DayBoard: Design and Setup Guide

Introduction

DayBoard is a cross-platform desktop and mobile application designed to help interns
and students manage their daily schedule and finances. It brings together your meeting calendar, recurring bills, and internship compensation into a single dashboard, offering actionable insights like estimated take-home pay, daily commute costs, and upcoming renewals.

Architecture Overview

DayBoard is composed of a SwiftUI client (macOS and iOS) and a lightweight Go backend. The client displays a menu bar card summarising your next meeting, commute costs, subscription renewals, and pay outlook. The backend handles OAuth with external providers (Google Calendar, Outlook Calendar, Plaid), fetches and normalizes data, stores it in PostgreSQL, and exposes RESTful endpoints for the client.

Backend Stack

* Language: Go 1.22+
* Framework: Gin (HTTP routing), pq (PostgreSQL driver)
* Database: PostgreSQL (e.g. Neon/Supabase cloud)
* Migrations: Goose or Atlas
* External APIs: Google Calendar, Microsoft Outlook, Plaid Link, Google Distance Matrix

Frontend Stack

* Language: SwiftUI (macOS/iOS)
* Networking: URLSession + Combine
* Notifications: UserNotifications
* Persistence: Keychain for local secrets
* UX: Menu bar extra on macOS; application scene on iOS

Backend Setup

1. Create the project directory structure under `dayboard/backend` as shown in this repository.
2. Write the `go.mod` file with the Gin and PostgreSQL driver dependencies.
3. Add SQL migration scripts under `backend/migrations` to create the required tables (users, oauth_tokens, calendar_events, subscriptions, transactions, profiles, tax tables, and city cost models).
4. Implement a Go service in `cmd/server/main.go` using Gin. Start with simple routes under `/api/v1` and stub handlers for agenda, subscriptions, tax estimation, commute estimation, and profile management.
5. Set up OAuth credentials in Google Cloud Console for Calendar API, Microsoft Azure Portal for Outlook, and Plaid Link (development environment). Configure redirect URIs pointing to your backend (e.g. `https://your-domain.com/auth/google/callback`).
6. Store access and refresh tokens encrypted using a secure library (e.g. libsodium) in the `oauth_tokens` table.
7. Implement services in the internal package to fetch calendar events, detect recurring charges from Plaid transactions, estimate taxes using bracket data from the database, and call the Google Distance Matrix API for commute estimates.
8. Expose RESTful endpoints as defined in the API contract and secure them with session cookies or JWTs.
9. Deploy the backend to a cloud platform (Fly.io, Render, Railway, or Cloud Run) and set environment variables like `DATABASE_URL`, `GOOGLE_CLIENT_ID`, `GOOGLE_CLIENT_SECRET`, `PLAID_CLIENT_ID`, and `PLAID_SECRET`.

Frontend Setup

1. Open Xcode and create a new SwiftUI App project. Replace the default content with the provided `DayBoardApp.swift` and `ContentView`.
2. Add a `DayBoardViewModel` class to orchestrate network calls to the backend. Use `URLSession` and `Combine` to fetch the agenda, commute estimates, subscriptions, and pay outlook.
3. Configure a menu bar extra (macOS) or an app scene (iOS) to display the next event, commute cost, bills this week, and pay outlook.
4. Request notification permissions and schedule local notifications for upcoming meetings (10-minute reminders) and bill renewals. Local notifications do not require a server but rely on the client's persisted schedule.
5. Provide a settings screen where users can connect their calendars (Google and Outlook) and Plaid accounts via the backend's OAuth endpoints. Store session cookies securely in the Keychain.
6. When ready to distribute, code-sign and notarize the macOS binary and upload the DMG to your release page. For iOS distribution, enroll in the Apple Developer Program and use TestFlight for external testers.

Continuous Integration & Deployment

Use GitHub Actions to automate testing and deployment. For the backend, set up a workflow that runs `go test`, builds a Docker image, and deploys it to your chosen hosting provider. For the client, configure a macOS runner to build and notarize the app bundle. Attach the DMG to GitHub Releases on tags.

Next Steps

Start by creating the backend project and running the server locally. Once the REST endpoints are stubbed, build the SwiftUI interface to consume them. Iterate by gradually replacing placeholder data with live data fetched via OAuth. Document API keys and environment variables in a `.env` file, and commit database migrations to version control.