Budget Tracker — Project Plan (Expo React Native + Python FastAPI)

# Project Name

SmartTracker

# The Team

Ahmet Sahiner  
Victoria Turner  
Direala Kalaiti  
Nancy Padilla

# The Clients Anton Georgiev - Manager Lisa - Future customer Overview

The project is to develop a personal budget tracker that connects to a user’s bank, fetches recent transactions, categorizes them, allows users to add manual expenses/income, and sends alerts when budgets or thresholds are exceeded. The tracker will support both web and mobile platforms, with a focus on usability and privacy.   
The overall goal is to empower our clients to take control over their finances with a minimal effort.

The project consists of three main parts:

1. Defining what the product does, how the user will interact with it, and how the data will be handled.
2. The development plan and process by which the product will be built and how we will ensure quality.
3. Developing a risk management strategy to handle sensitive data.

**User Stories**

**User Story 1 - Anton**

Currently, the client is using Microsoft Excel to keep track of his income and expenses. It has been difficult for him to keep track because he needs to manually add the expenses. This becomes an issue when he becomes busy, and his tracking is inconsistent.

| US1 | As a user, I want to sync my bank transactions so I do not have to enter every expense manually |
| --- | --- |
| US2 | As a user, I want to be able to manually add expenses or income that are not reflected on my bank account |
| US3 | As a user, I want the expenses to be categorized so I know where the money is going |
| US4 | As a user, I want to receive notifications when I am spending over in some categories |
| US5 | As a user, I want to view monthly summaries to see if there is a pattern of high spending throughout the months. |
| US6 | As a user, I want to access the dashboard on the phone when I am on the go. |

**User Story 2 - Lisa**

This client checks bank and card balances/statements, when bills are due or a significant expense arises, without consistent quantifiable financial goals. The client would like to keep closer track of finances but feels analysis paralysis when looking over documents. She would benefit from recommendations based on defined goals, user credit profile, and current financial market status.

| US1 | As a user, I want to define my financial goals and quickly identify budgeting strategies to get me there. |
| --- | --- |
| US2 | As a user, I want to be made aware of wasteful spending and potential ways to lower current expenses. |
| US3 | As a user, I want to separate my expenses by business and personal expenses. |
| US4 | As a user, I want to have a quick idea of my tax responsibility for the year. |
| US5 | As a user, I would like to keep track of my loan payoff info and carve out a goal and plan for budget friendly early payoff insights. |
| US6 | As a user, I would like to keep track of receipt data (in case of lost documentation) and confirmation numbers, to contest any payment disputes. |

# End-to-end user flow

**1. Onboarding & Sign-in —** Create an account or sign in with email/password. Optional Sign in with Apple (iOS) and Google Sign-In (Android). “Forgot password” and “Change password” are included.

**2. Connect a bank —** Tap Connect Bank ==> secure third-party bank connector flow ==> user authenticates there and is returned to our app with permission granted.

**3. First sync —** Recent transactions are fetched and shown on the Home and Transactions screens with automatic categorization (editable by the user).

**4. Daily/periodic sync —** Transactions refresh daily by default; users can choose weekly/manual refresh. Pull-to-refresh is available.

**5. Manual entries —** Add cash expenses or income from the Add screen (with quick presets and optional note/receipt photo later).

**6. Budgets & alerts —** Users set rules like “Food > $500 this month” or “Any purchase at X > $100.” When conditions are met, the app sends a push notification.

**7. Review & adjust —** Users review monthly charts, edit categories, adjust alerts/budgets, and manage notification preferences.

# Mobile app structure (tabs & key screens)

* **Auth Flow (not a tab):** Sign In/Up, Forgot Password, Reset/Change Password.
* **Tab 1 — Home (Dashboard):** Month-to-date totals, category snapshot, “Connect Bank” CTA if not linked; light, tasteful animations on summary cards.
* **Tab 2 — Transactions:** Full list (bank + manual), filters (date/category/amount), edit category, add notes, mark recurring, keep track of transaction/confirmation numbers, to support returns and disputes.
* **Tab 3 — Add:** Fast manual add for expense/income with common presets.
* **Tab 4 — Account:** Profile, connected banks, notification & sync preferences, “Sync now,” change password, sign out, delete account.
* **Insights (from Home/Transactions):** Monthly category report, top merchants, trends.
* **Tab 5 — Goals:** Users name and define financial goals, including pay off loan or save dollar amount by designated date, then system recommends goals based on cash flow analysis. If expense cutting is needed, the user chooses which categories then specific merchants or habits. If the goal is currently unattainable, maximum recommendation is made.

