1. Introduce the Project and Its Features

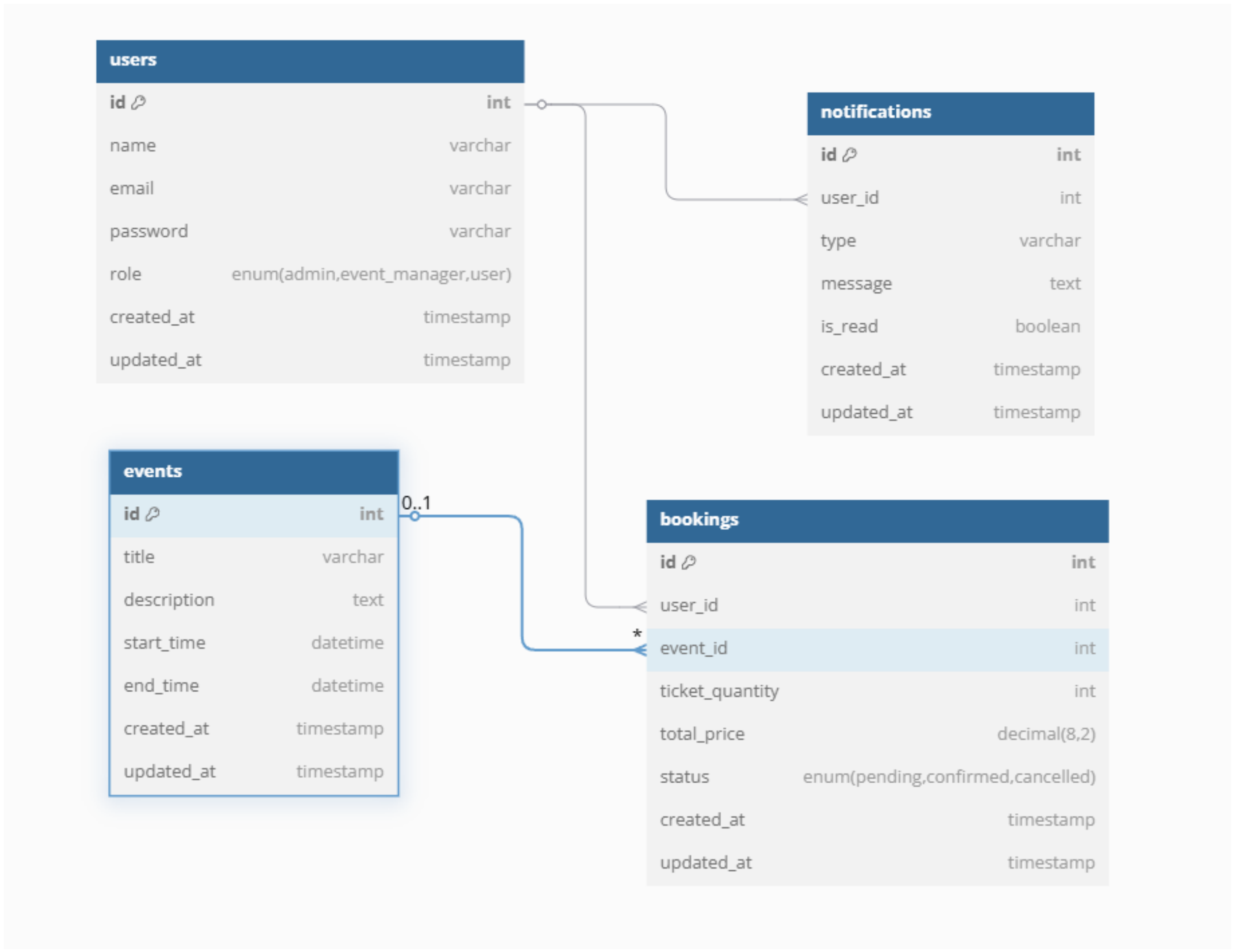
Project Overview

- **Backend (Laravel API):**
  - Provides RESTful APIs for the frontend (Vue.js) to interact with.
  - Handles authentication, database operations, and third-party integrations (e.g., Twilio, Pusher).
- **Frontend (Vue.js):**
  - A user-facing application for browsing events, making bookings, etc.
  - An admin dashboard for managing events, bookings, and notifications.

Key Features

- **User Features:**
  - User registration and login.
  - Browse events and book tickets.
  - Receive notifications (e.g., booking confirmation).
- **Admin Features:**
  - Manage events (create, update, delete).
  - View bookings and send notifications.
- **Technical Features:**
  - Authentication using Laravel Sanctum.
  - Real-time notifications using Pusher.
  - SMS notifications using Twilio.

