Adventure Log - Complete Project Files for Claude Code

Create these files in your project root folder so Claude Code has full context:

1. PROJECT_OVERVIEW.md

markdown

```
# Adventure Log - Project Overview

## Mission
Social travel logging platform that transforms personal journeys into beautiful, shareable stories through interactive alb

## Tech Stack
- **Frontend**: Next.js 14 (App Router) + TypeScript + React 18
- **Styling**: Tailwind CSS + shadcn/ui + Framer Motion
- **Backend**: Supabase (Database, Auth, Storage, Edge Functions)
- **Database**: PostgreSQL with PostGIS for geospatial data
- **Hosting**: Vercel (frontend) + Supabase (backend)
- **3D Globe**: Three.js + react-globe-gl for immersive 3D visualization
- **State**: TanStack Query + Zustand
- **Testing**: Vitest + Testing Library + Playwright

## Core Features
1. **Travel Documentation**: Albums with photos, location tagging, EXIF data
2. **Interactive 3D World Globe**: Immersive 3D globe showing visited places
3. **Social Features**: Follow, like, comment, activity feed
4. **Gamification**: Badges, challenges, achievements
5. **Discovery**: Search, explore, recommendations

## Current Phase
Phase 1: Foundation & Authentication - Building MVP in 72 hours

## Rapid Build Timeline
- Day 1 (8h): Authentication & Layout
- Day 2 (8-10h): Albums & Photos
- Day 3 (6-8h): 3D Globe & Deploy
```

2. ARCHITECTURE.md

markdown

```
# Adventure Log - System Architecture

## Folder Structure
```

```
src/
├── app/                    # Next.js App Router
│   ├── (auth)/             # Authentication pages (/login, /signup)
│   ├── (marketing)/        # Landing pages (/, /about)
│   ├── (app)/              # Main application (protected routes)
│   │   ├── dashboard/      # User dashboard with stats
│   │   ├── albums/         # Album management
│   │   │   ├── [id]/       # Individual album pages
│   │   │   └── new/        # Create new album
│   │   ├── globe/          # 3D globe view
│   │   ├── profile/        # User profile management
│   │   └── settings/       # User settings
│   ├── api/                # API routes
│   │   ├── auth/           # Authentication endpoints
│   │   ├── albums/         # Album CRUD operations
│   │   ├── photos/         # Photo upload and management
│   │   └── users/          # User profile operations
│   ├── globals.css         # Global styles
│   └── layout.tsx          # Root layout
├── components/             # Reusable components
│   ├── ui/                 # shadcn/ui components
│   ├── auth/               # Authentication components
│   ├── albums/             # Album-related components
│   ├── photos/             # Photo management components
│   ├── globe/              # 3D globe components
│   ├── forms/              # Form components
│   ├── layout/             # Layout components (nav, header, sidebar)
│   └── common/             # Common utility components
├── lib/                    # Utilities and configurations
│   ├── supabase/           # Supabase client and utilities
│   ├── validations/        # Zod schemas for validation
│   ├── utils/              # General utilities
│   ├── hooks/              # Custom React hooks
│   └── constants/          # App constants
├── stores/                 # Zustand stores for global state
├── types/                  # TypeScript type definitions
└── tests/                  # Test files (unit and e2e)
```

Key Architecture Patterns
- **Server Components**: Default for data fetching
- **Client Components**: Only when interactivity needed
- **React Query**: All server state management
- **Zustand**: Minimal global state (user preferences, UI state)
- **Zod**: Runtime validation for all user inputs
- **RLS**: Database-level security for all operations

Database Design
- **Core Tables**: profiles, albums, photos, countries, cities
- **Social**: follows, likes, comments, activities
- **Gamification**: badges, user_badges, challenges
- **Security**: Row-Level Security (RLS) on all tables
- **Storage**: Supabase Storage for photos with CDN

