



The goal to prepare initial requirements files for creating an application called "Personal Loan Calculator".

Personal Loan Calculator is a simple web application that helps users estimate monthly loan payments based on the loan amount, APR, and repayment period. It allows users to adjust interest rates with a slider, compare different loan amounts and terms, and instantly see how these changes affect their monthly payments.

I will be using GitHub Copilot and the Context Engineering approach.

Run a Q&A session (suggest possible options for each question) to help clarify and define these elements. Move to Step 2 only after all questions are answered.

## ## Step 1: Requirements Clarification

Conduct an initial research phase to define:

- Initial requirements
- Technology stack

- Architectural considerations

Use an example UI of a similar application (screenshot is attached to this message).

Replicate the UI/UX design and idea from this example, all input fields, sliders, and buttons.

Based on the provided design, also cover some relevant questions during the Q&A.

## ## Step 2: Implementation Plan Preparation

Once the requirements are clarified, generate the files needed for working with the GitHub Copilot Agent to initialize the implementation plan:

- ARCHITECTURE.md : (max 2 page) file that describes the overall architecture of the project.
- PRODUCT.md : (max 2 page) file that describes the product functionality of the project.
- CONTRIBUTING.md : (max 1 page) file that describes developer guidelines and best practices for contributing to the project.
- custom-instructions.md : file defines persistent, project-specific rules and preferences that guide GitHub Copilot's behavior, ensuring consistent code style, architecture, and development practices across AI-assisted workflows.

## Step 1: Requirements clarification (Q&A)

I will use the attached UI as the reference layout and interactions: left panel has loan amount input + Calculate button, an APR slider with minus/plus controls and a big APR % readout. Right panel shows "Estimated monthly payment" with a 3-option loan-amount selector (ex: \$6k, \$7k, \$8k) and a grid of terms (24/36/48/60 months) with monthly payments for each amount.

Answer the questions below (just copy/paste with your choices). I included options for each, and you can add your own.

## A) Product scope and users

### 1. Target audience

- A1: General consumers (simple, friendly copy)
- A2: Internal demo/prototype
- A3: Partner embed / white-label

### 2. Primary goal of v1

- A1: Quick estimate only (no saving, no accounts)
- A2: Estimate + shareable link to inputs
- A3: Estimate + printable/exportable summary (PDF not required, just print CSS)

### 3. Geography and language

- A1: US only, English only
  - A2: US + Canada, English only
  - A3: Multi-locale later (design for i18n now)
- 

## B) Loan calculation rules (core)

### 4. Monthly payment formula

- B1: Standard amortized payment (principal + interest, fixed APR, fixed term)
- B2: Interest-only payments (not typical for personal loans)
- B3: Let user choose amortized vs interest-only

### 5. APR interpretation

- B1: Nominal APR, compounded monthly (APR/12)
- B2: Effective APR with more precise compounding assumptions
- B3: Keep it simple and document assumptions in UI tooltip

### 6. Rounding and display

- B1: Round to nearest whole dollar
- B2: Show cents (2 decimals)
- B3: Whole dollars in the grid, cents in a details panel

### 7. Edge cases

- B1: If APR is 0%, payment is principal/term
  - B2: For very small APR, still use amortization formula
  - B3: Disallow 0% APR
-

## C) Inputs, ranges, and defaults (match the UI)

### 8. Loan amount input behavior (left input)

- C1: Currency input with commas, accepts typing and paste (ex: 7000, 7,000, \$7,000)
- C2: Digits only, format on blur
- C3: Slider + input (even though screenshot shows only input)

### 9. Loan amount limits

- C1: Min \$1,000, max \$50,000
- C2: Min \$500, max \$25,000
- C3: Configurable constants in code (pick defaults)

### 10. APR slider range (screenshot shows roughly 18% to 35.99%)

- C1: Exactly 18.00% to 35.99%
- C2: 0% to 36% (wider, more flexible)
- C3: Configurable, but default to 18% to 35.99% to match UI

### 11. APR step size

- C1: 0.01% steps (precise, heavier UI)
- C2: 0.1% steps
- C3: 0.25% steps
- C4: 1% steps (with plus/minus)

### 12. Default values on first load

- C1: Amount \$7,000, APR 25%, selected amount column \$7,000
- C2: Amount \$10,000, APR 20%
- C3: Persist last-used values in localStorage

### 13. Term options shown in the grid

- C1: 24, 36, 48, 60 months (match screenshot)
- C2: Add 12 and 72 months too
- C3: Let terms be configurable but default to 24/36/48/60

