WeCare - Technical Design Document

Table of Contents :-

1. Introduction

WeCare is a unified platform designed to help communities discover, share, and participate in local social events and initiatives. The platform aims to facilitate meaningful contributions to local causes and enhance community engagement through streamlined event management and transparent communication.

Core Features

- Event discovery and creation
- Community engagement tracking
- User profile management
- Real-time event updates
- Social sharing capabilities
- Volunteer management system
- Donation tracking and management

Target Audience

- Community members
- Event organizers
- Local government representatives
- Non-profit organizations
- Volunteers and social workers

2. System Overview

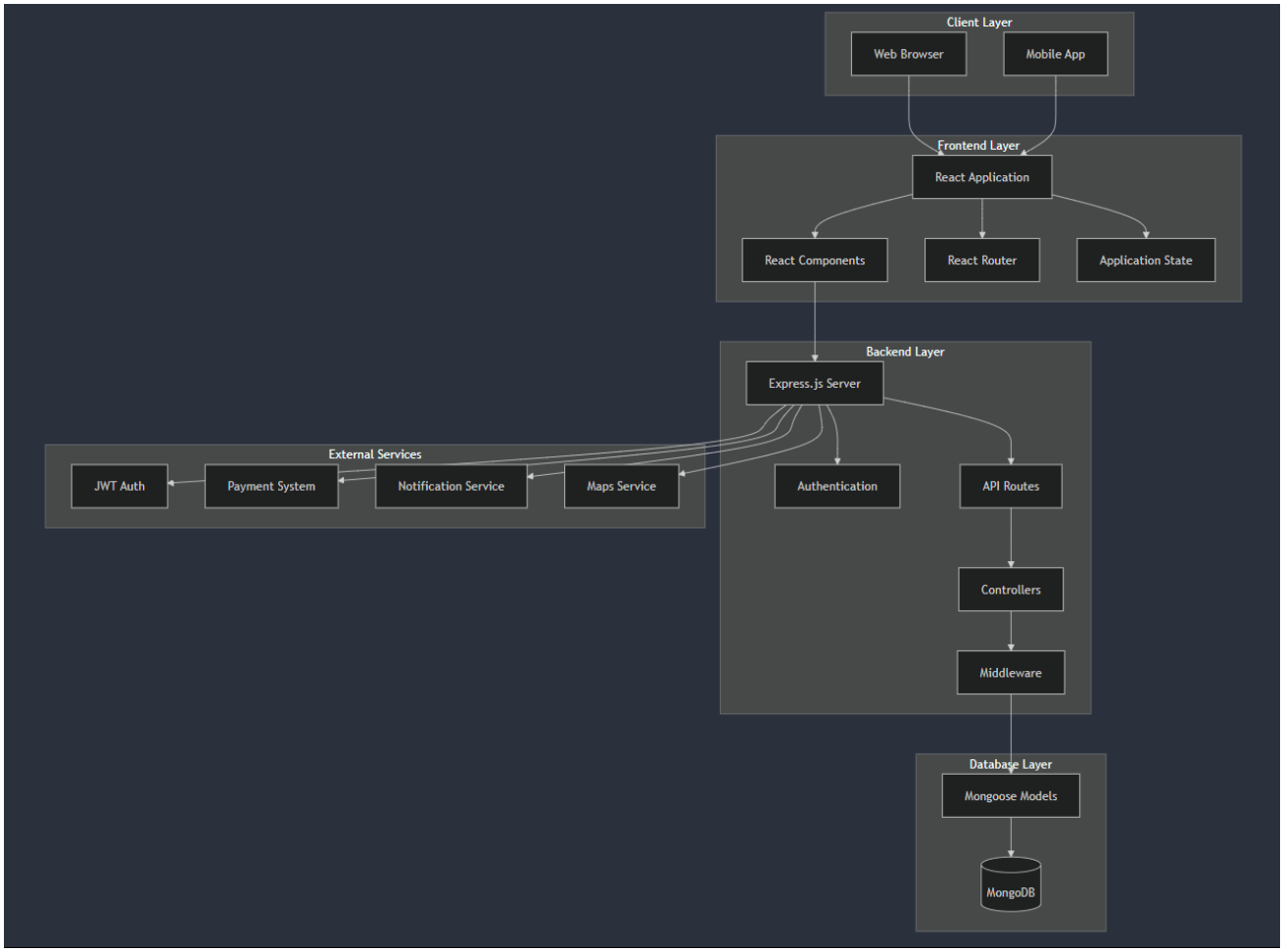
WeCare is built using a modern tech stack that includes React for the frontend and Node.js with Express for the backend. The application is designed to be scalable, maintainable, and responsive, ensuring a seamless user experience across various devices.

