AI-Powered Personal Finance Assistant

ASE 485 Capstone — Project Plan

Josh · Spring 2026

Project Vision

An AI-powered personal finance app that uses **machine learning** to:

- **Auto-categorize** transactions with 80%+ accuracy
- **Generate personalized budgets** from spending history
- **Provide actionable savings recommendations** (3+ per analysis)
- **Adapt** budgets as spending patterns change over time

Tech Stack

Layer	Technology
Frontend	Flutter / Dart (mobile + web)
Backend	FastAPI (Python)
Database	PostgreSQL
ML Pipeline	scikit-learn + pandas
Deployment	Docker / Docker Compose
State Mgmt	Provider (`ChangeNotifier`)
Charts	fl_chart
Auth	JWT (access + refresh tokens)

Architecture

```
┌─────────────────────────────────────────────┐
│              Flutter App (Dart)             │
│   Screens → Providers → Services → ApiClient │
└──────────────────┬──────────────────────────┘
                   │  REST (JSON over HTTP)
                   ▼
┌─────────────────────────────────────────────┐
│            FastAPI Backend (Python)         │
│  ┌──────┐  ┌──────────┐  ┌──────────────┐   │
│  │ Auth │  │   CRUD   │  │ ML Pipeline  │   │
│  │ JWT  │  │ Endpoints│  │ categorizer  │   │
│  └──┬───┘  └────┬─────┘  │ budget gen   │   │
│     │           │        │  recommender │   │
│     ▼           ▼        └──────┬───────┘   │
│        PostgreSQL               │           │
│        (SQLAlchemy)     joblib model files  │
└─────────────────────────────────────────────┘
```

Flutter App Structure

```
lib/
├── main.dart / app.dart        # Entry point, MultiProvider, routing
├── config/                     # Theme, colors, constants, routes
├── models/                     # User, Transaction, Budget, Goal, Recommendation
├── providers/                  # Auth, Transaction, Budget, Goal (ChangeNotifier)
├── services/                   # ApiClient + feature services (HTTP layer)
├── screens/                    # Auth, Home, Transactions, Budget,
│                               #   Goals, Analytics, Recommendations,
│                               #   Settings, Account
├── widgets/                    # SummaryCard, TransactionTile, CategoryCard,
│                               #   GoalProgressCard, LoadingOverlay
└── utils/                      # Validators, Formatters, Categories
```

Current State

What exists (~10–15% complete)

✅ Folder structure & route wiring (11 named routes)
✅ Home screen shell with bottom nav (5 tabs)
✅ 5 model classes (fields only — no serialization)
✅ 4 provider stubs extending `ChangeNotifier`
✅ 6 service stubs (all empty TODOs)
✅ 8 spending categories defined
✅ Light/dark theme with Material 3
✅ 17 test cases (14 pre-written TDD-style, only 3 pass)

Current State (cont.)

What's missing

- ❌ Model serialization (`fromJson` / `toJson`)
- ❌ `provider` package not in pubspec — providers not wired in
- ❌ API client implementation
- ❌ All service logic
- ❌ All provider logic
- ❌ All screen UIs (placeholder text only)
- ❌ Validators & formatters
- ❌ Charts / analytics
- ❌ Entire FastAPI backend
- ❌ Entire ML pipeline

Implementation Roadmap

Phase	Weeks	Dates	Focus
1	1–2	Feb 18 – Mar 3	Flutter Foundation + Mock Data
2	3–4	Mar 4 – Mar 17	FastAPI Backend + PostgreSQL
3	5–7	Mar 18 – Apr 7	**ML Pipeline (Core)**
4	8–9	Apr 8 – Apr 21	Full Integration
5	10	Apr 22 – May 1	Polish & Demo Prep

> **Strategy:** Mock-first Flutter dev → Backend → ML depth → Connect → Ship

Flutter Foundation

Weeks 1–2 · Feb 18 – Mar 3

Phase 1 — Tasks

1. **Add dependencies** — `provider`, `fl_chart`, `flutter_secure_storage`
2. **Model serialization** — `fromJson()`, `toJson()`, `copyWith()` on all 5 models; add `progressPercent` / `isCompleted` to `Goal`
3. **Validators** — `Validators.email()`, `.password()`, `.amount()`
4. **Formatters** — currency, date, percentage using `intl`
5. **MockApiClient** — returns sample data for all endpoints; toggle mock vs real
6. **Wire providers** — `MultiProvider` in `app.dart`
7. **Implement providers** — state fields, loading states, methods calling services

Phase 1 — Tasks (cont.)

8. **Auth screens** — login & register forms with validation, auth-gated routing
9. **Transaction screens** — list view consuming provider, add form (amount, category, description, date)
10. **Complete widgets** — `CategoryCard` with progress bar, `GoalProgressCard` with progress + target

✅ Verification

- `flutter test` → all 17 tests pass
- App runs on emulator with mock data
- Can navigate all screens, add mock transactions

FastAPI Backend

Weeks 3–4 · Mar 4 – Mar 17

Phase 2 — Tasks

1. **Scaffold `backend/`** — `main.py`, `requirements.txt`, `Dockerfile`, `.env`
2. **Database models** — SQLAlchemy: `User`, `Transaction`, `Budget`, `Goal`, `Recommendation`
3. **Auth system** — JWT: `/auth/register`, `/auth/login`, `/auth/me`, bcrypt passwords
4. **CRUD endpoints** — full REST for transactions, budgets, goals, recommendations (with filtering)
5. **Seed data script** — 6+ months of realistic spending data across 8 categories
6. **Docker Compose** — `api` + `postgres` services for local dev

✅ Verification

- `pytest` passes on all endpoints
- FastAPI Swagger UI (`/docs`) shows all routes

ML Pipeline ⭐

Weeks 5–7 · Mar 18 – Apr 7

This is the capstone centerpiece.