3. DATABASE_SCHEMA.md

markdown

Adventure Log - Database Schema

Complete Database Setup SQL

```sql
-- Enable necessary extensions
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS "postgis";

-- Profiles table (extends Supabase auth.users)
CREATE TABLE profiles (
  id UUID REFERENCES auth.users ON DELETE CASCADE PRIMARY KEY,
  username VARCHAR(50) UNIQUE NOT NULL,
  display_name VARCHAR(100),
  bio TEXT,
  avatar_url TEXT,
  website TEXT,
  location VARCHAR(100),
  privacy_level VARCHAR(20) DEFAULT 'public' CHECK (privacy_level IN ('private', 'friends', 'public')),
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Countries reference data
CREATE TABLE countries (
  id SERIAL PRIMARY KEY,
  code CHAR(2) UNIQUE NOT NULL,
  name VARCHAR(100) NOT NULL,
  latitude DECIMAL(10, 8),
  longitude DECIMAL(11, 8)
);

-- Cities reference data
CREATE TABLE cities (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  country_id INTEGER REFERENCES countries(id),
  latitude DECIMAL(10, 8),
  longitude DECIMAL(11, 8),
  population INTEGER
);

-- Albums table
CREATE TABLE albums (
  id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
  user_id UUID REFERENCES profiles(id) ON DELETE CASCADE NOT NULL,
  title VARCHAR(200) NOT NULL,
```

```
  description TEXT,
  cover_photo_url TEXT,
  start_date DATE,
  end_date DATE,
  visibility VARCHAR(20) DEFAULT 'public' CHECK (visibility IN ('private', 'friends', 'public')),
  tags TEXT[],
  location_name VARCHAR(200),
  country_id INTEGER REFERENCES countries(id),
  city_id INTEGER REFERENCES cities(id),
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Photos table
CREATE TABLE photos (
  id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
  album_id UUID REFERENCES albums(id) ON DELETE CASCADE NOT NULL,
  user_id UUID REFERENCES profiles(id) ON DELETE CASCADE NOT NULL,
  file_path TEXT NOT NULL,
  file_size INTEGER,
  width INTEGER,
  height INTEGER,
  caption TEXT,
  taken_at TIMESTAMPTZ,
  latitude DECIMAL(10, 8),
  longitude DECIMAL(11, 8),
  country VARCHAR(100),
  city VARCHAR(100),
  exif_data JSONB,
  processing_status VARCHAR(20) DEFAULT 'completed',
  order_index INTEGER DEFAULT 0,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Social follows table
CREATE TABLE follows (
  id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
  follower_id UUID REFERENCES profiles(id) ON DELETE CASCADE,
  following_id UUID REFERENCES profiles(id) ON DELETE CASCADE,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  UNIQUE(follower_id, following_id)
);

-- Likes table (for albums and photos)
CREATE TABLE likes (
  id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
  user_id UUID REFERENCES profiles(id) ON DELETE CASCADE,
```

```
  target_type VARCHAR(20) NOT NULL CHECK (target_type IN ('album', 'photo')),
  target_id UUID NOT NULL,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  UNIQUE(user_id, target_type, target_id)
);

-- Comments table
CREATE TABLE comments (
  id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
  user_id UUID REFERENCES profiles(id) ON DELETE CASCADE,
  target_type VARCHAR(20) NOT NULL CHECK (target_type IN ('album', 'photo')),
  target_id UUID NOT NULL,
  content TEXT NOT NULL,
  parent_id UUID REFERENCES comments(id),
  created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Activity feed table
CREATE TABLE activities (
  id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
  user_id UUID REFERENCES profiles(id) ON DELETE CASCADE,
  activity_type VARCHAR(50) NOT NULL,
  target_type VARCHAR(20),
  target_id UUID,
  metadata JSONB,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Enable Row Level Security
ALTER TABLE profiles ENABLE ROW LEVEL SECURITY;
ALTER TABLE albums ENABLE ROW LEVEL SECURITY;
ALTER TABLE photos ENABLE ROW LEVEL SECURITY;
ALTER TABLE follows ENABLE ROW LEVEL SECURITY;
ALTER TABLE likes ENABLE ROW LEVEL SECURITY;
ALTER TABLE comments ENABLE ROW LEVEL SECURITY;
ALTER TABLE activities ENABLE ROW LEVEL SECURITY;

-- Profiles RLS Policies
CREATE POLICY "Public profiles are viewable by everyone" ON profiles
  FOR SELECT USING (privacy_level = 'public');

CREATE POLICY "Users can view their own profile" ON profiles
  FOR SELECT USING (auth.uid() = id);

CREATE POLICY "Users can update their own profile" ON profiles
  FOR UPDATE USING (auth.uid() = id);
```

```
CREATE POLICY "Users can insert their own profile" ON profiles
  FOR INSERT WITH CHECK (auth.uid() = id);

-- Albums RLS Policies
CREATE POLICY "Public albums are viewable by everyone" ON albums
  FOR SELECT USING (visibility = 'public');

CREATE POLICY "Users can view their own albums" ON albums
  FOR SELECT USING (auth.uid() = user_id);

CREATE POLICY "Friends can view friends-only albums" ON albums
  FOR SELECT USING (
    visibility = 'friends' AND
    EXISTS (
      SELECT 1 FROM follows
      WHERE follower_id = auth.uid() AND following_id = user_id
    )
  );

CREATE POLICY "Users can manage their own albums" ON albums
  FOR ALL USING (auth.uid() = user_id);

-- Photos RLS Policies (inherit from album visibility)
CREATE POLICY "Photos follow album visibility" ON photos
  FOR SELECT USING (
    EXISTS (
      SELECT 1 FROM albums
      WHERE albums.id = photos.album_id
      AND (
        albums.visibility = 'public' OR
        albums.user_id = auth.uid() OR
        (albums.visibility = 'friends' AND
         EXISTS (SELECT 1 FROM follows WHERE follower_id = auth.uid() AND following_id = albums.user_id))
      )
    )
  );

CREATE POLICY "Users can manage their own photos" ON photos
  FOR ALL USING (auth.uid() = user_id);

-- Social features policies
CREATE POLICY "Users can manage their own follows" ON follows
  FOR ALL USING (auth.uid() = follower_id);

CREATE POLICY "Users can view public follows" ON follows
  FOR SELECT USING (true);
```

```
CREATE POLICY "Users can manage their own likes" ON likes
  FOR ALL USING (auth.uid() = user_id);

CREATE POLICY "Users can view all likes" ON likes
  FOR SELECT USING (true);

CREATE POLICY "Users can manage their own comments" ON comments
  FOR ALL USING (auth.uid() = user_id);

CREATE POLICY "Users can view comments on visible content" ON comments
  FOR SELECT USING (true);

CREATE POLICY "Users can view their own activities" ON activities
  FOR SELECT USING (auth.uid() = user_id);

CREATE POLICY "Users can insert their own activities" ON activities
  FOR INSERT WITH CHECK (auth.uid() = user_id);

-- Storage bucket for photos
INSERT INTO storage.buckets (id, name, public) VALUES ('photos', 'photos', true);

-- Storage policies
CREATE POLICY "Anyone can view photos" ON storage.objects
  FOR SELECT USING (bucket_id = 'photos');

CREATE POLICY "Authenticated users can upload photos" ON storage.objects
  FOR INSERT WITH CHECK (
    bucket_id = 'photos' AND auth.role() = 'authenticated'
  );

CREATE POLICY "Users can update their own photos" ON storage.objects
  FOR UPDATE USING (
    bucket_id = 'photos' AND auth.uid()::text = (storage.foldername(name))[1]
  );

CREATE POLICY "Users can delete their own photos" ON storage.objects
  FOR DELETE USING (
    bucket_id = 'photos' AND auth.uid()::text = (storage.foldername(name))[1]
  );

-- Indexes for performance
CREATE INDEX idx_albums_user_id ON albums(user_id);
CREATE INDEX idx_albums_visibility ON albums(visibility);
CREATE INDEX idx_photos_album_id ON photos(album_id);
CREATE INDEX idx_photos_user_id ON photos(user_id);
CREATE INDEX idx_photos_location ON photos(latitude, longitude);
CREATE INDEX idx_follows_follower ON follows(follower_id);
```

```
CREATE INDEX idx_follows_following ON follows(following_id);
CREATE INDEX idx_likes_target ON likes(target_type, target_id);
CREATE INDEX idx_activities_user_created ON activities(user_id, created_at DESC);

-- Insert some sample countries for the globe
INSERT INTO countries (code, name, latitude, longitude) VALUES
('US', 'United States', 39.8283, -98.5795),
('GB', 'United Kingdom', 55.3781, -3.4360),
('FR', 'France', 46.2276, 2.2137),
('DE', 'Germany', 51.1657, 10.4515),
('IT', 'Italy', 41.8719, 12.5674),
('ES', 'Spain', 40.4637, -3.7492),
('JP', 'Japan', 36.2048, 138.2529),
('AU', 'Australia', -25.2744, 133.7751),
('BR', 'Brazil', -14.2350, -51.9253),
('CA', 'Canada', 56.1304, -106.3468);
```

