price, is\_free, fundraising\_goal, and current\_funds facilitate transparency regarding event costs and fundraising progress. Capacity management attributes including max\_participants and current\_participants enable the system to track registration limits and display availability information.

The is\_suspended boolean flag provides administrators with the capability to temporarily remove events from public visibility in cases of policy violations or organizational requests, while the image\_url attribute supports visual content association with events. The created\_at and updated\_at timestamps enable audit tracking and support chronological sorting operations.

### Indexing Strategy

To optimize query performance for common access patterns, several indexes are implemented on the charity\_events table:

|  |  |  |
| --- | --- | --- |
| Index Name | Column(s) | Purpose |
| idx\_event\_date | event\_date | Accelerates chronological sorting and date-based filtering |
| idx\_event\_city | city | Improves performance of location-based searches |
| idx\_event\_category | category\_id | Optimizes category filtering operations |
| idx\_event\_status | status | Enhances queries filtering by event status |
| idx\_event\_suspended | is\_suspended | Speeds up queries that exclude suspended events |

These indexes significantly reduce query execution time for typical operations including loading the home page event listing, executing searches with multiple filter criteria, and retrieving events by specific attributes.

### Data Integrity and Constraints

Referential integrity is enforced through foreign key constraints that prevent orphaned records and ensure that every event references valid organization and category identifiers. The use of the ON UPDATE CURRENT\_TIMESTAMP clause for the updated\_at column automatically maintains modification timestamps without requiring explicit application logic. Character encoding is set to UTF-8 (utf8mb4) to support international characters in event names, descriptions, and organizational information, ensuring global accessibility of the platform.

**API design**

The RESTful API layer provides a well-defined interface for client-server communication, implementing resource-oriented endpoints that adhere to REST architectural principles including statelessness, uniform interface, and proper utilization of HTTP methods and status codes. The API design prioritizes clarity, consistency, and extensibility while maintaining security best practices.

### API Endpoints Overview

The following table summarizes the primary API endpoints implemented in this application:

|  |  |  |  |
| --- | --- | --- | --- |
| HTTP Method | Endpoint | Purpose | Authentication Required |
| GET | /api/events | Retrieve all current and upcoming charity events | No |
| GET | /api/events/search | Search events by date, city, or category | No |
| GET | /api/events/:id | Retrieve detailed information for a specific event | No |
| GET | /api/categories | Retrieve all event categories | No |
| GET | /api/cities | Retrieve all cities with available events | No |

### Detailed Endpoint Specification: GET /api/events/search

Purpose:  
This endpoint enables clients to retrieve a filtered subset of charity events based on specified search criteria including date, geographical location, and event category. The endpoint supports single or multiple filter combinations, allowing users to progressively narrow their search results to locate events matching their specific preferences.

Request Format:

The endpoint accepts query parameters through the URL query string, with all parameters being optional to support flexible search combinations:

GET /api/events/search?date=YYYY-MM-DD&city=CityName&category=CategoryID

Query Parameters:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Type | Required | Description | Example |
| date | String (ISO Date) | No | Filter events by specific date in YYYY-MM-DD format | 2025-11-15 |
| city | String | No | Filter events by city name (partial matching supported) | Sydney |
| category | Integer | No | Filter events by category ID | 2 |

Request Example:

GET /api/events/search?city=Sydney&category=2  
Host: localhost:3000

Response Format:

The endpoint returns a JSON object containing a success indicator, applied filter values, result count, and an array of matching event objects:

{  
 "success": true,  
 "count": 3,  
 "filters": {  
 "date": null,  
 "city": "Sydney",  
 "category": "2"  
 },  
 "data": [  
 {  
 "event\_id": 2,  
 "event\_name": "Green Environment Fun Run",  
 "description": "5km fun run supporting environmental protection projects",  
 "event\_date": "2025-10-20T00:00:00.000Z",  
 "event\_time": "07:00:00",  
 "location": "Hyde Park",  
 "city": "Sydney",  
 "ticket\_price": 35.00,  
 "is\_free": 0,  
 "image\_url": "https://images.unsplash.com/photo-1452626038306-9aae5e071dd3",  
 "status": "upcoming",  
 "category\_name": "Fun Run",  
 "organization\_name": "Green Earth Association"  
 }  
 ]  
}

