Event-Go Project Code Explanation

Table of Contents

1. Project Structure
2. Core Components
3. API Layer
4. Database Layer
5. Authentication
6. State Management
7. UI Components
8. Utilities

Project Structure

Directory Layout

```
event-go/
├── src/
│   ├── app/                    # Next.js app router pages
│   │   ├── auditoriums/       # Auditorium pages
│   │   ├── shows/            # Show pages
│   │   └── layout.tsx        # Root layout
│   ├── components/           # Reusable components
│   │   ├── atoms/           # Basic UI components
│   │   ├── molecules/       # Composite components
│   │   └── templates/       # Page templates
│   ├── trpc/                # API client setup
│   │   ├── clients/        # tRPC clients
│   │   └── server/         # tRPC server
│   ├── server/             # Backend server code
│   │   ├── router/        # tRPC routers
│   │   └── middleware/    # Custom middleware
│   ├── store/             # Redux store
│   ├── types/             # TypeScript types
│   ├── schema/            # Zod schemas
│   ├── forms/             # Form components
│   ├── lib/               # Utility libraries
│   └── util/              # Helper functions
├── prisma/                # Database schema
└── public/               # Static assets
```

Core Components

1. Auditorium Page (`src/app/auditoriums/[city]/page.tsx`)

```
// Main component for city-specific auditorium view
export default function AuditoriumsPage({ params }: AuditoriumsPageProps)
{
  // State management
  const [lat, setLat] = useState<number | null>(null)
  const [lng, setLng] = useState<number | null>(null)
  const [cityName, setCityName] =
useState(decodeURIComponent(resolvedParams.city))

  // API queries
  const { data: coordinates } =
trpcClient.geocoding.getCoordinates.useQuery(...)
  const { data: showsData } = trpcClient.shows.shows.useQuery(...)

  // Event handlers
  const handleBookNow = (show: Show) => {...}

  // Render UI
  return (
    <div>
      <HeroSection />
      <CategoryFilter />
      <ShowsList />
      <BookingDialog />
    </div>
  )
}
```

2. Show Page (`src/app/shows/[id]/page.tsx`)

```
// Main component for individual show view
export default function ShowPage({ params }: ShowPageProps) {
  // State management
  const [selectedShowtime, setSelectedShowtime] = useState<Showtime |
null>(null)

  // API queries
  const { data: show } = trpcClient.shows.show.useQuery(...)
  const { data: showtimes } = trpcClient.showtimes.showtimes.useQuery(...)

  // Event handlers
  const handleShowtimeSelect = (showtime: Showtime) => {...}

  // Render UI
  return (
    <div>
      <ShowDetails />
      <ShowtimeSelector />
      <BookingSection />
    </div>
  )
}
```

API Layer

1. Shows Router (`src/trpc/server/router/shows.ts`)

```
export const showsRouter = router({
  shows: publicProcedure
    .input(z.object({
      lat: z.number().optional(),
      lng: z.number().optional(),
      city: z.string().optional()
    }))
    .query(async ({ input }) => {
      // Fetch and filter shows
      const shows = await prisma.show.findMany({
        where: {...},
        include: {...}
      })

      return {
        matchingShows: [...],
        allShows: [...],
        hasNearbyShows: true
      }
    })
})
```

2. Showtimes Router (`src/trpc/server/router/showtimes.ts`)

```
export const showtimesRouter = router({
  showtimes: publicProcedure
    .input(z.object({
      where: z.object({
        Show: z.object({ id: z.number() }).optional()
      }).optional()
    }))
    .query(async ({ input }) => {
      // Fetch showtimes
      return await prisma.showtime.findMany({
        where: input.where,
        include: {...}
      })
    })
})
```

Database Layer

1. Schema (`prisma/schema.prisma`)

```
model Show {
  id          Int        @id @default(autoincrement())
  title       String
  genre       Genre
  duration    Int
  releaseDate DateTime
  posterUrl   String?
  showtimes   Showtime[]
}

model Showtime {
  id        Int      @id @default(autoincrement())
  show      Show     @relation(fields: [showId], references: [id])
  showId    Int
  screen    Screen   @relation(fields: [screenId], references: [id])
  screenId  Int
  startTime DateTime
  bookings  Booking[]
}

model Screen {
  id           Int        @id @default(autoincrement())
  number       Int
  auditorium   Auditorium @relation(fields: [auditoriumId], references:
[id])
  auditoriumId Int
  showtimes    Showtime[]
}

model Auditorium {
  id        Int      @id @default(autoincrement())
  name      String
  address   Address  @relation(fields: [addressId], references: [id])
  addressId Int
  screens   Screen[]
}

model Address {
  id       Int          @id @default(autoincrement())
  address String
  lat      Float
  lng      Float
  auditoriums Auditorium[]
}
```

Authentication

1. Clerk Setup (`src/lib/clerk.ts`)

```
import { ClerkProvider } from '@clerk/nextjs'

export function AuthProvider({ children }: { children: React.ReactNode })
{
  return (
    <ClerkProvider
      publishableKey={process.env.NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY}
    >
      {children}
    </ClerkProvider>
  )
}
```

2. Protected Routes (`src/middleware.ts`)

```
import { authMiddleware } from '@clerk/nextjs'

export default authMiddleware({
  publicRoutes: ['/'],
  ignoredRoutes: ['/api/trpc/shows.shows']
})
```

