SmartTrip Project Structure

Complete guide to the project's file and directory structure, including explanations of each component's purpose and role in the application.

Project Tree Structure

```
trip-recommendations/
│
├── Root Configuration Files
│   ├── .gitignore                       # Git ignore rules for the entire project
│   ├── README.md                        # Main project documentation and setup guide
│   ├── render.yaml                      # Render.com deployment configuration
│   ├── vercel.json                      # Vercel deployment configuration
│   └── Procfile                         # Process configuration for deployment
│
├── frontend/                            # Next.js Frontend Application
│   ├── Configuration Files
│   │   ├── package.json                 # Frontend dependencies and npm scripts
│   │   ├── package-lock.json            # Locked dependency versions
│   │   ├── tsconfig.json                # TypeScript compiler configuration
│   │   ├── tailwind.config.ts           # Tailwind CSS utility configuration
│   │   ├── next.config.js               # Next.js framework configuration
│   │   ├── eslint.config.mjs            # ESLint linting rules
│   │   └── postcss.config.mjs           # PostCSS processing configuration
│   │
│   ├── Documentation Files
│   │   ├── FRONTEND_OVERVIEW.md         # Frontend architecture overview
│   │   └── CHECK_ENV.md                 # Environment variable checking guide
│   │
│   ├── public/                          # Static assets served directly
│   │   ├── images/                      # Image assets
│   │   │   ├── continents/              # Continent map images (7 images)
│   │   │   ├── logo/                    # Brand logo files
│   │   │   └── trip status/             # Trip status indicator icons (4 icons)
│   │   └── *.svg                        # SVG icon files (file.svg, globe.svg, etc.)
│   │
│   └── src/                             # Source code directory
│       ├── app/                         # Next.js App Router pages
│       │   ├── layout.tsx               # Root layout component with providers
│       │   ├── page.tsx                 # Home page component
│       │   ├── error.tsx                # Global error boundary
│       │   ├── not-found.tsx            # 404 page component
│       │   ├── globals.css              # Global CSS styles and Tailwind directives
│       │   ├── favicon.ico              # Site favicon
│       │   │
│       │   ├── auth/                    # Authentication pages
│       │   │   ├── page.tsx             # Login/signup page with Supabase OAuth
│       │   │   └── callback/            # OAuth callback handler
│       │   │       └── page.tsx         # Supabase auth callback processor
│       │   │
│       │   ├── search/                     # Search functionality
│       │   │   ├── page.tsx                # Search form page with filters
│       │   │   ├── error.tsx               # Search error boundary
│       │   │   └── results/                # Search results pages
│       │   │       ├── page.tsx            # Results display page with trip cards
│       │   │       ├── loading.tsx         # Loading state component
│       │   │       └── error.tsx           # Results error boundary
│       │   │
│       │   └── trip/                       # Trip detail pages
│       │       └── [id]/                   # Dynamic route for trip ID
│       │           └── page.tsx             # Individual trip detail page
│       │
│       ├── components/                     # Reusable React components
│       │   ├── features/                   # Feature-specific components
│       │   │   └── README.md               # Component documentation
│       │   └── ui/                         # UI primitives
│       │       └── README.md               # UI component docs
│       │
│       ├── hooks/                          # Custom React hooks
│       │   └── useTracking.ts              # User event tracking hooks (usePageView,
useResultsTracking, etc.)
│       │
│       ├── lib/                            # Library and utility code
│       │   ├── dataStore.tsx               # Global state management (Context API) for
reference data
│       │   └── supabaseClient.ts          # Supabase client initialization and auth utilities
│       │
│       └── services/                       # Service layer for API calls
│           ├── api.service.ts              # Backend API client service with retry logic and
auth
│           └── tracking.service.ts          # Event tracking service with batching and
session management
│
├── backend/                                # Flask Backend Application
│   ├── Configuration Files
│   │   ├── .gitignore                      # Backend-specific git ignore rules
│   │   ├── Procfile                        # Render.com process configuration
│   │   ├── requirements.txt                # Python production dependencies
│   │   ├── requirements-dev.txt           # Development dependencies
│   │   └── runtime.txt                     # Python version specification
│   │
│   ├── app/                                # Main application package
│   │   ├── __init__.py                     # Package initialization
│   │   ├── main.py                         # Flask application entry point and blueprint
registration
│   │   │
│   │   ├── core/                           # Core application components
│   │   │   ├── __init__.py
│   │   │   ├── config.py                   # Application configuration management
│   │   │   ├── database.py                 # Database connection management and session
handling
```

```
│   │   │   └── auth.py                    # Authentication utilities for Supabase JWT
validation
│   │   │
│   │   ├── models/                        # SQLAlchemy database models
│   │   │   ├── __init__.py
│   │   │   ├── trip.py                    # Trip-related models (V2 schema: TripTemplate,
TripOccurrence, etc.)
│   │   │   └── events.py                  # Event tracking models (User, Session, Event,
TripInteraction)
│   │   │
│   │   ├── services/                      # Business logic services
│   │   │   ├── __init__.py
│   │   │   ├── recommendation.py         # Recommendation algorithm service with scoring
logic
│   │   │   └── events.py                  # Event tracking service with user resolution and
validation
│   │   │
│   │   └── api/                           # API route handlers (Flask Blueprints)
