SUM System Design

SUM

React-Native Application

expo-sqlite persistant db across app restarts
metro.config.js root for project assets

[Data from Expo Docs SDK/Sqlite]

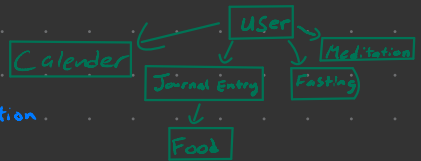
Const db = SQLite.openDatabase("db.db") does not store db file directly in the project
IIS managed by Expo SDK
Expo Sqlite pkg abstracts physical file location and provides connection object.
I likely will be using an ORM here to create my DB tables as that's what I'm used to.
OPTIONS: TypeORM, WatermelonDB, Sequelize, SQLite ORM
Sequelize: Large community and extensive Docs. [Primarily for server side use.
WatermelonDB is ideal for offline data storage and synchronization. Intuitive API, needs to work offline and sync when network is available
For local storage our options then become either AsyncStorage or WatermelonDB but they're very different
Watermelon useful for complex data structs & large datasets, uses SQLite db engine.
AsyncStorage: is a simple key-value storage mechanism primarily used for basic local storage doesn't offer advanced querying or relational data modelling. Performance also degrades w/ larger data sets
For our app then we decided to use WatermelonDB.
WatermelonDB goal is real world performance — Open Source.
But the main reason were going for WMDB is b/c it's 1. ORM 2. Relational DB.
Redux and WatermelonDB serve different purposes and CAN be used together.
In some cases WatermelonDB can replace the need for redux when it comes to managing local data persistence & creating db.
WmDB & ReduX can co-exist in RN application. WmDB primarily focuses on local persistent data and management while redux provides a broader framework for managing app state.
In our use case we will be using WmDB w/ Redux.
Our main use case for WmDB will be its own object relational mapping (ORM Layer)
ORMs help define models, perform db queries & manage relationships b/w entities.
*Cocoapods pkg manager for iOS development & RN App for iOS devices requires it.
Babel bug "require decorator bfe export option, value must be bool is caused by using babel/plugin-proposal-decorators the dev dependencies. Adding Legacy: True adds this dependency to dev dependencies again without us knowing.

WatermelonDB has a lot of installation issues, reverting back to SQLite...

Query Builder: Provides a programmatic interface for constructing db queries. Simplifying process of building queries. It uses methods & functions w/ builder library.
1. Simplified query construction | 2. Dynamic query generation | 3. Database Agnostics | 4. Safety & Security
Just use SQLite w/o sql builder
DB obj referenced using db [const db = SQLite.openDatabase("db.db")]

Data Model

what is the relationship b/w Calender to (Fasting, Meditation, Journal)



We should discuss the configuration of journal entries, is it just for food daily log completion or relationship w/ Fasting/Meditation

User
- id - PK
- email - String
- name - String
- MembershipStatus
- weight
- height

JournalEntry
- id - PK
- user_id - FK
- date - Datetime
- food_id - FK
- EventID - FK

Food
- id - PK | user_id
- userID - FK
- name - String

PK - Primary Key
FK - Foreign Key

with this structure we can establish a many-to one relationship between the Food tabe and JournalEntry table

~~**Meditation**~~
- ~~id - PK~~
- ~~user_id - FK~~
- ~~date - Timestamp~~
- ~~duration - TIME(Mins)~~
- ~~mood - String(optional)~~

~~**Calender**~~
- ~~id - PK~~
- ~~date - TEXT~~
- ~~Fasting_id - FK~~
- ~~Meditation_id - FK~~
- ~~food_journal_id - FK~~

*Fasting/Meditation can be one table (Activity/Event)

Event
- id - PK
- User_id - FK
- start_time - TIME
- End_time - TIME
- Duration - (hours/Mins)
- notes - String(optional)
- mood - String
- Type (Meditation/Fasting

Meal (Many-to-one) -> Food table
- ID - uuid
- FOODID - FK
- JournalEntryID - FK
- mealCategory

Date of calender entry was created

By linking respective FKS to Calender Table, We establish relationship between (Fasting, meditation, journal) and corresponding dates in the Calendar

This Struct allows us to query Calendar Table based on dates and retrieve data for each of these activities.

Videos
- id - PK
- videoUrl - String
- title - String
- description - String
- category - String
- likes - int
- views - int
- upload_date - time
- thumbnail_url - String

Videos are very Resource-intensive utilize CDN

FoodHistory

Users food history should generally be cached Local cache

Asynch Storage in RN

Backend Storage: We will only need this when user wants to access data from other device such as website.