TypeScript Types

```typescript
```

```typescript
// types/database.ts
export interface Profile {
  id: string;
  username: string;
  display_name?: string;
  bio?: string;
  avatar_url?: string;
  website?: string;
  location?: string;
  privacy_level: 'private' | 'friends' | 'public';
  created_at: string;
  updated_at: string;
}

export interface Album {
  id: string;
  user_id: string;
  title: string;
  description?: string;
  cover_photo_url?: string;
  start_date?: string;
  end_date?: string;
  visibility: 'private' | 'friends' | 'public';
  tags?: string[];
  location_name?: string;
  country_id?: number;
  city_id?: number;
  created_at: string;
  updated_at: string;
  photos?: Photo[];
  user?: Profile;
}

export interface Photo {
  id: string;
  album_id: string;
  user_id: string;
  file_path: string;
  file_size?: number;
  width?: number;
  height?: number;
  caption?: string;
  taken_at?: string;
  latitude?: number;
  longitude?: number;
  country?: string;
```

```
  city?: string;
  exif_data?: any;
  processing_status: string;
  order_index: number;
  created_at: string;
}

export interface Country {
  id: number;
  code: string;
  name: string;
  latitude?: number;
  longitude?: number;
}
```

4. CODING_STANDARDS.md
```markdown
# Adventure Log - Coding Standards

## TypeScript Standards
- **Strict Mode**: Always enabled, no `any` types
- **Type Inference**: Prefer inference over explicit types when clear
- **Zod Validation**: All user inputs must be validated with Zod schemas
- **Interface over Type**: Use interfaces for object shapes

## React Patterns
- **Server Components**: Default choice for pages and data fetching
- **Client Components**: Only when interactivity needed ('use client')
- **Custom Hooks**: Extract reusable logic into hooks
- **Error Boundaries**: Wrap components that might fail
- **Suspense**: Use for loading states with fallbacks

## File Naming Conventions
- **Components**: PascalCase (e.g., `AlbumCard.tsx`)
- **Hooks**: camelCase starting with `use` (e.g., `useAlbums.ts`)
- **Utilities**: camelCase (e.g., `formatDate.ts`)
- **Pages**: lowercase with hyphens in URLs
- **Types**: PascalCase interfaces/types

## Component Structure
```typescript
// 1. Imports (grouped: React, libraries, internal)
import { useState } from 'react';
import { Button } from '@/components/ui/button';
import { useAlbums } from '@/lib/hooks/useAlbums';

// 2. Types and Interfaces
interface AlbumCardProps {
  album: Album;
  onEdit?: () => void;
}