│   │       ├── __init__.py
│   │       │
│   │       ├── v2/                        # API version 2 (current, uses V2 schema)
│   │       │   ├── __init__.py
│   │       │   └── routes.py             # V2 API endpoints (trips, templates, occurrences,
recommendations)
│   │       │
│   │       ├── events/                    # Event tracking endpoints
│   │       │   ├── __init__.py
│   │       │   └── routes.py              # Event API routes (track events, batch events,
session start)
│   │       │
│   │       ├── analytics/                  # Analytics and metrics endpoints
│   │       │   ├── __init__.py
│   │       │   └── routes.py              # Metrics API routes (current metrics, daily
metrics, top searches, evaluation)
│   │       │
│   │       ├── resources/                 # Resource data endpoints
│   │       │   ├── __init__.py
│   │       │   └── routes.py              # Read-only endpoints (countries, guides, trip
types, tags, locations)
│   │       │
│   │       └── system/                    # System endpoints
│   │           ├── __init__.py
│   │           └── routes.py              # Health check endpoint with database status
│   │
│   ├── recommender/                       # Recommendation engine module
│   │   ├── __init__.py
│   │   ├── logging.py                    # Request logging functionality for recommendation
API calls
│   │   ├── metrics.py                    # Performance metrics aggregation (on-demand
computation)
│   │   ├── evaluation.py                 # Algorithm quality evaluation using test scenarios
│   │   └── README.md                     # Recommender module documentation
│   │
│   ├── migrations/                       # Database migration scripts
│   │   ├── __init__.py
│   │   ├── _001_add_recommendation_logging.py
│   │   ├── _002_add_user_tracking.py
│   │   ├── _003_add_companies.py
│   │   ├── _004_refactor_trips_to_templates.py
│   │   ├── _005_normalize_events_and_load_scenarios.py
│   │   ├── _006_add_properties_jsonb.py
│   │   └── run_schema_v2_migration.py     # V2 schema migration runner
│   │
│   ├── scenarios/                        # Test scenarios and personas
│   │   ├── generated_personas.json        # Generated test user personas
│   │   └── README.md                      # Scenarios documentation
│   │
│   └── scripts/                          # Utility and maintenance scripts
│       ├── __init__.py
│       ├── README.md                      # Scripts documentation
│       ├── verify_restructure.py          # Verification script for project restructure
│       │
│       ├── _archive/                      # Archived scripts (no longer in use)
│       │   ├── analyze_scoring.py
│       │   └── check_v2_tables.py
│       │
│       ├── db/                            # Database management scripts
│       │   ├── __init__.py
│       │   ├── seed.py                    # Database seeding script
│       │   ├── check_schema.py             # Schema validation
│       │   ├── check_db_state.py          # Database state inspection
│       │   ├── verify_schema.py            # Schema verification
│       │   ├── verify_seed.py             # Seed data verification
│       │   ├── check_types.py             # Type checking utilities
│       │   ├── verify_names.py            # Name validation
│       │   └── check_guaranteed_trips.py   # Guaranteed trips checker
│       │
│       ├── data_gen/                      # Test data generation
│       │   ├── __init__.py
│       │   ├── generate_personas.py       # Generate test personas
│       │   ├── generate_trips.py          # Generate trip test data
│       │   └── create_realistic_data.py   # Create realistic test datasets
│       │
│       └── analytics/                     # Analytics and reporting scripts
│           ├── __init__.py
│           ├── aggregate_daily_metrics.py # Daily metrics aggregation (manual script)
│           ├── aggregate_trip_interactions.py # Trip interaction metrics
│           ├── analyze_scoring_v2.py     # V2 scoring analysis
│           ├── export_data.py            # Data export utilities
│           └── cleanup_sessions.py       # Session cleanup job
│
└── docs/                                  # Project documentation
    ├── README.md                          # Documentation index and overview
    │
    ├── architecture/                       # Architecture documentation
    │   ├── README.md                      # Architecture overview
    │   ├── PROJECT_STRUCTURE.md            # This file
    │   ├── DATABASE_SCHEMA_FULL.md        # Complete database schema documentation
    │   ├── DESIGN_PATTERNS.md             # Design patterns used in the project
    │   └── DESIGN_PATTERNS_README.md      # Design patterns overview
    │
    ├── deployment/                         # Deployment guides
    │   ├── DEPLOYMENT.md                  # General deployment guide
    │   ├── RENDER_STEPS.md                # Render.com deployment steps
    │   ├── PRODUCTION_ENV_SETUP.md        # Production environment setup
    │   ├── PRODUCTION_ENV_VARIABLES.md    # Production environment variables
    │   ├── POST_RESTRUCTURE_DEPLOYMENT_CHECKLIST.md
    │   └── CRON_SCHEDULING_GUIDE.md       # Cron job scheduling guide
    │
    ├── technical/                          # Technical documentation
    │   ├── README.md                      # Technical docs overview
    │   ├── README_DATABASE.md             # Database connection and setup
    │   ├── AUTHENTICATION_FLOW.md         # Authentication flow documentation
    │   ├── AUTHENTICATION_IMPLEMENTATION.md
    │   ├── COLD_START_FEATURE.md          # Cold start handling
    │   └── [various troubleshooting guides]
    │
    ├── testing/                            # Testing documentation
    │   └── MASTER_TEST_PLAN.md            # Comprehensive test plan
    │
    ├── proposals/                          # Feature proposals
    │   └── POPULARITY_SCORE_PROPOSAL.md   # Popularity score feature proposal
    │
    ├── archive/                            # Archived documentation
    │   └── [historical documentation and migration plans]
    │
    ├── DATA_PIPELINES.md                   # Data pipelines documentation
    ├── API_STRUCTURE.md                    # API structure and endpoints
    ├── LOCAL_DEVELOPMENT_SETUP.md         # Local development setup guide
    ├── SUPABASE_OAUTH_SETUP.md            # Supabase OAuth configuration
    └── [various other documentation files]
```

