Building a Mobile Facility Management App

with Flutter, FastAPI, PostgreSQL, AWS & Claude AI

Course Duration: 12 Days

Overview

This course teaches participants how to design and build a real, production-ready mobile application for managing maintenance requests and facility workflows across multiple restaurant outlets. The program follows a foundations-first approach — every technology layer is introduced conceptually before any code is written, ensuring participants understand the why behind every decision.

A key element of this course is the Claude AI vibe coding methodology. Rather than memorizing syntax, participants learn to describe features in plain language, prompt Claude to generate code, review the output, and make targeted modifications. By the end of the program, participants will be confident enough to extend the system independently using Claude as their development partner.

Audience Profile

This program is designed for those individuals who need to understand, use, and maintain a custom-built facility management mobile application. No prior programming experience is required. Participants who are comfortable using Google Forms and basic digital tools will find this course accessible and practical.

Prerequisites

- ☐ Basic comfort with digital tools such as Google Forms, WhatsApp, and spreadsheets
- ☐ Understanding of the outlet's existing maintenance and repair workflow
- ☐ A laptop or desktop computer with internet access
- ☐ A Google or Microsoft account for cloud service access during training
- ☐ AWS free-tier account set up

Technology Stack

- ☐ Database: PostgreSQL 16 — relational database for storing all outlet, product, repair, and vendor data
- ☐ Cloud Database: AWS RDS — managed PostgreSQL hosting on Amazon Web Services
- ☐ Mobile App: Flutter (Dart) — single codebase delivering both Android APK and iOS app
- ☐ Backend API: FastAPI (Python) — REST API connecting the mobile app to the database
- ☐ File Storage: AWS S3 — storing vendor quotation PDFs and photo attachments
- ☐ Deployment: AWS EC2 with Nginx — hosting the FastAPI backend with SSL
- ☐ Notifications: WhatsApp Business API — automated workflow notifications to outlet channels
- ☐ AI Copilot: Claude (Anthropic) — vibe coding partner for generating, explaining, and refining all code

Course Syllabus

Day 1: Understanding Databases — Why Before How

Module 1: What Is a Database?

- ☐ Why databases over spreadsheets and Google Forms

- ☐ Real-world analogies: ledger, index, register
- ☐ Relational vs non-relational databases — when to use which
- ☐ Tables, rows, columns, primary keys, and foreign keys explained visually
- ☐ Mapping existing Google Form fields to database table columns
- ☐ Introduction to PostgreSQL and where AWS RDS fits in

Module 2: Identifying Your Data Entities

- ☐ Business entity mapping: Outlets, Products, Repairs, Vendors, Requests
- ☐ Drawing entity relationship diagrams on paper
- ☐ Identifying primary keys and relationships between entities
- ☐ Why a unique product ID enables full repair history lookup

Claude Demo

- ☐ Prompt Claude: explain what database tables are needed for a restaurant maintenance system using simple language
- ☐ Observe how Claude reasons through entity design and relationships

Hands-On Lab

- ☐ Draw your own entity diagram: Outlets, Products, Repairs, Vendors
- ☐ Map all existing Google Form fields to table columns
- ☐ Identify unique identifiers and potential primary keys

Day 2: PostgreSQL Hands-On — Queries, Schema, and Data

Module 1: Getting Started with PostgreSQL

- ☐ Connecting to PostgreSQL using pgAdmin
- ☐ Navigating the pgAdmin interface: databases, schemas, tables
- ☐ Understanding data types: TEXT, INTEGER, NUMERIC, BOOLEAN, TIMESTAMP
- ☐ Running your first SELECT query

Module 2: Core SQL Operations

- ☐ SELECT, WHERE, ORDER BY, LIMIT — reading data
- ☐ INSERT, UPDATE, DELETE — managing records
- ☐ CREATE TABLE and ALTER TABLE — defining structure
- ☐ Joins: connecting products to their repair history
- ☐ Aggregate functions: SUM, COUNT, AVG for cost analysis
- ☐ Introduction to indexes and why lookup speed matters

Claude Demo

- ☐ Prompt Claude: create a Products table with unique ID, name, purchase price, and outlet ID
- ☐ Prompt Claude: write a query to fetch all repair records for a given product ID with total amount spent
- ☐ Run the generated SQL live and validate results in pgAdmin

Hands-On Lab

- ☐ Create Products and Repairs tables in local PostgreSQL
- ☐ Insert 5 sample product records and 10 repair records manually
- ☐ Run a query showing total repair cost per product, sorted highest first
- ☐ Modify the table: add a warranty_expiry column to Products

Day 3: Schema Design — Building the Real Database

Module 1: Normalization Principles

- What normalization means in plain language
- First, second, and third normal form — practical examples only
- Why duplication is dangerous and how foreign keys solve it
- Audit columns: created_at, updated_at, created_by

Module 2: Full Schema for the Facility Management System

- Tables: Outlets, Users, Products, Maintenance Requests, Repair History, Vendors, Quotations
- Status workflow table: modelling Pending, Approved, In Progress, Resolved
- Role columns: manager, approver, admin
- Introduction to database migrations — versioning schema changes

Claude Demo

- Prompt Claude: design a complete PostgreSQL schema for the restaurant facility management system with all tables, constraints, and indexes
- Walk through every generated table, explain each design decision
- Prompt Claude: add a preventive maintenance schedule table and integrate it

Hands-On Lab

- Execute the Claude-generated migration script in PostgreSQL
- Validate all relationships by inserting test data across all tables
- Simulate a full repair workflow: create request, add quotation, approve, log repair cost

Day 4: Flutter Foundations — Getting Started Without Getting Lost

Module 1: What Is Flutter?

