AdventureConnect MVP - Architecture Overview

🏗️ Project Structure

```
adventureconnect-mvp/
├── backend/                    # Node.js Express API
│   ├── src/
│   │   ├── config/            # Configuration files
│   │   │   ├── database.js    # PostgreSQL connection & migrations
│   │   │   ├── amadeus.js     # Amadeus API configuration
│   │   │   └── auth.js        # JWT token management
│   │   ├── controllers/       # Business logic
│   │   │   ├── authController.js      # User authentication
│   │   │   ├── providerController.js  # Provider profile management
│   │   │   ├── tripController.js      # Trip CRUD operations
│   │   │   ├── bookingController.js   # Booking management
│   │   │   └── amadeusController.js   # Flight/hotel search (future)
│   │   ├── middleware/        # Express middleware
│   │   │   ├── auth.js        # Authentication & authorization
│   │   │   ├── upload.js      # Multer file upload config
│   │   │   ├── validation.js  # Request validation rules
│   │   │   └── errorHandler.js # Centralized error handling
│   │   ├── routes/            # API route definitions
│   │   ├── utils/             # Helper functions
│   │   │   ├── email.js       # Email templates & sending
│   │   │   └── apiResponse.js # Standardized API responses
│   │   ├── scripts/           # Database scripts
│   │   │   └── seed.js        # Sample data seeding
│   │   └── app.js             # Express app configuration
│   ├── uploads/               # User uploaded images
│   ├── server.js              # Server entry point
│   ├── Dockerfile             # Docker configuration
│   ├── package.json           # Dependencies & scripts
│   └── .env                   # Environment variables
│
├── frontend/                   # React SPA
│   ├── src/
│   │   ├── components/        # Reusable UI components
│   │   │   ├── common/        # Layout, Navbar, Footer, etc.
│   │   │   ├── trips/         # Trip-related components
│   │   │   └── provider/      # Provider-specific components
│   │   ├── contexts/          # React Context providers
│   │   │   └── AuthContext.js # Global authentication state
│   │   ├── pages/             # Page components (routes)
│   │   │   ├── provider/      # Provider dashboard, create trip, etc.
│   │   │   ├── traveler/      # Traveler dashboard, bookings
│   │   │   └── ...           # Home, Login, Register, etc.
│   │   ├── services/          # API communication layer
│   │   │   ├── api.js         # Axios configuration
│   │   │   ├── auth.js        # Authentication services
│   │   │   ├── trips.js        # Trip-related API calls
│   │   │   └── bookings.js     # Booking API calls
│   │   ├── hooks/              # Custom React hooks
│   │   ├── utils/              # Helper functions & constants
│   │   ├── App.js              # Main app component with routing
│   │   └── index.js            # React entry point
│   ├── public/                 # Static assets
│   ├── Dockerfile              # Docker configuration
│   ├── nginx.conf              # Nginx configuration for production
│   ├── tailwind.config.js      # Tailwind CSS configuration
│   └── package.json            # Dependencies & scripts
│
├── .github/
│   └── workflows/
│       └── ci.yml              # GitHub Actions CI/CD pipeline
├── docker-compose.yml          # Multi-container Docker setup
├── setup.sh                    # Quick setup script
├── .gitignore                  # Git ignore patterns
└── README.md                   # Project documentation
```

🔄 Data Flow Architecture

1. Authentication Flow

```
User Registration/Login
    ↓
Frontend (React) → POST /api/auth/register or /login
    ↓
Backend validates credentials
    ↓
Generate JWT token
    ↓
Return token + user data
    ↓
Frontend stores token in localStorage
    ↓
Token included in all subsequent API requests
```

2. Trip Booking Flow

```
Traveler browses trips → GET /api/trips
    ↓
Views trip details → GET /api/trips/:id
    ↓
Sends booking inquiry → POST /api/bookings/inquiry
    ↓
Email sent to provider
    ↓
Provider reviews inquiry → GET /api/bookings/:id
    ↓
Provider responds → PUT /api/bookings/:id/status
    ↓
Email sent to traveler
    ↓
Manual payment processing (Phase 1)
```

🗄️ Database Schema

Core Tables:

- **users**: Authentication & basic info
- **provider_profiles**: Extended provider information
- **trips**: Trip listings created by providers
- **trip_dates**: Available dates for each trip
- **bookings**: Booking inquiries and their status
- **reviews**: Future feature for rating system

Key Relationships:

- User (1) → Provider Profile (1)
- Provider (1) → Trips (Many)
- Trip (1) → Trip Dates (Many)
- Trip (1) → Bookings (Many)
- Traveler (1) → Bookings (Many)

🔐 Security Implementation

1. **Authentication**: JWT tokens with 7-day expiration
2. **Password Security**: bcrypt hashing with salt rounds
3. **Authorization**: Role-based (provider vs traveler)
4. **API Security**:

- Helmet.js for security headers
- CORS configuration
- Rate limiting (100 requests/15 min)

5. **Input Validation**: Express-validator on all endpoints
6. **File Upload**: Type & size restrictions
7. **SQL Injection Prevention**: Parameterized queries

🚀 Deployment Architecture

Development:

```
PostgreSQL (local) ← Backend (localhost:5000) → Frontend (localhost:3000)
```

Production (recommended):

```
               ┌─────────────┐
               │    Nginx    │
               │(Reverse Proxy)│
               └──────┬──────┘
                      │
        ┌─────────────┴─────────────┐
        │                           │
 ┌──────┴──────┐             ┌──────┴──────┐
 │  Frontend   │             │   Backend   │
 │  (React)    │             │  (Express)  │
 └─────────────┘             └──────┬──────┘
                                    │
                             ┌──────┴──────┐
                             │ PostgreSQL  │
                             │  Database   │
                             └─────────────┘
```

📧 Email System

Email notifications are sent for:

- User registration (welcome email)
- New booking inquiry (to provider)
- Booking confirmation (to traveler)
- Booking status updates

Currently using Nodemailer with SMTP. For production, consider:

- SendGrid
- Amazon SES
- Mailgun

🎯 Phase 1 MVP Features

Completed:

- ✅ User authentication (JWT)
- ✅ Provider profile management
- ✅ Trip creation and management
- ✅ Trip search and filtering
- ✅ Booking inquiry system
- ✅ Email notifications
- ✅ Image uploads
- ✅ Responsive design

Future Phases:

- ❌ Real-time chat (Socket.IO)
- ❌ Payment processing (Stripe)
- ❌ Trip customization engine
- ❌ Review & rating system
- ❌ Amadeus API integration
- ❌ Mobile apps
- ❌ Advanced analytics

🧪 Testing Strategy

1. **Backend Testing**:
    - Unit tests for controllers
    - Integration tests for API endpoints
    - Database migration tests
2. **Frontend Testing**:
    - Component testing with React Testing Library
    - E2E testing with Cypress
    - Visual regression testing
3. **Performance Testing**:

    - Load testing with K6
    - Database query optimization
    - Frontend bundle analysis

📊 Monitoring & Analytics

For production deployment, implement:

- **Error Tracking**: Sentry
- **Analytics**: Google Analytics or Mixpanel
- **Performance**: New Relic or DataDog
- **Uptime**: Pingdom or UptimeRobot
- **Logs**: ELK Stack or CloudWatch

🔧 Development Workflow

1. **Local Development**:

   ```bash
   # Terminal 1 - Backend
   cd backend && npm run dev

   # Terminal 2 - Frontend
   cd frontend && npm start
   ```

2. **Database Changes**:
   - Modify `backend/src/config/database.js`
   - Run `npm run db:migrate`
   - Update seed data if needed
3. **API Changes**:
   - Update controller logic
   - Add validation rules
   - Update routes
   - Test with Postman/Insomnia
   - Update frontend services
4. **UI Changes**:
   - Create/modify components
   - Update pages
   - Test responsive design

- Check accessibility

📝 Code Standards

- **Backend**:
  - ESLint with Airbnb config
  - Async/await over callbacks
  - Proper error handling
  - RESTful API design
- **Frontend**:
  - React hooks only (no class components)
  - Functional components
  - PropTypes or TypeScript (future)
  - Tailwind for styling
- **Git**:
  - Feature branches
  - Conventional commits
  - PR reviews required
  - CI/CD must pass

🚦 Quick Commands

```bash
# Setup everything
chmod +x setup.sh && ./setup.sh

# Backend commands
cd backend
npm run dev              # Start development server
npm run db:migrate       # Run migrations
npm run db:seed         # Seed sample data
npm test                # Run tests

# Frontend commands
cd frontend
npm start               # Start development server
npm run build          # Production build
npm test               # Run tests

# Docker commands
docker-compose up -d    # Start all services
docker-compose down     # Stop all services
docker-compose logs -f  # View logs
```

This architecture provides a solid foundation for the AdventureConnect MVP while maintaining flexibility for future enhancements.