// 3. Component Implementation
export function AlbumCard({ album, onEdit }: AlbumCardProps) {
  // 4. Hooks (state, queries, etc.)
  const [isLoading, setIsLoading] = useState(false);
  const { mutate: deleteAlbum } = useDeleteAlbum();

  // 5. Event Handlers
  const handleDelete = async () => {
```
```

```
    setIsLoading(true);
    try {
      await deleteAlbum(album.id);
    } finally {
      setIsLoading(false);
    }
  };

  // 6. JSX Return
  return (
    <div className="rounded-lg border p-4">
      {/* Component JSX */}
    </div>
  );
}
```

CSS and Styling

- **Tailwind CSS**: Primary styling method
- **shadcn/ui**: Use existing components when available
- **Custom CSS**: Only for complex animations or unavoidable cases
- **Mobile First**: All designs start mobile, scale up
- **Consistent Spacing**: Use Tailwind spacing scale (4, 8, 12, 16, etc.)

API and Data Patterns

- **React Query**: All server state management
- **Optimistic Updates**: For better UX on mutations
- **Error Handling**: Consistent error boundaries and toast notifications
- **Loading States**: Always provide loading feedback
- **Caching**: Strategic cache invalidation with React Query

Security Rules

- **Input Validation**: Validate everything with Zod
- **RLS First**: Database security at row level
- **No Client Secrets**: Never expose sensitive keys
- **File Upload**: Validate file types and sizes
- **XSS Prevention**: Sanitize user content

Performance Guidelines

- **Image Optimization**: Always use Next.js Image component
- **Code Splitting**: Dynamic imports for large components
- **Bundle Analysis**: Regular bundle size monitoring
- **Database Queries**: Efficient queries with proper indexes
- **Lazy Loading**: For images and non-critical components

Accessibility Standards

- **WCAG 2.2 AA**: Minimum compliance level
- **Semantic HTML**: Use proper HTML elements
- **Keyboard Navigation**: All interactive elements accessible
- **Screen Readers**: Proper ARIA labels and descriptions
- **Color Contrast**: Minimum 4.5:1 ratio for normal text

Git and Development

- **Conventional Commits**: Use standard commit message format
- **Feature Branches**: One feature per branch
- **Small Commits**: Atomic commits with clear messages
- **Code Review**: All code must be reviewed before merge
- **Testing**: Unit tests for logic, E2E for user flows

5. API_DESIGN.md
```markdown
# Adventure Log - API Design

## API Routes Structure

### Authentication
- `POST /api/auth/signup` - User registration
- `POST /api/auth/login` - User login
- `POST /api/auth/logout` - User logout
- `POST /api/auth/reset-password` - Password reset

### Users/Profiles
- `GET /api/users/me` - Current user profile
- `PUT /api/users/me` - Update current user profile
- `GET /api/users/[id]` - Get user profile by ID
- `POST /api/users/[id]/follow` - Follow/unfollow user
- `GET /api/users/[id]/followers` - Get user followers
- `GET /api/users/[id]/following` - Get users being followed

### Albums
- `GET /api/albums` - List albums (with filters)
- `POST /api/albums` - Create new album
- `GET /api/albums/[id]` - Get album details
- `PUT /api/albums/[id]` - Update album
- `DELETE /api/albums/[id]` - Delete album
- `POST /api/albums/[id]/photos` - Upload photos to album

### Photos
- `GET /api/photos/[id]` - Get photo details
- `PUT /api/photos/[id]` - Update photo (caption, location)
- `DELETE /api/photos/[id]` - Delete photo
- `POST /api/photos/[id]/like` - Like/unlike photo

### Social
- `POST /api/like` - Like album or photo
- `DELETE /api/like` - Remove like
- `POST /api/comments` - Add comment
- `GET /api/comments` - Get comments for target
- `DELETE /api/comments/[id]` - Delete comment
- `GET /api/feed` - Get activity feed

### Globe/Travel
- `GET /api/travel/stats` - User travel statistics
- `GET /api/travel/countries` - Countries visited by user
```

- `GET /api/countries` - List all countries

Request/Response Formats

Create Album
```typescript
// POST /api/albums
{
  title: string;
  description?: string;
  start_date?: string;
  end_date?: string;
  visibility: 'private' | 'friends' | 'public';
  location_name?: string;
  tags?: string[];
}

// Response
{
  id: string;
  user_id: string;
  title: string;
  // ... other fields
  created_at: string;
}
```

Upload Photos

typescript

```
// POST /api/albums/[id]/photos
// FormData with:
{
  files: File[];
  captions?: string[];
  coordinates?: { lat: number; lng: number }[];
}

// Response
{
  photos: Array<{
    id: string;
    file_path: string;
    caption?: string;
    latitude?: number;
    longitude?: number;
    // ... other fields
  }>;
}
```

Error Handling

- `400`: Bad Request (validation errors)
- `401`: Unauthorized (not logged in)
- `403`: Forbidden (insufficient permissions)
- `404`: Not Found
- `429`: Too Many Requests (rate limiting)
- `500`: Internal Server Error

Error Response Format

typescript

```
{
  error: {
    code: string;
    message: string;
    details?: any;
  };
}
```

