AdventuresTracked Project Plan

Brent Reynolds

Project Overview:

Deadline: August 9th, 2024 at 12:00 pm

Project Name: TBD

Objective: Develop a full stack web application that allows users to track their trips. This includes managing trips, trip legs, journals for each trip, photos for each trip, and calculating total miles traveled.

Features:

- User authentication (login, registration)
- Dashboard for displaying all trips for the user
- Ability to add, edit, and delete trips
- Detailed view for each trip, including:
    - Trip legs
    - Journal/Notes
    - Photo gallery
- A visual representation of the trips taken for the year
- Calculation and display of total miles for each trip and for the year as a whole

Technologies Used:

Frontend: React.js

Backend: .NET Web API

Database: Start with SQLite for development

Authentication: TBD - Maybe use login with Google

Hosting: TBD

Entity Relationship Diagram:

User Table:

Column Name	Data Type	Constraints	Description
UserId	TEXT	Primary Key	Unique identifier for the user, stored as UUID
Username	TEXT	Not Null, Unique	Username chosen by the user
Email	TEXT	Not Null, Unique	Email address of the user
PasswordHash	TEXT	Not Null	Hash of the user's password
CreatedAt	DATETIME	Not Null	Timestamp when the user account was created

Trips Table:

Column Name	Data Type	Constraints	Description
TripId	TEXT	Primary Key	Unique identifier for the trip stored as a UUID
UserId	TEXT	Not Null, Foreign Key	Identifier for the user who created the trip
TripName	TEXT	Not Null	Name of the trip
StartDate	DATE	Not Null	Start date of the trip
EndDate	DATE	Not Null	End date of the trip
CreatedAt	DATETIME	Not Null	Timestamp when the trip was created

Legs Table:

Column Name	Data Type	Constraints	Description
LegId	TEXT	Primary Key	Unique identifier for the leg of the trip, stored as a UUID
TripId	TEXT	Not Null, Foreign Key	Identifier for the trip this leg belongs to
DepartureAirportId	TEXT	Not Null, Foreign Key	Airport Id of the departure airport
ArrivalAirportId	TEXT	Not Null, Foreign Key	Airport Id of the arrival airport
DepartureDate	DATE	Not Null	Departure date of this leg
ArrivalDate	DATE	Not Null	Arrival date of this leg
DistanceMiles	REAL	Not Null	The distance traveled in miles between the DepartureAirport and

			the ArrivalAirport
CreatedAt	DATE	Not Null	The date the leg was created on

Journals Table:

Column Name	Data Type	Constraints	Description
JournalId	TEXT	Primary Key	Unique identifier for the journal entry, stored as UUID
TripId	TEXT	Not Null, Foreign Key	Identifier for the trip this journal belongs to
Title	TEXT	Not Null	Title of the journal entry
Entry	TEXT	Not Null	Text content of the journal entry
CreatedAt	DATETIME	Not Null	Timestamp when the journal entry was created

Photos Table:

Column Name	Data Type	Constraints	Description
PhotoId	TEXT	Primary Key	Unique identifier for the photo, stored as UUID
TripId	TEXT	Not Null, Foreign Key	Identifier for the trip this photo belongs to
PhotoUrl	TEXT	Not Null	URL where this photo is stored
Description	TEXT	Nullable	Optional description

			of the photo
CreatedAt	DATETIME	Not Null	Timestamp when the photo was uploaded

Airports Table:

Column Name	**Data Type**	**Constraints**	**Description**
AirportId	TEXT	Primary Key	Unique identifier for the airport, stored as a UUID
Name	TEXT	Not Null	Name of the airport, may include the city name
City	TEXT	Not Null	Main city served by the airport
Country	TEXT	Not Null	Country or territory where the airport is located
IATA	TEXT	Nullable, Indexed	3-letter IATA code, null if not assigned
ICAO	TEXT	Nullable, Indexed	4-letter ICAO code, null if not assigned
Latitude	REAL	Not Null	Latitude in decimal degrees
Longitude	REAL	Not Null	Longitude in decimal degrees
Altitude	INTEGER	Not Null	Altitude in feet
Timezone	REAL	Not Null	Hours offset from UTC
DST	TEXT	Not Null	Daylight savings time category
TzDatabaseTimezone	TEXT	Not Null	Olson timezone format

Type	TEXT	Not Null	Type of the airport
Source	TEXT	Not Null	Source of the data

- **A description of all of the columns in the Airports table can be found at this website: [OpenFlights: Airport and airline data](#).**

Cardinality:

1. Users and Trips

Relationship: One-to-Many

Description: Each user can record multiple trips, but each trip is recorded by only one user.

Cardinality Details:

- Users (one) to Trips (many)
- Foreign key in Trips table (UserId) references the primary key in Users table (UserId).

2. Trips and Legs

Relationship: One-to-Many

Description: Each trip can consist of multiple legs (segments of the trip), but each leg belongs to only one trip.

Cardinality Details:

- Trips (one) to Legs (many)
- Foreign key in Legs table (TripId) references the primary key in Trips table (TripId).

3. Trips and Journals

Relationship: One-to-Many

Description: Each trip can have multiple journal entries describing different experiences or days, but each journal entry is associated with only one trip.

Cardinality Details:

- Trips (one) to Journals (many)
- Foreign key in Journals table (TripId) references the primary key in Trips table (TripId).

4. Trips and Photos

Relationship: One-to-Many

Description: Each trip can have multiple photos taken during the trip, but each photo is linked to only one trip.

Cardinality Details:

- Trips (one) to Photos (many)
- Foreign key in Photos table (TripId) references the primary key in Trips table (TripId).

5. Airports (used in Legs)

Relationship: Many-to-Many (indirectly through Legs)

Description: Airports serve as departure and arrival points for trip legs. Each airport can serve as the departure or arrival location for many different legs, and each leg can involve up to two different airports (one for departure, one for arrival).

Cardinality Details:

Since airports relate to legs in a many-to-many fashion, the connection is managed through the Departure Code and Arrival Code in the Legs table, which would reference the IATA/ICAO codes in the Airports table.

Notes on the database and the user interface logic:

- At least one of the airport codes is required, either the IATA code or the ICAO code for both the departure airport and the arrival airport of each leg. Since not every airport will have both an IATA code and an ICAO code, these need to be nullable columns. The application logic and database queries need to be able to handle scenarios where one of

these codes is null. The query should be able to return the correct airport data if the user inputs one of the airport codes even if the other code is null. These columns in the Airports table need to be indexed because they will be frequently queried when the user looks up an airport code. Indexing these columns will make the search process faster.

- **Validation logic in the application to check user input against the Airports table:**
    - **Check IATA Code:** First, attempt to match the user input against the IATA codes in the database. Most people know airports by their three letter IATA codes so most people will attempt to enter an airport IATA code.
    - **Fallback to ICAO Code:** If the IATA code does not yield a match, then check against the ICAO codes.
    - **Error Handling:** If neither code matches, provide a user-friendly error message. Possibly, add functionality to allow the user to request that an airport be added to the database.
- **Application Logic:**
    - **User Interface:** The UI should guide users through inputting airport codes. Possibly by providing auto-complete functionality that suggests airports as they start typing a code.

TODO: Validation to ensure that users cannot enter malicious data into the application in any of the text areas or the photo file upload is to be decided. Also, the storage of photos in the database is to be decided. I am not sure how I will store the photos yet. Possibly, I can store the photos in an external storage system and keep references to the photos within the database.

TODO: Create Figma designs for the rest of the web app.