# Alerts & rules (plain-English logic)

Rules behave like simple if/then checks:

* If a single purchase at a given merchant exceeds a threshold ==> notify.
* Identify and alert user when potential duplicate transactions are detected.
* If total spending in a category exceeds a monthly limit ==> notify.

1. Function is called if and only if expense is reported AND limit is set for indicated category.

We’ll ship a few defaults that users can enable/disable and customize.

# Bank connectivity

Preferred: use a well-known bank connector with a sandbox (standard practice; the app never handles bank passwords).

Fallback for demo: allow CSV/OFX/XML upload exported from the bank website. The app processes the file and treats the transactions as fetched.

Both paths feed the same in-app views and features.

# Data handling & privacy (commitments)

* We do not store bank credentials; the connector handles authentication.
* Only data required for features is retained; users can revoke access and delete their account/data.
* Notification settings and alert thresholds are user-controlled.

# Milestones & scope

**Sprint 1 (MVP)**

* Auth (email/password + password reset), basic settings.
* Connect Bank (sandbox) or CSV/OFX import.
* Transactions list, basic auto-categorization, and manual add including note field with picture capture or upload for miscellaneous data (transaction IDs, payment confirmation numbers, undefined categories, money loaned, etc).
* Business and personal expense manual separation
* Home dashboard with month-to-date totals.
* Daily auto-sync and “Sync now.”

**Sprint 2 (Insights & Alerts)**

* Monthly category report and simple trends.
* Create/edit alert rules; push notifications when thresholds are exceeded.
* Notification and sync-schedule preferences.
* User-defined financial goals, budgeting tips and investment % recommendations
* Loan/mortgage payoff data, user defined payoff date goal and payment schedule builder
* Credit score integration estimate qualified loan/refinance terms, market and credit score based update alerts

**Sprint 3 (Polish & Stretch)**

* Annual category report, income tax info
* Projected income
* More categories/presets, improved empty states, accessibility passes.
* Optional receipt photo on manual entries.
* If time remains, explore statement-image parsing as a non-critical stretch.

# Testing & Quality

* Short test plan per sprint: happy-path flows, edge cases (no bank, no data, offline), and alert rule scenarios.
* Lightweight usability checks on navigation, empty states, and error messages.
* Demo script for class presentations.

# Risk Analysis There could be potential issues in this project because the tracker will use private information. We will need to address risks like unauthorized access of bank statements or user credentials by encrypting data, use secure authentication and audit access logs and permissions.

Also, in the event of bank connectivity issues like being down or failed authentication, we can use CSV/OFX/XML files to upload manually. This will help in continuing tracking when the information cannot be sync automatically.

Another potential challenge is wrong categorization of transactions and as a fallback, users should be able to manually make the change to the correct categories and eventually to include AI to improve categorization.

We could also have the issue that the users might find it hard to use or feel that there is no value to them so we will have to make it very intuitive to create a positive user experience.

**Visibility Plan**We will ensure transparency and clear communication with our clients throughout the project lifecycle.

We will communicate via email or text message when we have completed a task, what issues we had and what are our next steps. Since we will be using Trello, we will keep it up to date for the clients to view it to check on progress.

# Collaboration & ownership (kept flexible)

We won’t pre-assign roles here. Feature ownership and pairing will be decided together during sprint planning and can rotate as needed.

Anyone can take an end-to-end slice (mobile screen + supporting API) or focus on a shared area (UI polish, data sync, alerts).

Documentation, diagrams, and requirements are shared responsibilities.