File and Directory Explanations

Root Level Files

`.gitignore`

Defines files and directories that Git should ignore. Includes patterns for Node.js (`node_modules`), Python (`__pycache__` , `venv/`), environment files (`.env*`), build artifacts, and IDE-specific files. Prevents committing sensitive data, dependencies, and generated files to version control.

`README.md`

Main project documentation providing an overview, architecture diagram, tech stack details, installation instructions, API reference, and deployment guides. Serves as the entry point for developers and stakeholders to understand the project.

`render.yaml`

Render.com deployment configuration file. Defines the backend web service, build commands, start commands, and environment variables. Enables automated deployment to Render's cloud platform with proper configuration.

`vercel.json`

Vercel deployment configuration for the frontend. Defines build settings, environment variables, and routing rules for Next.js application deployment.

`Procfile`

Process configuration file specifying the command to start the application. Used by deployment platforms like Render and Heroku.

Frontend Directory (`frontend/`)

Configuration Files

`package.json` Defines frontend project metadata, dependencies, and npm scripts. Lists all Node.js packages required (Next.js, React, TypeScript, Tailwind CSS, Supabase client, etc.) and provides commands for development (`dev`), building (`build`), and production (`start`).

`package-lock.json` Locks exact versions of all dependencies and their sub-dependencies. Ensures consistent installations across different environments and prevents version conflicts.

`tsconfig.json` TypeScript compiler configuration specifying compilation options, module resolution, target JavaScript version, and path aliases. Enables type checking and modern TypeScript features.