Entity-Relationship Diagram (ERD)



Explanation of Relationships

1. `users` **Table:**
   - **Primary Key (PK):** `id`
   - **Relationships:**
     - A user can have many `bookings` (one-to-many relationship).
     - A user can have many `notifications` (one-to-many relationship).
2. `events` **Table:**
   - **Primary Key (PK):** `id`
   - **Relationships:**
     - An event can have many `bookings` (one-to-many relationship).
3. `bookings` **Table:**
   - **Primary Key (PK):** `id`
   - **Foreign Keys (FK):**
     - `user_id` references `users(id)`.
     - `event_id` references `events(id)`.

- **Relationships:**
    - A booking belongs to one `user` (many-to-one relationship).
    - A booking belongs to one `event` (many-to-one relationship).

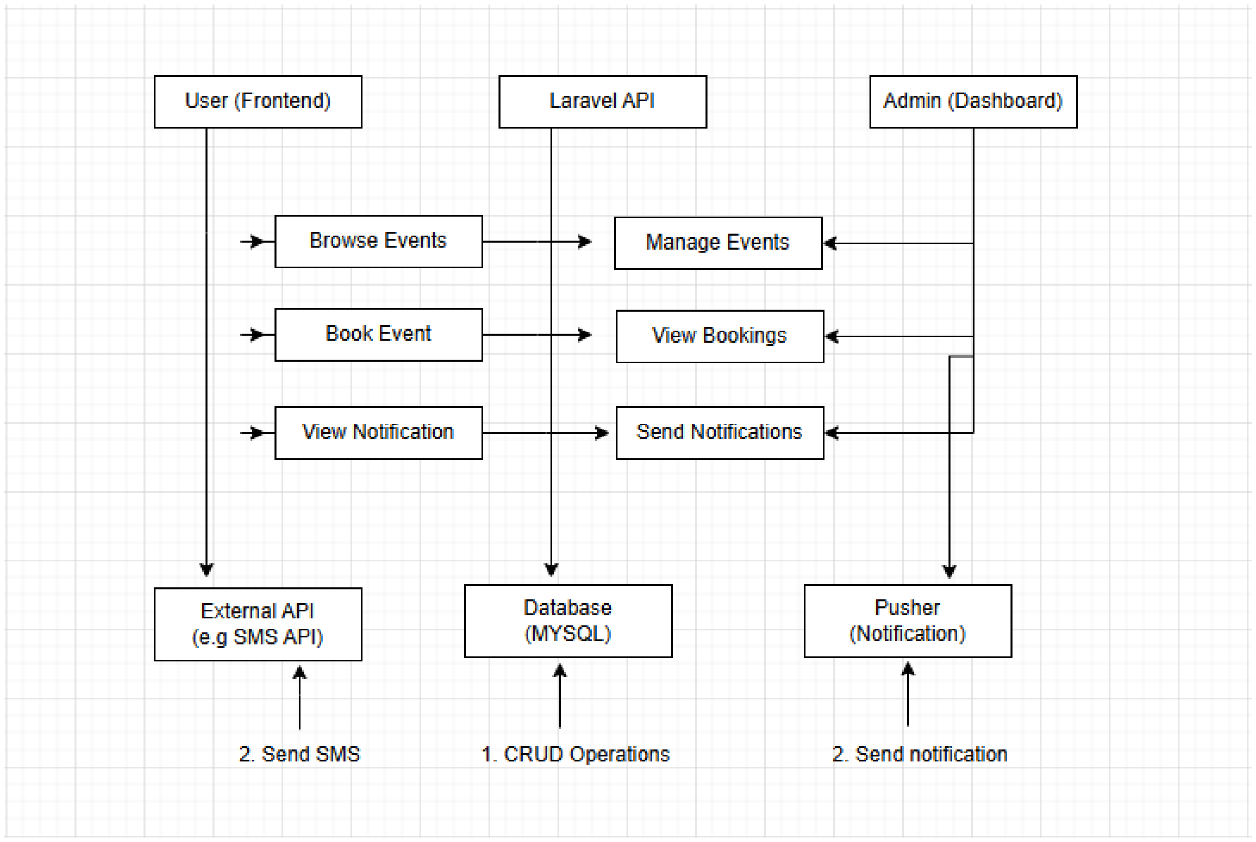
4. **`notifications` Table:**
    - **Primary Key (PK):** `id`
    - **Foreign Key (FK):**
        - `user_id` references `users(id)`.
    - **Relationships:**
        - A notification belongs to one `user` (many-to-one relationship).