- Flutter vs native Android and iOS — the cross-platform advantage
- Why one codebase delivers both Android APK and iOS app
- Introduction to Dart — just enough to read the language
- Flutter project structure: lib, pubspec.yaml, assets explained

Module 2: Understanding Widgets

- Everything is a widget — the core Flutter concept
- Stateless vs StatefulWidget — when state is needed
- Core layout widgets: Scaffold, Column, Row, Container, Padding, Text
- Hot reload and hot restart — the Flutter development loop
- Basic navigation: Navigator.push between two screens
- How to read Flutter code generated by Claude — a pattern recognition guide

Claude Demo

- Prompt Claude: create a Flutter screen showing a list of maintenance requests with product name, date, and color-coded status chips
- Walk through the generated widget tree structure step by step

Hands-On Lab

- Set up Flutter and run the starter app on emulator

- Build a static screen showing 3 product cards with name, ID, and last repair date
- Add navigation: tap a card to open a detail screen
- Modify Claude-generated code: change colors, add a new field, reorder layout

Day 5: Flutter Forms and State — Capturing Data from Center Managers

Module 1: Building Forms in Flutter

- Flutter Form widget and GlobalKey for validation
- TextFormField with validators: required, minimum length
- DropdownButtonFormField: outlet selector, product category
- DatePicker and time selection widgets
- Showing SnackBars for success and error feedback

Module 2: Managing Form State

- setState and TextEditingController for managing input values
- Collecting all form fields into a Map for submission
- Introduction to pubspec.yaml — adding Flutter packages
- Real-time field interactions: auto-fetch product details on ID entry

Claude Demo

- Prompt Claude: create a Flutter maintenance request form with outlet dropdown, product ID auto-fetch, issue description, urgency selector, and photo attachment
- Show how Claude builds the complete validated form in a single prompt

Hands-On Lab

- Build the maintenance request form using Claude's generated code
- Implement real-time product detail display when a product ID is entered
- Add form validation: required fields and minimum description length
- Export form data as a formatted JSON object on submission

Day 6: FastAPI Foundations — The Bridge Between App and Database

Module 1: What Is a REST API?

- APIs explained in plain language: why the app cannot talk to the database directly
- HTTP methods: GET, POST, PUT, PATCH, DELETE
- JSON format: how data travels between app and server
- Status codes: 200, 201, 400, 404, 500 — what each means

Module 2: Building the FastAPI Backend

- FastAPI setup and project structure
- Path parameters and query parameters
- Request and response models with Pydantic
- Connecting to PostgreSQL using SQLAlchemy ORM
- CRUD endpoints: GET /products/{id}, POST /requests, GET /requests
- Swagger UI: testing the API without writing any frontend code
- Environment variables and .env files for database credentials

Claude Demo

- Prompt Claude: build a FastAPI backend with GET /products/{product_id} returning product details plus total repair cost and last repair date, and POST /maintenance-requests to create a new request
- Run the API live and test all endpoints in Swagger UI

Hands-On Lab

- Set up FastAPI project and connect to local PostgreSQL
- Run the Claude-generated API and test via Swagger UI
- Test GET /products/101 — verify product details and repair history populate
- POST a maintenance request and confirm it appears in the database

Day 7: Connecting Flutter to FastAPI — Making the App Come Alive

Module 1: HTTP in Flutter

- Making API calls using the dio package
- GET and POST requests from Flutter
- Parsing JSON responses into Dart model classes
- FutureBuilder for async data loading with loading indicators
- Error handling: network failures, 404 responses, validation errors

Module 2: End-to-End Integration

- Product auto-fetch: calling GET /products/{id} on text field input
- Submitting the maintenance form to POST /maintenance-requests
- Requests List screen: fetching and displaying all pending requests
- Pull-to-refresh for live data updates
- API base URL configuration for local vs production environments
- CORS configuration in FastAPI for Flutter clients

Claude Demo

- Prompt Claude: connect the Flutter maintenance form to the FastAPI backend — auto-fetch product details on ID entry and submit the form to the API with a success screen
- Show the complete live end-to-end connection working in real time

Hands-On Lab

- Integrate the product ID auto-fetch feature into the maintenance form
- Connect form submission to POST /maintenance-requests
- Build a Requests List screen that fetches and displays all pending requests
- Add pull-to-refresh on the list screen