`tailwind.config.ts` Tailwind CSS configuration defining custom colors, spacing, fonts, and breakpoints. Extends default Tailwind utilities with project-specific design tokens.

`next.config.js` Next.js framework configuration for image optimization, environment variables, redirects, and build settings. Configures how Next.js processes and serves the application.

`eslint.config.mjs` ESLint linting rules and configuration. Enforces code quality standards, catches common errors, and maintains consistent code style across the frontend codebase.

`postcss.config.mjs` PostCSS configuration for CSS processing. Configures Tailwind CSS plugin and autoprefixer for vendor prefixing.

Documentation Files

`FRONTEND_OVERVIEW.md` Frontend architecture overview documenting component structure, state management, routing, and key design decisions.

`CHECK_ENV.md` Guide for checking and verifying environment variables required for frontend operation.

Public Assets (`public/`)

`public/images/` Static image assets served directly by Next.js without processing. Contains continent maps (7 PNG images), logo files, and trip status icons (4 SVG/PNG files) used throughout the UI.

`public/*.svg` SVG icon files for various UI elements. SVGs are scalable and lightweight, ideal for icons and simple graphics.

Source Code (`src/`)

`src/app/layout.tsx` Root layout component wrapping all pages. Sets up HTML structure, metadata, fonts, and global providers (like DataStoreProvider). Defines the base structure for the entire application.

`src/app/page.tsx` Home page component displaying the landing page with hero section, features, and call-to-action. First page users see when visiting the application.

`src/app/error.tsx` Global error boundary component catching unhandled errors in the application. Displays user-friendly error messages and provides recovery options.

`src/app/not-found.tsx` 404 page component shown when users navigate to non-existent routes. Provides navigation options back to valid pages.

`src/app/globals.css` Global CSS styles including Tailwind directives, custom CSS variables, and base styles applied across the entire application.

`src/app/favicon.ico` Site favicon displayed in browser tabs and bookmarks. Brand identifier for the application.

`src/app/auth/page.tsx` Authentication page handling user login and signup. Integrates with Supabase for OAuth and email/password authentication.

`src/app/auth/callback/page.tsx` OAuth callback handler processing authentication redirects from Supabase. Exchanges authorization codes for session tokens, links users to tracking system, and redirects users appropriately.

`src/app/search/page.tsx` Search form page allowing users to input preferences (locations, trip types, themes, budget, duration, difficulty). Collects user preferences, tracks search submission events, and navigates to results page with URL parameters.

`src/app/search/error.tsx` Error boundary specific to search functionality. Catches errors in search-related components and displays appropriate error messages.

`src/app/search/results/page.tsx` Search results display page showing recommended trips based on user preferences. Renders trip cards with match scores, tracks impressions and clicks, and provides navigation to trip detail pages.

`src/app/search/results/loading.tsx` Loading state component displayed while search results are being fetched. Provides visual feedback during API requests.

`src/app/search/results/error.tsx` Error boundary for search results page. Handles errors specific to results display and provides recovery options.

`src/app/trip/[id]/page.tsx` Dynamic trip detail page displaying comprehensive information about a specific trip. Uses Next.js dynamic routing to handle trip IDs, fetches detailed trip data from the API, and tracks trip view events.

`src/components/` Directory for reusable React components. Organized into `features/` for feature-specific components and `ui/` for primitive UI components. Currently contains README files documenting component organization.

`src/hooks/useTracking.ts` Custom React hooks for user event tracking. Provides hooks like `usePageView()` , `useResultsTracking()` , `useImpressionTracking()` , and `useFilterTracking()` that encapsulate tracking logic and integrate with the tracking service.

`src/lib/dataStore.tsx` Global state management using React Context API. Provides centralized access to reference data (countries, trip types, theme tags) with loading states and error handling. Includes DataStoreProvider and custom hooks (`useCountries()` , `useTripTypes()` , `useThemeTags()`) for consuming the store.

`src/lib/supabaseClient.ts` Supabase client initialization and authentication utilities. Creates and exports the Supabase client instance, provides authentication helpers (`getCurrentUser()` , `getAccessToken()`), and manages user session state.

`src/services/api.service.ts` Backend API client service providing typed functions for all API endpoints. Handles HTTP requests, error handling, retry logic for cold starts, authentication header injection, and response parsing. Abstracts API communication from components.