Rate Limiting

- Authentication: 5 attempts per minute per IP

- File uploads: 10 per minute per user
- API calls: 100 per minute per user
- Comments: 20 per minute per user

````
## 6. DEVELOPMENT_WORKFLOW.md
```markdown
# Adventure Log - Development Workflow

## Quick Start Commands
```bash
# Development
npm run dev              # Start development server
npm run build            # Build for production
npm run start            # Start production server

# Code Quality
npm run lint             # ESLint check
npm run type-check       # TypeScript check
npm run format           # Prettier format

# Testing
npm run test             # Unit tests with Vitest
npm run test:watch       # Watch mode tests
npm run test:e2e         # Playwright E2E tests

# Database
npm run db:generate      # Generate TypeScript types
npm run db:reset         # Reset local database
npm run db:seed          # Seed with test data
````

Git Workflow

```bash
# Feature development
git checkout -b feature/album-creation
# ... make changes
git add .
git commit -m "feat: add album creation form"
git push origin feature/album-creation
# Create PR
```

Commit Message Format

- `feat:` New features
- `fix:` Bug fixes
- `docs:` Documentation updates
- `style:` Code style (formatting, semicolons)
- `refactor:` Code refactoring
- `test:` Adding tests
- `chore:` Maintenance tasks

Environment Setup

```bash
# Clone repository
git clone [repo-url]
cd adventure-log

# Install dependencies
npm install

# Environment variables
cp .env.example .env.local
# Add your Supabase credentials

# Database setup
# Run the SQL from DATABASE_SCHEMA.md in Supabase

# Start development
npm run dev
```

Testing Strategy

- **Unit Tests**: Business logic and utilities
- **Integration Tests**: API routes and database operations
- **E2E Tests**: Critical user journeys
- **Visual Tests**: Component rendering and responsive design

Code Review Checklist

- [ ] Follows coding standards
- [ ] Includes appropriate tests
- [ ] Performance considerations addressed

- [ ] Security best practices followed
- [ ] Accessibility requirements met
- [ ] Mobile responsive design
- [ ] Error handling implemented
- [ ] Loading states included

````
## 7. CLAUDE_CODE_PROMPTS.md
```markdown
# Claude Code Prompts for Adventure Log

## Base Context for All Prompts
````

Context: Adventure Log - Social travel platform
Tech Stack: Next.js 14, TypeScript, Supabase, Tailwind CSS, shadcn/ui, react-globe-gl
Architecture: See ARCHITECTURE.md, database schema in DATABASE_SCHEMA.md
Standards: Follow CODING_STANDARDS.md patterns
Goal: Build production-ready travel logging app with 3D globe visualization

Task: [specific request]

````
## Day 1: Authentication & Layout

### Authentication System
````

Build complete authentication system for Adventure Log.

Requirements:

- Email/password signup/login using Supabase Auth
- Profile creation flow after first login
- Password reset functionality
- Auth context provider with session management
- Protected route wrapper component
- Login/signup forms with validation using react-hook-form + Zod
- Error handling and loading states
- Responsive design with shadcn/ui components

Files to create:

- app/(auth)/login/page.tsx
````

- app/(auth)/signup/page.tsx
- components/auth/AuthProvider.tsx
- components/auth/ProtectedRoute.tsx
- lib/hooks/useAuth.ts
- lib/validations/auth.ts

Follow security best practices and include proper TypeScript types.

```
### App Layout & Navigation
```

Create main application layout and navigation for Adventure Log.

Requirements:

- Responsive layout with sidebar navigation
- Mobile hamburger menu with smooth animations
- Header with user avatar dropdown
- Navigation items: Dashboard, Albums, Globe, Profile, Settings
- User avatar with dropdown (profile, settings, logout)
- Loading states and error boundaries
- Consistent styling with shadcn/ui
- Mobile-first responsive design
- Accessibility compliance (WCAG 2.2 AA)

Files to create:

- app/(app)/layout.tsx
- components/layout/AppHeader.tsx
- components/layout/Sidebar.tsx
- components/layout/MobileNav.tsx
- components/ui/UserAvatar.tsx

Use Framer Motion for smooth animations.

```
### Profile Management
```

Build user profile management system.

Requirements:

- Profile creation form (username, display_name, bio, location)
- Avatar upload to Supabase Storage with image compression
- Profile viewing page with travel stats
- Edit profile functionality
- Form validation with Zod schemas
- Image cropping and optimization
- Real-time updates with React Query
- Error handling and success feedback

Files to create:

- app/(app)/profile/page.tsx
- app/(app)/profile/edit/page.tsx
- components/profile/ProfileForm.tsx
- components/profile/AvatarUpload.tsx
- lib/hooks/useProfile.ts
- lib/validations/profile.ts
- lib/utils/imageUtils.ts

Include proper file upload security and validation.

```
## Day 2: Albums & Photos

### Album Management System
```

Build complete album management system for Adventure Log.