3. Architecture

High-Level Architecture

WeCare implements a modern three-tier architecture designed for scalability and maintainability:

1. **Presentation Layer (Frontend)**
    - React.js-based user interface
    - Responsive design implementation
    - Client-side state management
    - Progressive Web App (PWA) capabilities



2. **Application Layer (Backend)**
    - Node.js and Express.js server
    - RESTful API implementation
    - Business logic processing
    - Authentication and authorization
3. **Data Layer**
    - MongoDB database
    - Data persistence
    - Data validation
    - Indexing and optimization

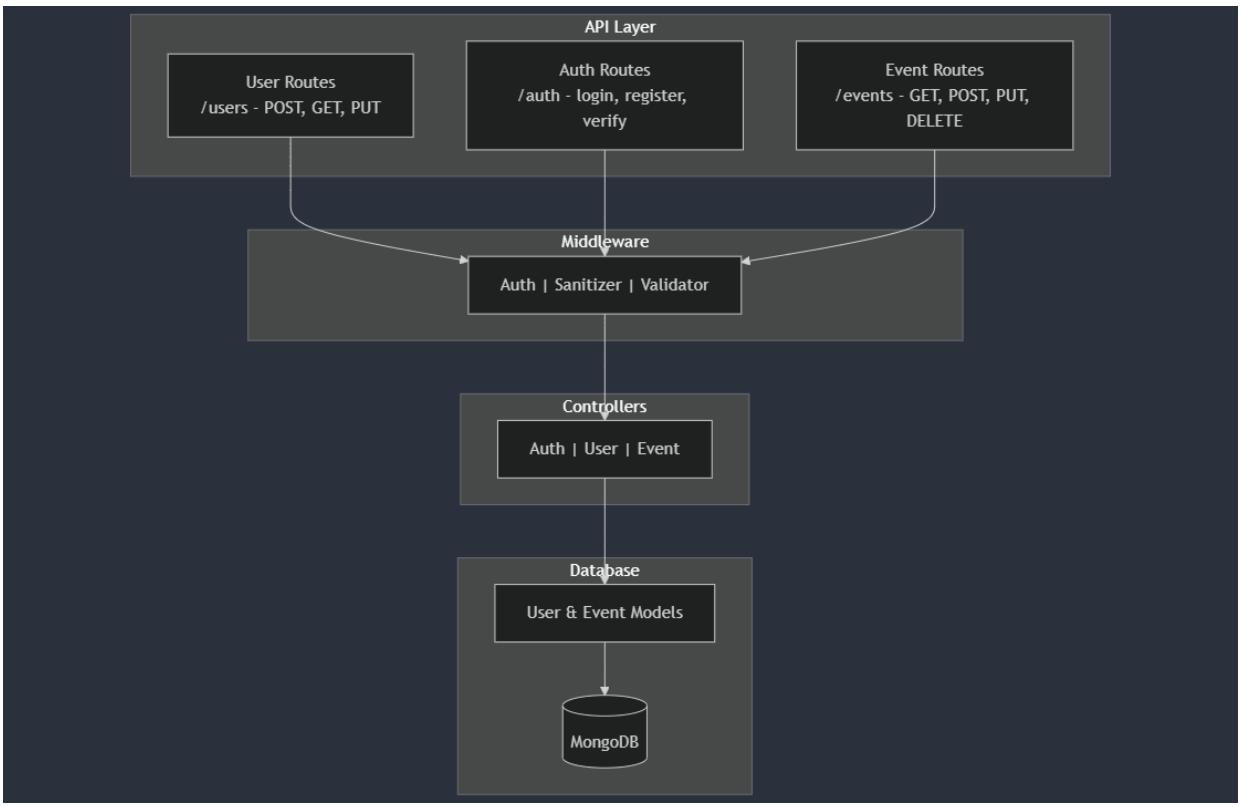
Component Interactions

- The frontend communicates with the backend through RESTful API calls.
- The backend interacts with the MongoDB database for data storage and retrieval.