`src/services/tracking.service.ts` Event tracking service managing user interaction tracking. Handles event batching (10 events or 5 seconds), queueing, session initialization, device detection, and sending events to the backend analytics API. Provides a facade for tracking operations.

Backend Directory (`backend/`)

Configuration Files

`backend/.gitignore` Backend-specific git ignore patterns for Python artifacts, virtual environments, database files, and environment variables.

`backend/Procfile` Render.com process file specifying the command to start the application in production. Uses Gunicorn as the WSGI server.

`backend/requirements.txt` Python production dependencies including Flask, SQLAlchemy, PostgreSQL driver, authentication libraries, and other runtime requirements.

`backend/requirements-dev.txt` Development-only dependencies like testing frameworks, code formatters, and development tools.

`backend/runtime.txt` Python version specification for deployment platforms. Ensures consistent Python version across environments.

Application Package (`app/`)

`app/__init__.py` Package initialization file making the `app` directory a Python package. May contain package-level initialization code.

`app/main.py` Flask application entry point creating the Flask app instance, registering blueprints (v2, events, analytics, resources, system), configuring CORS, initializing database, and starting the development server. Main file executed when running the backend.

`app/core/config.py` Application configuration management loading settings from environment variables. Defines configuration classes for different environments (development, production) and provides centralized access to app settings.

`app/core/database.py` Database connection management using SQLAlchemy. Creates database engine with connection pooling, manages session factory with scoped sessions, and provides database initialization functions. Handles connection lifecycle, pooling configuration, and error detection.

`app/core/auth.py` Authentication utilities for Supabase JWT validation. Provides middleware and helpers (`get_current_user()`) for verifying user authentication tokens and extracting user information from requests.

`app/models/trip.py` SQLAlchemy models for trip-related entities using V2 schema. Defines models for TripTemplate, TripOccurrence, Company, Country, Guide, TripType, Tag, and their relationships. Maps database tables to Python objects with `to_dict()` methods for serialization.

`app/models/events.py` SQLAlchemy models for event tracking system. Defines User, Session, Event, TripInteraction, and EventType models. Represents the analytics data structure with relationships and computed fields.

`app/services/recommendation.py` Recommendation algorithm service implementing the scoring logic. Processes user preferences, queries database, scores trips based on multiple factors (geography, themes, budget, duration, difficulty), implements relaxed search for expanded results, and returns ranked results with match scores.

`app/services/events.py` Event tracking service handling user event processing. Manages user resolution (anonymous and registered), session creation with device info, event validation, and trip interaction updates. Implements business logic for analytics with real-time counter updates.

`app/api/v2/routes.py` API version 2 route handlers for trip-related endpoints. Provides REST endpoints for trips, templates, occurrences, companies, and recommendations. Uses the V2 schema (TripTemplate + TripOccurrence) with backward compatibility formatting.

`app/api/events/routes.py` Event tracking API endpoints. Handles POST requests for single events, batch events, session initialization, and user identification. Processes tracking data from the frontend and updates analytics counters.

`app/api/analytics/routes.py` Analytics and metrics API endpoints. Provides GET endpoints for current metrics, daily metrics breakdown, top searches, and evaluation scenarios. Computes metrics on-demand from raw data in the database.

`app/api/resources/routes.py` Resource data API endpoints. Provides read-only GET endpoints for countries, guides, trip types, tags, and locations. Serves reference data used in search forms and trip displays.

`app/api/system/routes.py` System API endpoints. Provides health check endpoint (`/api/health`) that returns service status and database statistics. Used for monitoring and deployment health checks.

Recommender Module (`recommender/`)

`recommender/logging.py` Request logging functionality for recommendation API calls. Logs user preferences, results, timing, and metadata for analytics and debugging. Generates unique request IDs for tracking and stores data in `recommendation_requests` table.

`recommender/metrics.py` Performance metrics aggregation for recommendation engine. Computes metrics on-demand from raw recommendation request data. Tracks response times, result counts, scoring distributions, and algorithm performance metrics. Provides functions for current metrics, daily metrics, and top searches.

`recommender/evaluation.py` Algorithm quality evaluation using test scenarios. Compares recommendation results against expected outcomes, calculates accuracy metrics, and validates algorithm behavior. Loads scenarios from database and runs automated tests.