Requirements:

- Album creation form with title, description, date range, visibility
- Album listing page with grid layout and infinite scroll
- Individual album view with photo gallery
- Edit album functionality
- Delete album with confirmation
- Album visibility settings (private, friends, public)
- Tags system for albums

- Location association with albums
- Real-time updates using React Query
- Optimistic updates for better UX

Files to create:

- app/(app)/albums/page.tsx
- app/(app)/albums/new/page.tsx
- app/(app)/albums/[id]/page.tsx
- app/(app)/albums/[id]/edit/page.tsx
- components/albums/AlbumCard.tsx
- components/albums/AlbumForm.tsx
- components/albums/AlbumGallery.tsx
- lib/hooks/useAlbums.ts
- lib/validations/album.ts

Include pagination and search functionality.

```
### Photo Upload & Management
```

Create advanced photo management system for albums.

Requirements:

- Drag & drop photo upload with progress indicators
- Multiple file selection and batch upload
- Photo reordering within albums
- Photo caption editing
- Full-screen photo viewer with navigation
- EXIF data extraction (GPS, camera info, date taken)
- Image optimization (multiple sizes: thumbnail, medium, large)
- Lazy loading for performance
- Photo deletion with confirmation
- Bulk photo operations

Files to create:

- components/photos/PhotoUpload.tsx

- components/photos/PhotoGallery.tsx
- components/photos/PhotoViewer.tsx
- components/photos/PhotoCard.tsx
- lib/hooks/usePhotos.ts
- lib/utils/exifUtils.ts
- lib/utils/imageProcessing.ts
- api/photos/upload/route.ts

Use Supabase Storage with proper security policies.

```
### Location & EXIF Features
```

Add location tagging and EXIF processing to photos.

Requirements:

- Extract GPS coordinates from photo EXIF data
- Reverse geocoding (coordinates to city/country names)
- Manual location picker with map interface
- Location search with autocomplete
- Display location info on photos and albums
- Country/city aggregation for travel stats
- Privacy controls for location sharing
- Geolocation accuracy indicators

Files to create:

- components/location/LocationPicker.tsx
- components/location/LocationDisplay.tsx
- lib/hooks/useGeocoding.ts
- lib/utils/locationUtils.ts
- api/geocoding/route.ts

Integration with geocoding service (Mapbox or Google Maps).

Day 3: 3D Globe & Polish

3D Globe Implementation

Create immersive 3D globe component for Adventure Log travel visualization.

Requirements:

- Interactive 3D globe using react-globe-gl and Three.js
- Display visited countries from user's photo/album data
- Country highlighting with different colors based on visit frequency
- Smooth camera animations and transitions
- Click countries to view related albums
- Touch gestures for mobile (pinch, rotate, pan)
- Mouse controls for desktop (drag, zoom, click)
- Atmospheric effects and realistic lighting
- Performance optimization for mobile devices
- Loading states and error handling
- WebGL capability detection with 2D fallback

Files to create:

- app/(app)/globe/page.tsx
- components/globe/Globe3D.tsx
- components/globe/CountryTooltip.tsx
- components/globe/GlobeControls.tsx
- lib/hooks/useGlobeData.ts
- lib/utils/globeUtils.ts

Focus on smooth 60fps performance and mobile optimization.

Travel Statistics & Dashboard

Build travel statistics dashboard and user dashboard.

Requirements:

- Personal dashboard with key stats and recent albums

- Travel statistics: countries visited, cities explored, photos uploaded
- Interactive charts showing travel timeline
- Recent activity feed
- Quick actions (new album, upload photos)
- Travel goals and progress tracking
- Beautiful data visualizations
- Mobile-responsive layout

Files to create:

- app/(app)/dashboard/page.tsx
- components/dashboard/TravelStats.tsx
- components/dashboard/RecentActivity.tsx
- components/dashboard/QuickActions.tsx
- components/charts/TravelChart.tsx
- lib/hooks/useTravelStats.ts

Use recharts for data visualization.

```
### Final Polish & Deployment
```

Polish Adventure Log for production deployment.

Requirements:

- Responsive design review across all pages
- Loading states and error boundaries everywhere
- Image optimization and lazy loading
- Performance optimization (bundle analysis, Core Web Vitals)
- SEO optimization with proper meta tags
- Error tracking setup with Sentry
- Analytics integration
- Security audit and validation
- Accessibility testing and improvements
- Cross-browser testing
- Deploy to Vercel with proper environment variables

Tasks:

- Performance audit and optimization
- Security review and hardening
- Accessibility compliance check
- Mobile experience testing
- Production deployment setup
- Monitoring and analytics integration

Ensure production-ready quality and performance.

```
## Debugging Prompts

### Authentication Issues
```

Debug authentication problem in Adventure Log.

Issue: [describe specific problem]
Context: Using Supabase Auth with Next.js App Router
Check: RLS policies, middleware setup, auth helpers configuration
Requirements: Maintain security while fixing issue

Analyze the auth flow and provide solution with explanation.

```
### Performance Issues
```

Optimize performance for Adventure Log [specific area].

Current issues: [describe performance problems]
Target: Achieve Core Web Vitals thresholds
Context: Next.js 14 with React Query, 3D globe, image galleries
Requirements: Maintain functionality while improving performance

Provide specific optimizations with before/after metrics.

```
### Database Issues
```

Debug database/RLS issue in Adventure Log.

Problem: [describe database problem]
Context: PostgreSQL with RLS policies, Supabase Check: Row-level security policies, foreign key constraints Requirements: Maintain data security and integrity

Analyze schema and policies, provide fix with explanation.