Response Fields:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| success | Boolean | Indicates whether the request was processed successfully |
| count | Integer | Number of events matching the search criteria |
| filters | Object | Echo of applied filter parameters for client verification |
| data | Array | Collection of event objects matching the search criteria |

Each event object within the data array contains comprehensive information including identification, descriptive content, temporal details, location information, pricing, categorization, and organizational attribution.

Error Responses:

The endpoint implements comprehensive error handling with appropriate HTTP status codes:

// 500 Internal Server Error - Database connection failure  
{  
 "success": false,  
 "message": "Server error",  
 "error": "Database connection refused"  
}

Implementation Details:

The endpoint constructs dynamic SQL queries based on the presence of query parameters, utilizing parameterized queries to prevent SQL injection vulnerabilities. The base query retrieves events with is\_suspended = FALSE and status IN ('upcoming', 'ongoing') to exclude suspended and completed events from search results. Additional WHERE clauses are appended based on provided parameters, with the city parameter utilizing SQL LIKE operator for partial matching to accommodate variations in city name formatting.

The query joins the charity\_events table with event\_categories and charity\_organizations tables to retrieve associated category names and organization names, reducing the number of round-trip requests required by the client. Results are ordered chronologically by event\_date to present events in a logical temporal sequence.

### HTTP Method Justification

GET Method Selection:  
All implemented endpoints utilize the HTTP GET method, which is semantically appropriate for read-only operations that retrieve data without causing server-side state modifications. The GET method offers several advantages for this use case:

1. Idempotency: Multiple identical GET requests produce the same result without side effects, allowing safe request retry mechanisms and browser caching.
2. Cacheability: GET responses can be cached by browsers and intermediate proxies, reducing server load and improving client performance for repeated requests.
3. Bookmarkability: GET requests with query parameters can be bookmarked and shared, enabling users to save specific search configurations or event detail URLs.
4. Simplicity: GET requests can be easily tested through browser address bars and development tools without requiring specialized HTTP clients.

Future Method Extensions:  
While the current implementation focuses exclusively on GET operations as required by the assignment specifications, future enhancements would incorporate:

* POST for creating new events and handling registration submissions
* PUT for updating existing event information
* DELETE for removing events or canceling registrations

These additional methods would follow REST principles by operating on specific resource URIs and returning appropriate status codes (201 Created, 204 No Content, etc.) to indicate operation outcomes.

### API Security Considerations

The current implementation includes several security measures appropriate for the read-only nature of the application:

1. CORS Configuration: The API enables Cross-Origin Resource Sharing through the cors middleware, allowing the client application hosted on a different port to access API endpoints while preventing unauthorized cross-origin requests from unknown domains in production deployments.
2. Input Validation: Query parameters are validated and sanitized before incorporation into database queries, with parameterized query construction preventing SQL injection attacks.
3. Error Message Sanitization: Error responses provide sufficient information for debugging during development while avoiding exposure of sensitive implementation details such as database schema information or server configuration.
4. Connection Pool Management: Database connections are managed through a connection pool with configured limits, preventing resource exhaustion attacks and ensuring stable performance under concurrent load.

Future security enhancements would include authentication mechanisms, rate limiting, input validation libraries, and comprehensive logging for audit trail purposes as the application expands to include write operations and user account management.

## Conclusion

This project successfully demonstrates the practical application of full-stack web development technologies to create a functional and user-centric platform for charitable event management. The implementation showcases proficiency in database design through a normalized relational schema, server-side development through well-architected RESTful APIs, and client-side development through dynamic JavaScript-based interfaces that provide intuitive user experiences. The project fulfills all specified requirements while establishing a solid foundation for future enhancements including user authentication, event registration processing, and administrative management capabilities.