Migrations (`migrations/`)

`migrations/_001_add_recommendation_logging.py` Migration script adding recommendation logging tables. Creates `recommendation_requests` table for storing recommendation request logs and metrics.

`migrations/_002_add_user_tracking.py` Migration adding user tracking tables. Creates User, Session, and Event tables for analytics with proper relationships and indexes.

`migrations/_003_add_companies.py` Migration adding Companies table. Introduces company/tour operator entity to the schema, enabling multi-company support.

`migrations/_004_refactor_trips_to_templates.py` Major migration refactoring trips into TripTemplate and TripOccurrence. Separates trip definition (template) from scheduled instances (occurrences), improving data normalization and enabling multiple occurrences per template.

`migrations/_005_normalize_events_and_load_scenarios.py` Migration normalizing event structure and adding scenario support. Refines event tracking schema with EventType table (3NF) and adds test scenario capabilities for evaluation.

`migrations/_006_add_properties_jsonb.py` Migration adding JSONB properties column for flexible data storage. Enables storing additional metadata without schema changes, providing extensibility for future features.

`migrations/run_schema_v2_migration.py` Migration runner script executing all V2 schema migrations in order. Ensures proper migration sequence and rollback capabilities.

Scenarios (`scenarios/`)

`scenarios/generated_personas.json` Generated test user personas for algorithm evaluation. Contains realistic user profiles with varied preferences used for testing recommendation quality and algorithm performance.

`scenarios/README.md` Scenarios documentation explaining how to use test personas and scenarios for evaluation.

Scripts (`scripts/`)

`scripts/verify_restructure.py` Verification script for project restructure. Validates that the project structure matches expected organization and checks for removed components.

`scripts/_archive/` Directory containing archived scripts that are no longer in active use. Includes old analysis and verification scripts from previous project versions.

Database Management Scripts (`scripts/db/`)

`scripts/db/seed.py` Database seeding script populating the database with initial data. Creates countries, trip types, tags, guides, companies, templates, and occurrences for development and testing.

`scripts/db/check_schema.py` Schema validation script verifying database structure matches expected models. Validates tables, columns, constraints, and relationships.

`scripts/db/check_db_state.py` Database state inspection script showing current data counts, relationships, and overall database health. Useful for debugging and monitoring.

`scripts/db/verify_schema.py` Schema verification utility comparing database schema against model definitions. Ensures schema consistency between code and database.

`scripts/db/verify_seed.py` Seed data verification script validating that seeded data meets expected criteria and relationships. Checks data integrity after seeding.

`scripts/db/check_types.py` Type checking utilities for database columns. Validates data types match model definitions and identifies type mismatches.

`scripts/db/verify_names.py` Name validation script checking naming consistency across database entities. Ensures proper Hebrew and English name formatting.

`scripts/db/check_guaranteed_trips.py` Guaranteed trips checker identifying trips with guaranteed departure status and validating their configuration. Helps maintain data quality for trip statuses.

Data Generation Scripts (`scripts/data_gen/`)

`scripts/data_gen/generate_personas.py` Test persona generation script creating realistic user profiles with varied preferences for testing recommendation scenarios. Generates diverse user personas for algorithm evaluation.

`scripts/data_gen/generate_trips.py` Trip data generation script creating test trip templates and occurrences with realistic attributes. Useful for populating test databases with sample data.

`scripts/data_gen/create_realistic_data.py` Realistic data creation script generating comprehensive test datasets with proper relationships and constraints. Creates interconnected data for integration testing.

Analytics Scripts (`scripts/analytics/`)

`scripts/analytics/aggregate_daily_metrics.py` Daily metrics aggregation script processing raw events into daily summaries. Calculates daily statistics for analytics dashboards. Can be run manually or scheduled via cron. Populates `recommendation_metrics_daily` table if it exists.

`scripts/analytics/aggregate_trip_interactions.py` Trip interaction metrics aggregation script calculating engagement metrics per trip. Computes click-through rates, save rates, and other interaction statistics from event data.

`scripts/analytics/analyze_scoring_v2.py` V2 scoring analysis script specifically analyzing the new recommendation algorithm's performance and scoring patterns. Helps tune scoring weights and thresholds.

`scripts/analytics/export_data.py` Data export utility exporting database data to CSV or JSON formats. Useful for backups, reporting, and data analysis outside the application.