8. CURRENT_PHASE.md

markdown

Adventure Log - Current Development Phase

Current Status: Ready to Start Development

Project Setup Status
- [ ] Next.js project initialized
- [ ] Dependencies installed
- [ ] Supabase project created
- [ ] Database schema implemented
- [ ] Environment variables configured
- [ ] shadcn/ui components installed

Development Phase 1: Foundation (Day 1 - 8 hours)
Status: NOT STARTED

Authentication System (Hours 0-3)
- [ ] Supabase Auth integration
- [ ] Login/signup pages
- [ ] Auth context provider
- [ ] Protected routes
- [ ] Profile creation flow

App Layout & Navigation (Hours 3-6)
- [ ] Main layout component
- [ ] Responsive navigation
- [ ] Mobile hamburger menu
- [ ] User avatar dropdown
- [ ] Error boundaries

Profile Management (Hours 6-8)
- [ ] Profile form
- [ ] Avatar upload
- [ ] Profile viewing page
- [ ] Edit functionality

Development Phase 2: Core Features (Day 2 - 8-10 hours)
Status: NOT STARTED

Album System (Hours 8-12)
- [ ] Album creation
- [ ] Album listing
- [ ] Album editing
- [ ] Visibility controls

Photo Management (Hours 12-16)
- [ ] Photo upload

- [ ] Drag & drop interface
- [ ] Photo gallery
- [ ] EXIF extraction

Location Features (Hours 16-18)
- [ ] GPS extraction
- [ ] Manual tagging
- [ ] Geocoding
- [ ] Location display

Development Phase 3: 3D Globe & Polish (Day 3 - 6-8 hours)
Status: NOT STARTED

3D Globe (Hours 18-22)
- [ ] Globe component
- [ ] Country visualization
- [ ] Interactions
- [ ] Mobile optimization

Final Polish (Hours 22-26)
- [ ] Dashboard
- [ ] Performance optimization
- [ ] Responsive design
- [ ] Production deployment

Current Priority
1. **Project Setup**: Initialize Next.js project and install dependencies
2. **Database Setup**: Create Supabase project and run schema SQL
3. **Environment**: Configure environment variables
4. **Start Day 1**: Begin with authentication system

Blockers
None currently - ready to begin development

Next Steps
1. Run project initialization commands
2. Create Supabase project and configure database
3. Start with authentication system using Claude Code
4. Follow 72-hour rapid build plan

Notes
- Focus on MVP functionality first
- 3D globe is key differentiator
- Mobile-first approach essential
- Target 72-hour timeline for working beta

9. RAPID_BUILD_CHECKLIST.md

markdown

Adventure Log - 72-Hour Rapid Build Checklist

✅ Pre-Development Setup (2 hours)

Project Initialization
- [ ] `npx create-next-app@latest adventure-log --typescript --tailwind --eslint --app --src-dir`
- [ ] Install core dependencies (Supabase, React Query, Zod, etc.)
- [ ] Setup shadcn/ui with essential components
- [ ] Create folder structure as per ARCHITECTURE.md

Supabase Setup
- [ ] Create new Supabase project
- [ ] Copy project URL and anon key
- [ ] Create `.env.local` with environment variables
- [ ] Run complete database schema from DATABASE_SCHEMA.md
- [ ] Test database connection

Documentation
- [ ] Add all project files (this checklist included) to project root
- [ ] Verify Claude Code can access all documentation
- [ ] Test first Claude Code command

🔥 Day 1: Foundation (8 hours)

Authentication (3 hours)
- [ ] Supabase Auth helpers setup
- [ ] Login page with form validation
- [ ] Signup page with profile creation
- [ ] Auth context provider
- [ ] Protected route wrapper
- [ ] Middleware for auth checking
- [ ] Password reset functionality
- [ ] Error handling and loading states

Layout & Navigation (3 hours)
- [ ] Main app layout component
- [ ] Responsive sidebar navigation
- [ ] Mobile hamburger menu
- [ ] Header with user avatar
- [ ] User dropdown (profile, settings, logout)
- [ ] Navigation items (Dashboard, Albums, Globe, Profile)
- [ ] Loading states and error boundaries
- [ ] Mobile-responsive design

Profile Management (2 hours)
- [ ] Profile creation form

- [ ] Avatar upload to Supabase Storage
- [ ] Profile viewing page
- [ ] Edit profile functionality
- [ ] Form validation with Zod
- [ ] Success/error feedback

Day 1 Success Criteria:
- [ ] User can signup/login successfully
- [ ] Navigation works on desktop and mobile
- [ ] Profile creation and editing functional
- [ ] App feels responsive and polished

🚀 Day 2: Albums & Photos (8-10 hours)

Album System (4 hours)
- [ ] Album creation form
- [ ] Album listing with grid layout
- [ ] Individual album pages
- [ ] Album editing functionality
- [ ] Delete album with confirmation
- [ ] Visibility settings (private/friends/public)
- [ ] Album tags system
- [ ] Location association
- [ ] Pagination for large lists

Photo Management (4 hours)
- [ ] Drag & drop photo upload
- [ ] Multiple file selection
- [ ] Upload progress indicators
- [ ] Photo gallery component
- [ ] Full-screen photo viewer
- [ ] Photo reordering
- [ ] Caption editing
- [ ] Photo deletion
- [ ] Image optimization (multiple sizes)