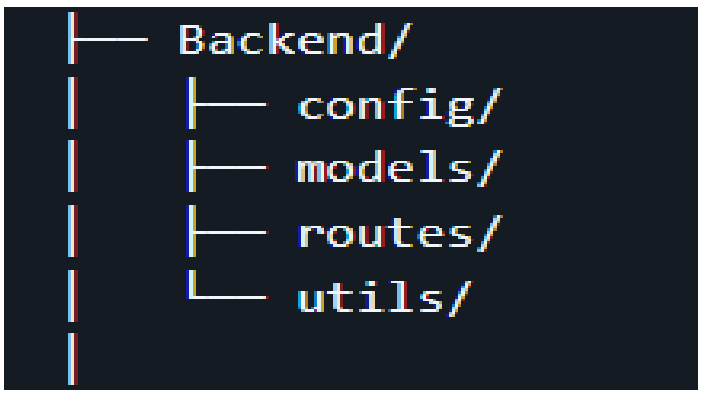
4. Backend Design

Technologies Used

- **Node.js**: JavaScript runtime environment.
- **Express.js**: Web application framework for building APIs.
- **MongoDB**: NoSQL database for data storage.
- **Mongoose**: ODM library for MongoDB.
- **Bcrypt**: Library for hashing passwords.
- **JWT**: JSON Web Tokens for authentication.
- **Multer**: Middleware for handling file uploads.



Project Structure :-



API Design

The backend exposes RESTful API endpoints categorized under:

- **Authentication (/api/auth)**

```
POST /api/auth/register
POST /api/auth/login
POST /api/auth/refresh-token
POST /api/auth/logout
```

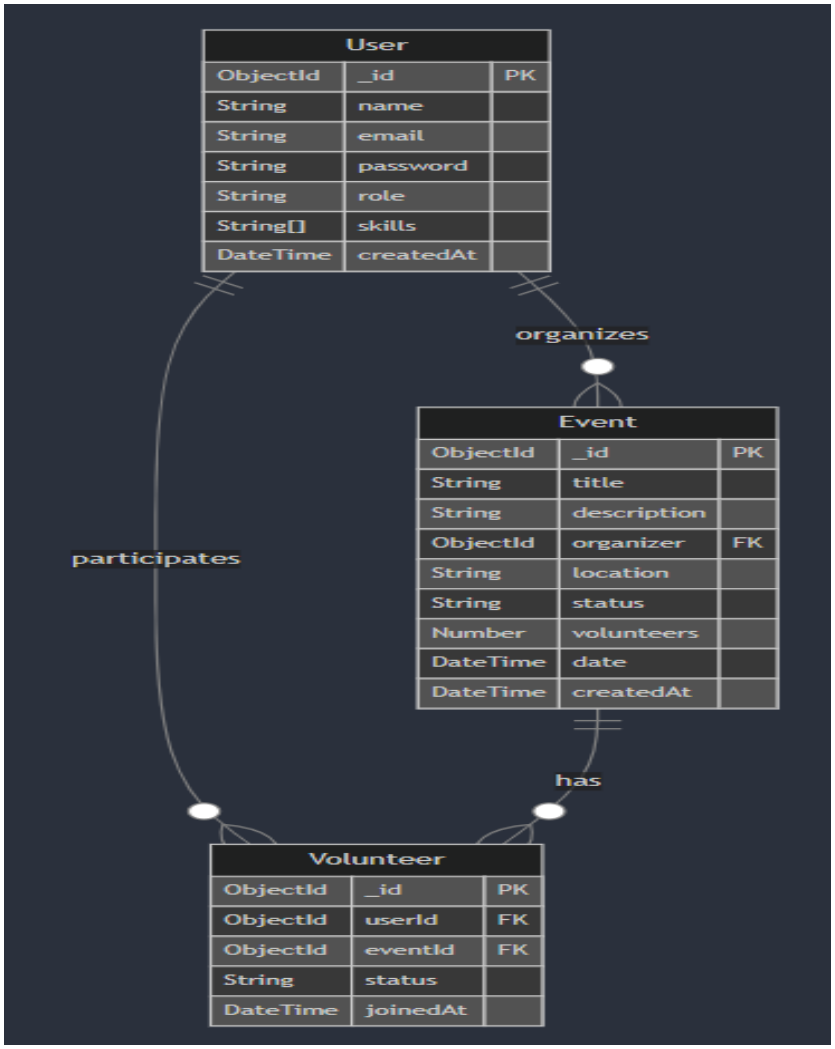
- **Events (/api/events)**

```
GET    /api/events
POST   /api/events
GET    /api/events/:id
PUT    /api/events/:id
DELETE /api/events/:id
```