`scripts/analytics/cleanup_sessions.py` Session cleanup job removing expired sessions and old event data. Maintains database performance and storage efficiency by archiving or deleting stale data.

Documentation Directory (`docs/`)

`docs/README.md` Documentation index and overview. Provides navigation to all documentation files and explains the documentation structure.

`docs/architecture/` Architecture documentation directory containing design patterns, database schema, and project structure documentation. Essential reading for understanding system design.

`docs/deployment/` Deployment guides for various platforms (Render, Vercel). Includes environment setup, configuration, and troubleshooting guides for production deployment.

`docs/technical/` Technical documentation covering authentication flows, database connections, troubleshooting, and implementation details. Contains guides for common technical tasks and issues.

`docs/testing/` Testing documentation including test plans and testing strategies. Documents how to test the application and what scenarios are covered.

`docs/proposals/` Feature proposals and design documents. Contains proposals for new features and improvements to the system.

`docs/archive/` Archived documentation from previous project versions. Contains historical migration plans, deprecated designs, and old documentation for reference.

`docs/DATA_PIPELINES.md` Comprehensive documentation of all data pipelines in the project. Explains how data flows from user interactions through to database storage and analytics.

`docs/API_STRUCTURE.md` Complete API structure documentation listing all endpoints, request/response formats, and API versioning. Reference guide for API consumers.

`docs/LOCAL_DEVELOPMENT_SETUP.md` Local development setup guide explaining how to configure and run the application locally. Includes environment variable setup and database configuration.

`docs/SUPABASE_OAUTH_SETUP.md` Supabase OAuth configuration guide. Documents how to set up OAuth providers (Google, etc.) and configure authentication in Supabase.

Architecture Overview

Frontend Architecture

The frontend follows Next.js App Router architecture with:

- **Pages**: Route-based components in `app/` directory using file-based routing
- **Components**: Reusable UI components organized by feature and type
- **Hooks**: Custom React hooks for shared logic (tracking, data access)
- **Services**: API communication layer with centralized error handling and retry logic
- **Lib**: Utilities and state management using React Context API

Backend Architecture

The backend follows a layered architecture:

- **API Layer**: Route handlers in `app/api/` organized by feature (v2, events, analytics, resources, system)
- **Service Layer**: Business logic in `app/services/` (recommendation algorithm, event processing)
- **Model Layer**: Database models in `app/models/` using SQLAlchemy ORM
- **Core Layer**: Configuration and infrastructure in `app/core/` (database, auth, config)

Data Flow

1. User interacts with frontend (`frontend/src/app/`)
2. Frontend calls API service (`frontend/src/services/api.service.ts`)
3. Request reaches backend route (`backend/app/api/`)
4. Route calls service layer (`backend/app/services/`)
5. Service queries models (`backend/app/models/`)
6. Models interact with database via SQLAlchemy
7. Response flows back through layers to frontend
8. Events are tracked and logged for analytics

Key Design Decisions

1. **Monorepo Structure**: Frontend and backend in separate directories but same repository for easier development and deployment coordination.

2. **Next.js App Router**: Modern file-based routing system providing better performance and developer experience than Pages Router.

3. **Layered Backend**: Separation of concerns with API routes, services, and models enabling testability and maintainability.

4. **V2 Schema**: TripTemplate/TripOccurrence separation allows multiple scheduled instances of the same trip template, improving data normalization and reducing duplication.

5. **Event Tracking Module**: Separate module for analytics enables independent development and scaling of tracking features without affecting core functionality.

6. **Blueprint Organization**: API endpoints organized by feature (v2, events, analytics, resources, system) rather than by version, making it easier to find and maintain related endpoints.

7. **On-Demand Metrics**: Metrics computed on-demand from raw data rather than pre-aggregated, ensuring always up-to-date results without background job complexity.
8. **Migration System**: Versioned migrations enable schema evolution and rollback capabilities, supporting safe database changes.
9. **Script Organization**: Scripts organized by purpose (db, data_gen, analytics) for easy discovery and maintenance.
10. **Centralized API Client**: Frontend uses centralized API service with retry logic, authentication, and consistent error handling, reducing code duplication.

This structure supports scalability, maintainability, and clear separation of concerns while enabling efficient development workflows and easy onboarding for new developers.