---

## D) Comparison table on the right (amount tabs + term rows)

### 14. Amount selector (the 3 pills: \$6k, \$7k, \$8k)

- D1: Always show 3 fixed amounts: input-1000, input, input+1000
- D2: Fixed: \$6k, \$7k, \$8k regardless of input
- D3: User-configurable increments (ex: 500, 1000, 2000)

### 15. Grid behavior

- D1: Always show all 3 amount columns, highlight selected amount pill
- D2: Only show selected amount column (simpler)
- D3: Show all, but allow toggling which columns appear

## 16. What triggers recalculation

- D1: "Calculate" button updates the table, slider changes update instantly
- D2: Everything updates instantly (no real need for Calculate)
- D3: Only Calculate triggers any update (strict)

## 17. Extra details

- D1: Grid only, no extra metrics
  - D2: Also show total paid and total interest for the selected term
  - D3: Add an "Amortization schedule" modal for the selected scenario
- 

## E) UX details to replicate

### 18. Slider controls (minus/plus buttons like screenshot)

- E1: Minus/plus adjust by 1 step (from Q11)
- E2: Minus/plus adjust by 1% even if slider is smaller step
- E3: Long-press to accelerate

### 19. Validation and errors

- E1: Inline validation with friendly message, disable Calculate
- E2: Auto-correct to min/max
- E3: Allow empty input and show placeholders until valid

### 20. Mobile responsive behavior

- E1: Stack panels vertically (left input panel above table)
- E2: Keep 2 columns until very small widths
- E3: Provide a simplified mobile view (single selected amount only)

### 21. Accessibility

- E1: WCAG-friendly basics: labels, keyboard slider control, focus states
  - E2: Also add screen-reader announcements for updated payment values
- 

## F) Technology stack (v1)

### 22. Frontend framework

- F1: React + TypeScript + Vite

- F2: Next.js + TypeScript
- F3: Vanilla TS + Web Components (no framework)

## 23. Styling

- F1: CSS Modules
- F2: Tailwind CSS
- F3: Plain CSS with design tokens variables

## 24. Component library

- F1: None (custom to match screenshot)
- F2: Radix UI primitives (unstyled)
- F3: MUI/Chakra (faster, but might diverge from screenshot)

## 25. Testing

- F1: Vitest + React Testing Library
- F2: Jest + RTL
- F3: Minimal tests only for calculation function

## 26. Deployment

- F1: GitHub Pages
  - F2: Vercel
  - F3: Netlify
- 

## G) Architectural considerations

### 27. App structure

- G1: Pure client-side SPA
- G2: SSR (only if using Next.js)
- G3: SPA now, but designed to embed as a widget later

### 28. Configuration

- G1: Hardcoded constants for v1
- G2: Config file (JSON/TS) for ranges, terms, increments
- G3: Environment variables for ranges

### 29. Telemetry/analytics

- G1: None
  - G2: Basic event tracking (input changes, calculate clicks)
  - G3: Add later, but instrument code with hooks
-

## Your turn

Reply with your choices like this (example):

- 1: A1
- 2: A1
- 3: A1

...and so on.

Once all are answered, I will move to Step 2 and generate: ARCHITECTURE.md, PRODUCT.md, CONTRIBUTING.md, custom-instructions.md.



1: A2  
2: A1  
3: A1  
4: B1  
5: B1  
6: B1  
7: B1  
8: C1  
9: C1 (min \$1500, max \$100000)  
10: C2  
11: C4  
12: C1  
13: C1  
14: D1  
15: D1  
16: D1  
17: D2  
18: E1  
19: E1  
20: E1  
21: E1  
22: F1  
23: F2  
24: F3  
25: F1  
26: F1

27: G1

28: G2

29: G1

## Step 2: Implementation plan preparation files

### ARCHITECTURE.md

md

#### # Personal Loan Calculator - Architecture

##### ## Overview

Personal Loan Calculator is a client-side web app that estimates monthly payments

- Loan amount (currency input)
- APR (slider + minus/plus step controls)
- Repayment terms (24/36/48/60 months)

It replicates the reference UI:

- Left panel: amount input + Calculate button, APR slider with minus/plus, large input field
- Right panel: “Estimated monthly payment” with 3 amount tabs and a grid of month
- Extra details section: total paid and total interest for the selected scenario

Target use: internal demo/prototype (no authentication, no backend).