- User Management

```
GET    /api/users
GET    /api/users/:id
PUT    /api/users/:id
DELETE /api/users/:id
```

Database Schema



User Model (User .js)

```
const userSchema = new Schema({
  name: {
    type: String,
    required: true,
    trim: true
  },
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true
  },
  password: {
    type: String,
    required: true,
    minlength: 8
  },
  role: {
    type: String,
    enum: ['user', 'organizer', 'admin'],
    default: 'user'
  },
  events: [{
    type: Schema.Types.ObjectId,
    ref: 'Event'
  }],
  createdAt: {
    type: Date,
    default: Date.now
  }
});
```

Event Model (Event.js)

```
const eventSchema = new Schema({
  title: {
    type: String,
    required: true,
    trim: true
  },
  description: {
    type: String,
    required: true
  },
  date: {
    type: Date,
    required: true
  },
  location: {
    address: String,
    city: String,
    state: String,
    coordinates: {
      type: [Number],
      index: '2dsphere'
    }
  },
  organizer: {
    type: Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  participants: [{
    user: {
      type: Schema.Types.ObjectId,
      ref: 'User'
    },
    status: {
      type: String,
      enum: ['registered', 'attended', 'cancelled'],
      default: 'registered'
    }
  }],
  category: {
    type: String,
    required: true,
    enum: ['social', 'environmental', 'educational', 'health', 'other']
  }
});
```

Middleware

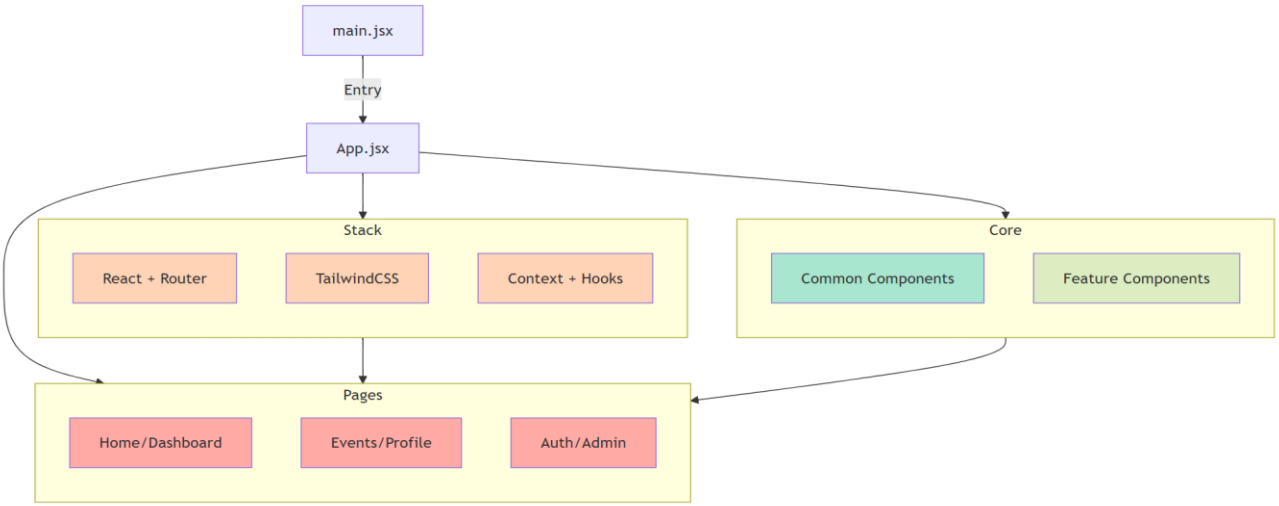
Authentication Middleware (auth.js)

- Validates JWT tokens sent in the Authorization header.
- Attaches the authenticated user's information to the request object.