Notes can be user writing how they feel during the fasting period

We should also discuss the motivational short videos we want to implement, how we would use a video scrolling feed like (Tiktok) for motivation

SUM App System Design

System Design: Distributed Systems Backend

Clients is a Frontend UI Design Problem

Customers —> SERVER —> Database

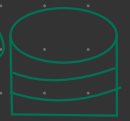
Frontend
Mobile (tablet/Phone)
Web (Desktop)

Server — Data Management CRUD — CREATE DELETE UPDATE. READ

API's

getCalendar(id, timestamp, author)
postCalendar
getUser
getFasting
postHistory
getHistory

SQL DB Local Storage
SQLite



Author is user Foreign key

Calendar

ID	Timestamp	Author
1	Date	User
2		
3		
4		
5		

User

ID	
1	Name

History

ID	Food
1	Food_List

Mobile specific Problem

1. Limited Resources: Internet traffic, battery usage, memory
2. High Level architecture 3 Design Patterns
4. UI Architecture 5 Backend
6. Data storage 7. App specific

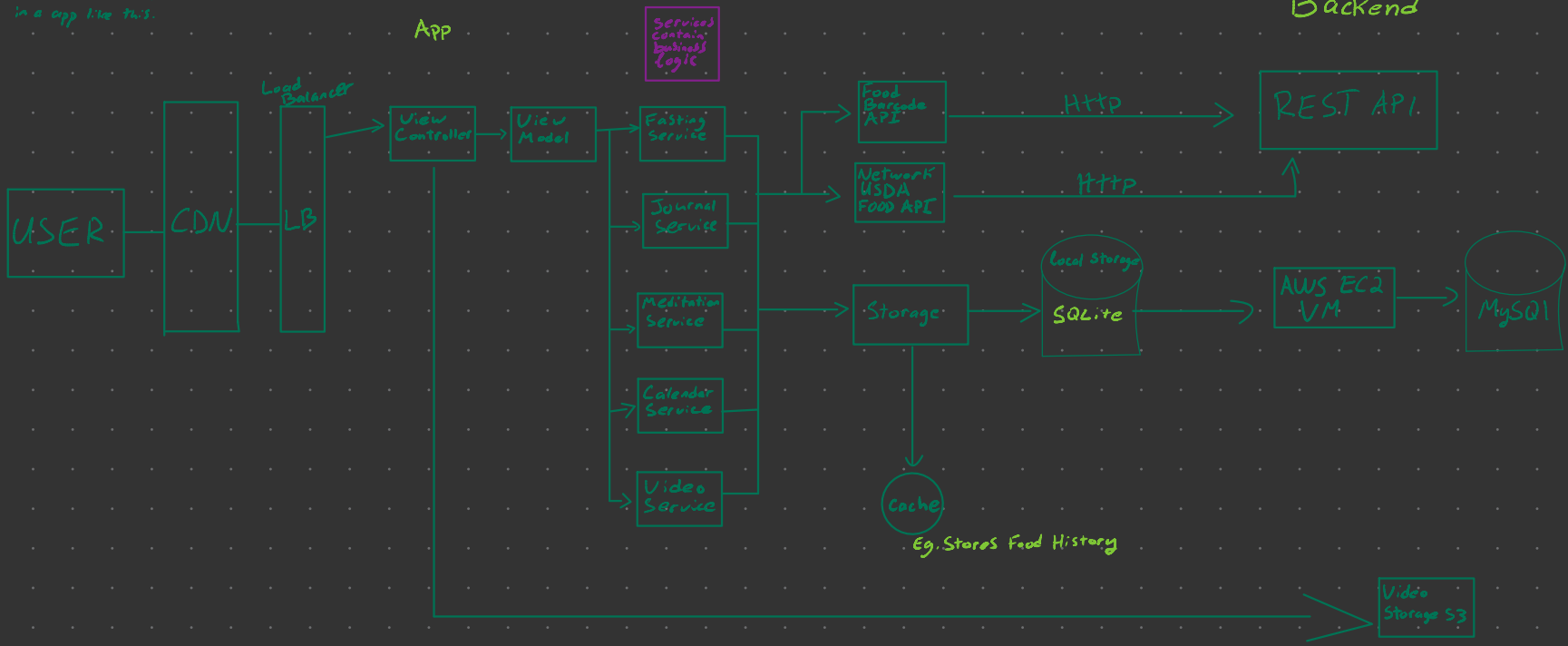
Solution needs to be organized to reduce usage of these resources.

Required Functions

- Fasting Timer
- Food Diary
- Food History
- Meditation Timer
- Calendar
- Offline Mode
- Food Search
- Video feed (for fasting)

Architecture

LB most likely unnecessary in a app like this.



UI LAYER

If Expo AsyncStorage meets data storage requirements we won't need to use SQLite.
AsyncStorage is a key-value storage allows small data
however complex queries aren't possible.

I can take the step to clone watermelonDB and try again.

CouchDB works w/ local or on a server.
handles synch across devices, synch local dB and CouchDB server.
Document Structure: Stores data in JSON Docs, self-contained, can hold nested data structures. Can reference related documents using unique identifiers AKA DOC IDS.

WatermelonDB: Mobile-focused dB & Synch Solution built on SQLite. Offline Ready
often uses caching and in-memory dB for fast & responsive data access.