State Management

1. Redux Store (`src/store/index.ts`)

```
import { configureStore } from '@reduxjs/toolkit'
import showsReducer from './showsSlice'
import bookingReducer from './bookingSlice'

export const store = configureStore({
  reducer: {
    shows: showsReducer,
    booking: bookingReducer
  }
})
```

2. Shows Slice (`src/store/showsSlice.ts`)

```
import { createSlice } from '@reduxjs/toolkit'

const showsSlice = createSlice({
  name: 'shows',
  initialState: {
    selectedShowId: null,
    selectedScreenId: null,
    selectedShowtimeId: null
  },
  reducers: {
    selectShow: (state, action) => {
      state.selectedShowId = action.payload
    }
  }
})
```

UI Components

1. Atoms (`src/components/atoms/`)

```
// Button.tsx
export function Button({ children, ...props }: ButtonProps) {
  return (
    <button
      className="bg-primary text-white px-4 py-2 rounded"
      {...props}
    >
      {children}
    </button>
  )
}

// Dialog.tsx
export function Dialog({ children, ...props }: DialogProps) {
  return (
    <div className="fixed inset-0 bg-black bg-opacity-50">
      <div className="bg-white rounded-lg p-4">
        {children}
      </div>
    </div>
  )
}
```

2. Molecules (`src/components/molecules/`)

```
// ShowCard.tsx
export function ShowCard({ show }: ShowCardProps) {
  return (
    <div className="bg-white rounded-lg shadow">
      <Image src={show.posterUrl} alt={show.title} />
      <div className="p-4">
        <h3>{show.title}</h3>
        <p>{show.duration} mins</p>
      </div>
    </div>
  )
}

// BookingStepper.tsx
export function BookingStepper({ show, onClose }: BookingStepperProps) {
  const [step, setStep] = useState(1)

  return (
    <div>
      {step === 1 && <ShowtimeSelector />}
      {step === 2 && <SeatSelector />}
      {step === 3 && <PaymentForm />}
    </div>
  )
}
```

Utilities

1. API Client (`src/trpc/clients/client.ts`)

```
import { createTRPCNext } from '@trpc/next'
import { httpBatchLink } from '@trpc/client'
import type { AppRouter } from '@/server/router'

export const trpcClient = createTRPCNext<AppRouter>({
  config() {
    return {
      links: [
        httpBatchLink({
          url: '/api/trpc'
        })
      ]
    }
  }
})
```

2. Form Validation (`src/schema/show.ts`)

```
import { z } from 'zod'

export const showSchema = z.object({
  title: z.string().min(1),
  genre: z.enum(['CULTURAL', 'CONFERENCE', 'SPORTS']),
  duration: z.number().min(1),
  releaseDate: z.date()
})
```

Data Flow

1. Show Discovery

```
graph TD
    A[User visits city page] --> B[Fetch coordinates]
    B --> C[Query shows]
    C --> D[Filter by location]
    D --> E[Display shows]
```

2. Booking Process

```
graph TD
    A[Select show] --> B[Choose showtime]
    B --> C[Select seats]
    C --> D[Enter payment]
    D --> E[Confirm booking]
```

Error Handling

1. API Errors

```
try {
  const result = await trpcClient.shows.shows.query(...)
} catch (error) {
  toast.error('Failed to fetch shows')
  console.error('Error:', error)
}
```

2. Form Validation

```
const result = showSchema.safeParse(data)
if (!result.success) {
  const errors = result.error.format()
  // Handle validation errors
}
```

Performance Optimizations

1. Image Loading

```
<Image
  src={show.posterUrl}
  alt={show.title}
  loading="lazy"
  sizes="(max-width: 768px) 100vw, 50vw"
/>
```

2. Query Caching

```
const { data } = trpcClient.shows.shows.useQuery(
  { city },
  {
    staleTime: 5 * 60 * 1000, // 5 minutes
    cacheTime: 30 * 60 * 1000 // 30 minutes
  }
)
```

Security Measures

1. Input Sanitization

```
const sanitizedCity = decodeURIComponent(city)
  .replace(/[<>]/g, '')
```

2. API Protection

```
const isAuthorized = await clerk.verifyToken(token)
if (!isAuthorized) {
  throw new TRPCError({ code: 'UNAUTHORIZED' })
}
```

Testing

1. Unit Tests

```
describe('Show filtering', () => {
  it('filters shows by city', () => {
    const result = filterShowsByCity(shows, 'Mumbai')
    expect(result.length).toBeGreaterThan(0)
  })
})
```

2. Integration Tests

```
describe('Booking flow', () => {
  it('completes booking process', async () => {
    // Test booking flow
  })
})
```

Deployment

1. Docker Configuration

```
version: '3.8'
services:
  app:
    build: .
    environment:
      DATABASE_URL: ${DATABASE_URL}
      NEXT_PUBLIC_CLERK_KEY: ${CLERK_KEY}
```

2. Environment Variables

```
DATABASE_URL=postgresql://user:password@localhost:5432/eventgo
NEXT_PUBLIC_CLERK_KEY=your_clerk_key
```

Conclusion

This comprehensive code explanation covers the entire Event-Go project, including its architecture, components, data flow, and various technical aspects. The project is built with modern web technologies and follows best practices for maintainability, performance, and security.