Phase 3 — Transaction Categorizer

Goal: Auto-classify transactions by description + amount

Step	Detail
Feature engineering	Tokenize descriptions, amount ranges, time features
Model	TF-IDF + Random Forest / Gradient Boosting
Target	**≥ 80% accuracy**
Endpoint	`POST /api/v1/ml/categorize`
Integration	Auto-categorize on `POST /transactions` when category omitted

```
backend/ml/categorizer.py
```

Phase 3 — Budget Generator

Goal: Analyze spending history → suggest per-category budgets

Step	Detail
Input	3+ months of transaction history
Method	Rolling averages, percentile limits, clustering
Output	Per-category budget allocations with reasoning
Endpoint	`POST /api/v1/ml/generate-budget`

```
backend/ml/budget_generator.py
```

Phase 3 — Savings Recommender

Goal: Detect wasteful patterns → generate actionable recommendations

Step	Detail
Analysis	Anomaly detection, recurring pattern analysis, category comparison
Output	≥ **3 recommendations** per analysis with `potentialSavings`
Endpoint	`GET /api/v1/recommendations`

```
backend/ml/recommender.py
```

Phase 3 — Model Evaluation

- Classification report for categorizer (precision, recall, F1)
- MAE for budget predictions
- Save models with `joblib`, load on API startup
- Evaluation script: `backend/ml/evaluate.py`
- Document metrics for capstone report

✅ Verification

- Categorizer ≥ 80% accuracy
- Budget generator produces reasonable allocations
- Recommender yields 3+ actionable recommendations
- All models persist and reload correctly

Integration

Weeks 8–9 · Apr 8 – Apr 21

Phase 4 — Tasks

1. **Real `ApiClient`** — HTTP methods, JWT storage (`flutter_secure_storage`), token refresh, error handling
2. **Implement all 5 services** — replace stubs with real HTTP calls
3. **Build remaining screens:**
    - **Budget** — per-category progress bars, "Generate AI Budget" button
    - **Goals** — goal cards with progress, add/edit forms
    - **Analytics** — pie chart (breakdown), line chart (trends), bar chart (comparison)
    - **Recommendations** — ML-generated cards with savings amounts
    - **Settings** — dark mode toggle, notifications, logout
4. **Home dashboard** — summary cards, recent transactions, top recommendation
5. **Error & loading states** — empty states, snackbars, pull-to-refresh

Phase 4 — Key Screens

Home Dashboard

- Total spending summary
- Budget status card
- Savings potential
- Recent transactions
- Top recommendation

Analytics

- Spending breakdown (pie)
- Spending over time (line)
- Category comparison (bar)

Budget

- Per-category spent vs. limit
- Progress bars
- **"Generate AI Budget"** button

Recommendations

- ML-generated cards
- Potential savings amounts
- Category + actionable title

Polish & Demo Prep

Week 10 · Apr 22 – May 1

Phase 5 — Tasks

1. **Testing sweep** — all existing tests pass + new integration tests (login → add transaction → budget update → recommendations)
2. **Edge cases** — offline handling, form validation, empty data states
3. **Demo data** — curated seed data showcasing clear spending patterns, meaningful ML output
4. **Documentation** — setup instructions, API docs (auto from FastAPI `/docs`), ML metrics, screenshots
5. **One-command demo** — `docker-compose up` starts API + DB + seeds data

✅ Verification

- Full flow: register → login → transactions → analytics → AI budget → recommendations

Key Decisions

Decision	Choice	Why
Mock-first Flutter	✅	Frontend + backend progress independently
Monorepo	✅	`backend/` alongside Flutter — simpler for capstone
Provider over Riverpod	✅	Already scaffolded with `ChangeNotifier`
fl_chart over Syncfusion	✅	Free, sufficient for spending charts
JWT over sessions	✅	Stateless, clean with Flutter `http`
ML in Python (FastAPI)	✅	scikit-learn ecosystem, not embedded in Dart

Risk Mitigation

Risk	Mitigation
ML accuracy < 80%	Start training early (Phase 3); try multiple models; augment training data
Backend delays	Mock-first Flutter means frontend never blocks on API
Scope creep	ML depth prioritized over UI polish — cut features from UI, not ML
Integration issues	Phase 4 is 2 full weeks for connecting pieces
Demo failures	Docker Compose for reproducible demo; seed script for guaranteed data

Week	Milestone
2 (Mar 3)	Flutter app runs with mock data, all tests pass
4 (Mar 17)	Backend API complete, DB seeded, Docker running
7 (Apr 7)	**All 3 ML models trained & serving predictions**
9 (Apr 21)	Flutter ↔️ Backend fully integrated
10 (May 1)	**Demo-ready** 🎯