Relationships in SQL (Foreign Keys)

Here's how the relationships are implemented in the database schema:

1. **`bookings` Table:**
    - `user_id` is a foreign key referencing `users(id)`.
    - `event_id` is a foreign key referencing `events(id)`.
2. **`notifications` Table:**
    - `user_id` is a foreign key referencing `users(id)`.

DFD



This diagram breaks down the **Laravel API** and **Vue.js Frontend/Admin Dashboard** into more detailed processes.

Vue.js Frontend (User)

1. **Browse Events:**
   - User requests event data from the Laravel API.
   - Laravel API fetches event data from the database and returns it to the frontend.
2. **Book Event:**
   - User submits a booking request.
   - Frontend sends the booking data to the Laravel API.
   - Laravel API processes the booking, updates the database, and sends a confirmation notification via Pusher/Twilio.

3. **View Notifications:**
    - User requests notifications from the Laravel API.
    - Laravel API fetches notifications from the database and returns them to the frontend.

Vue.js Admin Dashboard

1. **Manage Events:**
    - Admin creates, updates, or deletes events.
    - Frontend sends event data to the Laravel API.
    - Laravel API updates the database and returns a response.
2. **View Bookings:**
    - Admin requests booking data from the Laravel API.
    - Laravel API fetches booking data from the database and returns it to the admin dashboard.
3. **Send Notifications:**
    - Admin sends notifications to users.
    - Frontend sends notification data to the Laravel API.
    - Laravel API stores the notification in the database and sends it via Pusher/Twilio.

Laravel API (Backend)

1. **Authentication:**
    - Handles user login and registration.
    - Uses Laravel Sanctum for API token-based authentication.
2. **Database Operations:**
    - Manages CRUD operations for `users`, `events`, `bookings`, and `notifications`.
3. **Third-Party Integrations:**
    - Sends real-time notifications using **Pusher**.
    - Sends SMS notifications using **Twilio**.

Explanation of Data Flows

User (Frontend)

1. **Browse Events:**
   - User → Laravel API: Request event data.
   - Laravel API → Database: Fetch event data.
   - Database → Laravel API: Return event data.
   - Laravel API → User: Display event data.
2. **Book Event:**
   - User → Laravel API: Submit booking data.
   - Laravel API → Database: Store booking data.
   - Laravel API → Pusher/Twilio: Send confirmation notification.
   - Pusher/Twilio → User: Receive notification.
3. **View Notifications:**
   - User → Laravel API: Request notifications.
   - Laravel API → Database: Fetch notifications.
   - Database → Laravel API: Return notifications.
   - Laravel API → User: Display notifications.

Admin (Dashboard)

1. **Manage Events:**
   - Admin → Laravel API: Submit event data (create/update/delete).
   - Laravel API → Database: Update event data.
   - Database → Laravel API: Return success/failure response.
   - Laravel API → Admin: Display response.
2. **View Bookings:**
   - Admin → Laravel API: Request booking data.
   - Laravel API → Database: Fetch booking data.
   - Database → Laravel API: Return booking data.
   - Laravel API → Admin: Display booking data.
3. **Send Notifications:**
   - Admin → Laravel API: Submit notification data.

    - `Laravel API → Database: Store notification data.`
    - `Laravel API → Pusher/Twilio: Send notification.`
    - `Pusher/Twilio → User: Receive notification.`

Setup laravel & Vuejs

Laravel Setup (Backend)

1. Install Laravel
2. Configure Environment
3. Serve the application

Vue.js Setup (Frontend)

1. Install Vue.js
2. Serve the application

Create Git Repository

1. Backend
2. Frontend

Define the Database Schema

1. php artisan make:migration create_users_table
2. php artisan make:migration create_events_table
3. php artisan make:migration create_bookings_table
4. php artisan make:migration create_notifications_table

Resulting Database Schema

After running the migrations, your database will have the following tables:

1. **users**: Stores user information.
2. **events**: Stores event information.
3. **bookings**: Stores booking information with relationships to `users` and `events`.
4. **notifications**: Stores notifications with a relationship to `users`.

Relationships in Models

User Model

1. Users have many bookings.
2. Users have many notifications

Event model

1. Events have many bookings

Booking model

1. Booking belongs to user
2. booking belongs to event

Notification model

1. Notifications belongs to use

1. **Class 2: Backend Development (2 Hours)**
   - Create Seeders.
   - Set up API routes and controllers.
   - Implement basic CRUD operations.
   - Test the API using Postman.