Day 8: Approval Workflows and Vendor Quotations

Module 1: Modelling Business Workflows

- Status state machines: Pending, Approved, In Progress, Resolved
- Role-based views: submitter screens vs approver screens
- Audit trail: logging who approved what and when
- Notification trigger points in the workflow

Module 2: Approval and Quotation Features

- Approver dashboard: viewing pending requests with product history and cost context

- Approve and Reject actions with comment modal
- File upload in Flutter: picking images and PDFs for quotation attachments
- Multipart form data for file uploads via FastAPI
- Storing quotation metadata in PostgreSQL
- Cost comparison view: current quotation vs historical repair spend
- PATCH /requests/{id}/status endpoint for status updates

Claude Demo

- Prompt Claude: build an approver dashboard in Flutter showing pending requests with product repair history and quotation attachments, with Approve and Reject actions
- Prompt Claude: build the FastAPI status update endpoint with approver ID and timestamp logging

Hands-On Lab

- Build the approver dashboard screen with request cards
- Implement the Approve and Reject action with comment modal
- Add vendor quotation upload to a request
- View cost comparison: current quotation versus historical repair cost

Day 9: WhatsApp Integration — Real-Time Notifications

Module 1: WhatsApp Business API Overview

- WhatsApp Business API vs personal WhatsApp — how it works
- Setting up a Meta Business account and sandbox environment
- Message templates: pre-approved notification formats
- Webhook endpoints: receiving WhatsApp replies in FastAPI

Module 2: Sending Notifications from FastAPI

- Sending messages from FastAPI using httpx to the WhatsApp Cloud API
- Event-driven notifications: triggering on database status changes
- FastAPI BackgroundTasks for non-blocking notification dispatch
- Notification messages: request submitted, approved, and resolved events
- Configuring notification channels for different outlet groups

Claude Demo

- Prompt Claude: add WhatsApp notification functionality to FastAPI — send a notification to the outlet manager when a maintenance request is created, and notify the submitter on approval
- Show the WhatsApp message arrive on a real phone in real time

Hands-On Lab

- Set up a test WhatsApp Business number using the Meta sandbox
- Trigger a WhatsApp notification on maintenance request creation
- Send an approval notification when request status changes to Approved
- Test a notification to a group channel using WhatsApp broadcast

Day 10: AWS Deployment — Taking the System Live

Module 1: AWS Services for This Project

- AWS overview for this system: RDS, EC2, S3, and API Gateway

- AWS free tier: what is available and how to stay within limits
- Security groups and IAM roles: controlling access
- Environment variables in production: AWS Secrets Manager basics

Module 2: Deploying the Backend and Database

- Setting up AWS RDS PostgreSQL instance
- Running database migrations on production
- Deploying FastAPI with Gunicorn and Nginx on EC2
- Configuring SSL with Let's Encrypt
- Setting up a systemd service for the FastAPI app
- S3 bucket configuration for quotation file attachments
- Monitoring basics: CloudWatch logs for the FastAPI server

Module 3: Building and Distributing the Mobile App

- Building the Flutter Android APK for distribution
- Installing the APK on physical Android devices
- iOS build overview: requirements and distribution via TestFlight
- Pointing the Flutter app to the live AWS API endpoint

Claude Demo

- Prompt Claude: write a deployment checklist and Nginx configuration for deploying FastAPI on Ubuntu EC2 with systemd service and SSL
- Walk through deployment live on an EC2 instance

Hands-On Lab

- Launch an RDS PostgreSQL instance and run the migration script
- Deploy FastAPI to EC2 and test all endpoints from Postman
- Configure S3 bucket for file uploads and update FastAPI to use it
- Build the Flutter APK, install on a physical Android device, and test the full workflow

Days 11–12: Capstone — Build a New Feature End-to-End Using Claude

Module 1: Prompt Engineering for Vibe Coding

- Crafting effective Claude prompts: specificity, context, and expected output
- Iterative prompting: prompt, review, test, refine
- Asking Claude to explain, improve, or debug generated code
- Reading Claude-generated code confidently and making targeted changes

Module 2: Capstone Feature Options

- Option A: Preventive Maintenance Scheduling — recurring checks by equipment type with WhatsApp alerts before due date
- Option B: Vendor Performance Dashboard — rate vendors after each repair, visualize cost and turnaround trends in Flutter
- Option C: Budget Tracker — set monthly maintenance budgets per outlet, flag requests that would exceed the budget

Module 3: End-to-End Feature Build

- Design the feature: database changes, API endpoints, Flutter screens

- Day 11: prompt Claude iteratively to build database schema and API endpoints
- Day 12: prompt Claude to build the Flutter UI, connect to the API, and test end-to-end
- Deploy the new feature to the live AWS environment

Claude Demo

- Live capstone kick-off: each participant writes their first Claude prompt on screen
- Trainer walks through how to evaluate and refine Claude output as a group

Hands-On Lab

- Day 11: design your feature and build database and API components using Claude
- Day 12: build the Flutter UI, connect to the API, and run end-to-end tests
- Day 12: 10-minute group demo — present your working feature to the class
- Day 12: push your completed feature to the live AWS environment