EXIF & Location (2 hours)
- [ ] EXIF data extraction
- [ ] GPS coordinates from photos
- [ ] Reverse geocoding setup
- [ ] Manual location tagging
- [ ] Location display on photos
- [ ] Country/city aggregation
- [ ] Privacy controls for location

Day 2 Success Criteria:
- [ ] User can create albums and upload photos

- [ ] Photo gallery works smoothly
- [ ] Location data is captured and displayed
- [ ] Mobile photo upload experience is good

🌍 Day 3: 3D Globe & Deploy (6-8 hours)

3D Globe Implementation (4 hours)
- [ ] react-globe-gl setup
- [ ] 3D globe component
- [ ] Country data visualization
- [ ] Visited countries highlighting
- [ ] Click countries to view albums
- [ ] Smooth camera animations
- [ ] Touch gestures for mobile
- [ ] Mouse controls for desktop
- [ ] Performance optimization
- [ ] Loading states

Travel Statistics (2 hours)
- [ ] Dashboard with travel stats
- [ ] Countries visited counter
- [ ] Cities explored tracker
- [ ] Photos uploaded stats
- [ ] Recent albums display
- [ ] Quick action buttons
- [ ] Travel timeline visualization

Polish & Deploy (2 hours)
- [ ] Responsive design review
- [ ] Error handling everywhere
- [ ] Loading states polished
- [ ] Performance optimization
- [ ] SEO meta tags
- [ ] Production build testing
- [ ] Deploy to Vercel
- [ ] Environment variables setup
- [ ] SSL and custom domain

Day 3 Success Criteria:
- [ ] 3D globe shows user's travel data
- [ ] Globe interactions work on mobile/desktop
- [ ] App is deployed and publicly accessible
- [ ] Performance is acceptable on mobile

🎯 Final Validation (30 minutes)

End-to-End Testing

- [ ] Complete user journey: signup → create album → upload photos → view globe
- [ ] Test on multiple devices (desktop, tablet, mobile)
- [ ] Test in different browsers
- [ ] Check performance (loading times, 3D globe FPS)
- [ ] Verify all links and functionality work

Launch Preparation
- [ ] Share with 3-5 friends for immediate feedback
- [ ] Monitor Vercel analytics for errors
- [ ] Check Supabase logs for issues
- [ ] Create feedback collection method
- [ ] Document known issues for future fixes

🚨 Emergency Shortcuts

If behind schedule, prioritize in this order:

Must Have (Don't skip)
1. Authentication and user accounts
2. Album creation and photo upload
3. Basic photo gallery
4. 3D globe with visited countries
5. Mobile responsive design

Nice to Have (Can skip for v1)
1. Advanced photo editing features
2. Sophisticated location search
3. Complex animations
4. Advanced error handling
5. Performance optimizations

Quick Wins
1. Use shadcn/ui components instead of custom styling
2. Skip complex photo reordering for v1
3. Use simple country highlighting instead of complex globe effects
4. Basic form validation instead of advanced UX
5. Deploy early and iterate

📊 Success Metrics

Technical
- [ ] App loads in <3 seconds on mobile
- [ ] 3D globe maintains >30 FPS on mid-range devices
- [ ] Photo uploads complete successfully >95% of time
- [ ] No critical bugs in core user flow

User Experience

- [ ] Complete signup → first album → globe view in <10 minutes
- [ ] Intuitive navigation without explanation
- [ ] Mobile experience feels native
- [ ] Globe interaction is engaging and smooth

Business
- [ ] 5+ friends willing to use the app regularly
- [ ] Positive feedback on core value proposition
- [ ] Users upload multiple albums organically
- [ ] Globe view creates "wow" moment

This checklist ensures you stay on track for the 72-hour build while maintaining quality and core functionality.

10. .env.example

env

```
# Supabase Configuration
NEXT_PUBLIC_SUPABASE_URL=your_supabase_project_url
NEXT_PUBLIC_SUPABASE_ANON_KEY=your_supabase_anon_key

# Optional: For enhanced features
NEXT_PUBLIC_MAPBOX_ACCESS_TOKEN=your_mapbox_token_for_geocoding
NEXT_PUBLIC_APP_URL=http://localhost:3000

# Development
NODE_ENV=development
```

📁 File Creation Instructions

1. **Create these 10 files** in your project root directory
2. **Copy the content** from each section above into the corresponding file
3. **Customize** PROJECT_OVERVIEW.md and CURRENT_PHASE.md based on your progress
4. **Update** CURRENT_PHASE.md as you complete tasks

🚀 Using with Claude Code

Once these files are in your project:

bash

```
# Start development with context
claude-code "Following the 72-hour rapid build plan in RAPID_BUILD_CHECKLIST.md, implement the authentication sys

# Reference specific documentation
claude-code "Using the database schema in DATABASE_SCHEMA.md and following CODING_STANDARDS.md, build th

# Get help with debugging
claude-code "Debug this issue using the patterns in API_DESIGN.md and security policies in DATABASE_SCHEMA.md"
```

These files give Claude Code complete context about your project, ensuring consistent, high-quality code that follows your architecture and standards.