5. Frontend Design

Technologies Used

- **React.js**: JavaScript library for building user interfaces.
- **React Router**: Handling client-side routing.
- **Tailwind CSS**: Utility-first CSS framework for styling.
- **Vite**: Build tool for faster development.



Project Structure

```
WeCare/
└── Frontend/
    ├── public/
    └── src/
        ├── components/
        ├── Layout/
        ├── assets/
        └── routes/
```

```
└── adminFrontend/
    └── src/
        ├── components/
        ├── Layout/
        ├── assets/
        └── routes/
```

Routing

Implemented using React Router:

```
/: Home page.
/events: List of events.
/events/:id: View a specific event.
/create-event: Create a new event.
/login: User login page.
/register: User registration page.
```

State Management

- Local State: Managed using useState and useEffect hooks.
- Authentication State: Stored in localStorage and accessed via custom hooks.

Key Components

- **EventList**: Displays a list of events with filtering options.
- **EventDetail**: Shows details of a specific event.
- **NavBar**: Navigation links based on user authentication state.

6. Security Considerations

Authentication Flow

1. User registration with email verification
2. JWT-based authentication
3. Refresh token rotation
4. Session management

Security Measures

- Password hashing using bcrypt
- Rate limiting on sensitive endpoints
- CORS configuration
- XSS protection
- CSRF tokens
- Input validation and sanitization

7. Deployment Plan

Environment Setup

- **Backend Environment Variables**:
    - **MONGO_URI**: MongoDB connection string.
    - **JWT_SECRET_KEY**: Secret key for signing JWTs.

Deployment Steps

1. **Backend Deployment**:
    - Host on platforms like Heroku, AWS EC2, or DigitalOcean.
    - Ensure environment variables are securely set.
    - Use process managers like PM2 for process management.

2. **Frontend Deployment**:
    - Build the React application using **npm run build**.
    - Host static files on services like Vercel or AWS.
3. **Domain and SSL**:
    - Configure a custom domain.
    - Set up SSL certificates for secure HTTPS communication.
4. **Database Hosting**:
    - Use managed MongoDB services like MongoDB Atlas.
    - Configure IP whitelisting and security measures.
5. **Continuous Deployment**:
    - Set up CD pipelines to automatically deploy on code changes.
    - Use GitHub Actions or other CI/CD tools.

8. Unit Testing

Unit Testing

- Jest for backend testing
- React Testing Library for frontend
- Component isolation testing
- API endpoint testing

Integration Testing

- End-to-end testing with Cypress
- API integration testing
- Database operations testing
- Authentication flow testing

Performance Testing

- Load testing with Artillery
- Stress testing scenarios
- Response time benchmarking
- Database query optimization

9. Future Plans

- **Enhanced Global Reach**: Expand services across states and countries, implement multilingual support, and create region-specific features.
- **Revolutionary Payment Solutions**: Integrate blockchain technology for donation tracking, implement smart contracts, and support cryptocurrency donations.
- **Technology Advancement**: Develop a mobile application using Flutter, implement AI-powered matching algorithms, and create advanced analytics tools.
- **Community Features**: Introduce a volunteer management system, create a platform for success stories, and develop tools for organizing local fundraising events.

10. Conclusion

WeCare represents a modern, scalable solution for community engagement, built with best practices in mind. The platform's architecture ensures:

- Scalability for growing user base
- Maintainable and clean codebase
- Secure user data handling
- Optimal performance
- Extensible feature set

This technical design document serves as a living guide for development teams and stakeholders, ensuring alignment with project goals and technical requirements.