# Note on technology alignment (kind and clear)

To keep the project simple and finish reliably, it’s best not to mix two backend stacks. Using both Java and Python increases setup, testing, and hand-off overhead. For this class project, a single backend is the most practical path. Recommendation: Proceed with Python + FastAPI for the backend, together with Expo React Native for the mobile app. If everyone plans along these lines, our scope stays clear and the deliverables are easier to demonstrate.

# Agile & Kanban Implementation — Why and How (from my perspective)

## Why this model

We want a lightweight process that helps us ship working increments quickly, keep scope small, and reduce handoff overhead. Our app naturally fits vertical slices (mobile screen + API endpoint), and our team has mixed experience levels. To make collaboration smoother, We prefer a single backend stack (Python + FastAPI) and a mobile client (Expo React Native). This minimizes context switches and lets us iterate faster.

Agile practices give us cadence and clarity; Kanban policies help us control work-in-progress and keep the flow steady. Together, they support predictable progress for our class deadlines while staying flexible.

## Agile plan (Scrum-style cadence)

Cadence: weekly or bi‑weekly iterations aligned with class milestones. Every iteration should end with a demoable build.

Core ceremonies:

* Sprint Planning (choose a thin vertical slice; write acceptance criteria; agree on Definition of Done).
* Daily Check‑In (max 15 minutes; blockers and next steps).
* Sprint Review/Demo (show working software, even if small).
* Retrospective (what to keep, start, stop; 20–30 minutes).

Backlog & stories:

* User stories follow INVEST and include clear acceptance criteria.
* Prefer vertical slices (e.g., “Connect Bank E2E” ==> screen, API, storage, and a small test).
* Estimate simply (T‑shirt sizes or story points) to set realistic iteration goals.

Quality & Definition of Done (DoD):

* Code reviewed by at least one teammate.
* Basic tests run and pass; manual happy‑path verified on a device or emulator.
* No crashing errors; essential empty‑state and error messages present.
* Updated README/notes where relevant; demo script ready.

## Kanban plan (continuous flow policies)

We’ll visualize the work on a single board and limit WIP to keep cycle times short. This helps us finish what we start and makes bottlenecks visible early.

Board and policies:

* Columns: Backlog ==> Ready ==> In Progress ==> Code Review ==> Test ==> Done.
* WIP limits (adjustable): In Progress=3, Code Review=2, Test=2.
* Blocked work is clearly marked with a reason; we prioritize unblocking before starting new items.
* Small pull requests; trunk‑based development where possible.

Metrics & feedback:

* Track cycle time and throughput; glance at a weekly cumulative flow to spot queues.
* Hold a short weekly demo, even if it's not the end of a sprint, and always show something real.

## How we will apply it day‑to‑day

* Work intake: new ideas go to Backlog. We triage weekly, promote a few items to Ready, and pull into work only when capacity is available.
* Scopes: keep MVP small (Auth ==> Connect Bank/CSV ==> Transactions ==> Manual Add ==> Home totals). Insights and Alerts come next.
* Ownership: roles remain flexible. We can pair up and rotate. No hard split by language; anyone can take an end‑to‑end slice.
* Communication: short daily check‑ins (async is fine) and a single thread for blockers. Decisions are summarized on the board.
* Risk management: one backend (FastAPI) to reduce integration risk; OCR/statement‑image parsing remains a non‑critical stretch goal.

## Tools and working agreements

* GitHub Issues/Projects for the board; labels: story, bug, chore, blocked, demo.
* Milestones align with iterations/class deadlines; each milestone has a demo checklist.
* PR template includes: scope, screenshots (if UI), test notes, and checklist against DoD.
* Notification etiquette: mention teammates only when action is required; keep discussions on the issue card for traceability.

## Note on technology choice (kind and clear)

To keep our delivery predictable, we should stay with a single backend (Python + FastAPI). Mixing two backends (Java + Python) adds setup and testing overhead and slows handoffs. We’ll still benefit from everyone’s experience in planning, testing, and documentation, but our implementation will be simpler and easier to demo.

# Conclusion This plan supports the development of an user friendly web based and a mobile app while giving the team a chance to learn and be collaborative. By adopting an Agile method, Kanban, we have flexibility to make changes at any point during an iteration.