##### ## Tech stack

- Runtime: Node.js LTS
- Frontend: React + TypeScript + Vite
- Styling/layout: Tailwind CSS
- Component library: MUI (Material UI) for input, buttons, slider, tabs (styled to match)
- Testing: Vitest + React Testing Library
- Lint/format: ESLint + Prettier
- Deploy: GitHub Pages

##### ## Architecture and boundaries

Single-page application with clear separation:

- UI layer (React components): rendering, user interactions
- Domain layer (pure functions): loan math, formatting, validation helpers
- State layer (hooks): app state, derived data, and event handlers

No server dependencies. All computations happen in the browser.

```
## Core domain logic
### Payment calculation

Use standard amortized loan payment:
- monthlyRate = (APR / 100) / 12
- n = termMonths
- P = principal
- payment = P * r * (1 + r)^n / ((1 + r)^n - 1)
```

APR interpretation: nominal APR compounded monthly (APR/12).

Edge case:

- If APR == 0%: payment = P / n

Rounding:

- Monthly payments displayed as whole dollars (rounded to nearest dollar)

#### ### Comparison amounts

The 3 amount tabs are derived from the current input amount:

- A0 = amount - 1000
- A1 = amount
- A2 = amount + 1000

All amounts are clamped to the configured min/max. Duplicates after clamping are

#### ### Recalculation behavior

- Clicking Calculate updates the comparison table based on the current loan amount.
- Changing APR via slider or minus/plus updates the table instantly.
- Changing the amount input does not update the table until Calculate is clicked.

#### ## State model

Suggested state shape (React hook):

- inputAmount: number | null
- committedAmount: number (used for table; updated on Calculate)
- apr: number (0 to 36, step 1 via buttons; slider can still be continuous if desired)
- selectedAmountIndex: 0 | 1 | 2 (tab selection)
- terms: [24, 36, 48, 60]

Derived data:

- comparisonAmounts: number[] from committedAmount

- paymentGrid: Map(term -> Map(amount -> payment))
- selectedScenario: { amount, term } (term chosen via row focus/click)
- selectedDetails: { monthlyPayment, totalPaid, totalInterest }

### **## UI component structure**

Proposed folders:

- src/
  - app/
    - App.tsx
  - components/
    - LoanAmountInput.tsx
    - AprControl.tsx
    - AmountTabs.tsx
    - PaymentTable.tsx
    - SelectedDetails.tsx
  - domain/
    - loanMath.ts
    - currency.ts
    - validation.ts
  - styles/
    - globals.css

Component responsibilities:

- LoanAmountInput: currency input, validation, Calculate button
- AprControl: slider + minus/plus, large APR readout
- AmountTabs: 3 amount pills for selection
- PaymentTable: term rows x amount columns with highlighted column
- SelectedDetails: total paid + total interest for selected scenario

### **## Non-functional requirements**

- Responsiveness: on mobile, stack left panel above right panel
- Accessibility: labels, keyboard navigation, focus states; slider usable with keyboard
- Deterministic math: domain functions unit-tested

### **## Configuration**

Central config file in TypeScript:

- minAmount = 1500
- maxAmount = 100000
- aprMin = 0
- aprMax = 36

```
- aprStepButtons = 1  
- terms = [24, 36, 48, 60]  
- compareDelta = 1000
```

## ## Deployment

### GitHub Pages:

- Build output from Vite
- Base path configured for repo name
- CI workflow runs: install, lint, test, build, deploy

 Copy code

---

## PRODUCT.md

md

## # Personal Loan Calculator - Product

### ## Purpose

A lightweight internal demo web app for estimating personal loan monthly payments

### ## Key features

#### 1) Loan amount input

- Currency input that accepts: 7000, 7,000, \$7,000
- Limits: min \$1,500 and max \$100,000
- Inline validation: show message and disable Calculate when invalid

#### 2) APR adjustment

- Range: 0% to 36%
- Slider for quick changes
- Minus and plus buttons adjust APR by 1% per click
- Large APR value displayed (example: 25%)

#### 3) Monthly payment comparison table

- Header: "Estimated monthly payment"
- Amount selector tabs: three values derived from the committed amount:
  - committed - 1000, committed, committed + 1000
- Terms shown as rows: 24, 36, 48, 60 months
- Grid shows monthly payments for each term and amount
- Selected amount tab highlights the corresponding column

#### 4) Calculation behavior

- Calculate button commits the current amount and updates the table
- APR changes update results instantly (no Calculate required)
- Display rounding: monthly payments are whole dollars

#### 5) Selected scenario details (extra)

For the currently selected amount tab and an active term selection:

- Monthly payment (whole dollars)
- Total paid = monthlyPayment \* termMonths
- Total interest = totalPaid - principal

## ChatGPT 5.2



- Default active term: 36 months (or first term) on first load
- Clicking a row sets the active term and updates the details panel

#### ## Assumptions

- Loan is fully amortizing with fixed APR and fixed term
- APR is nominal and applied as APR/12 monthly rate
- If APR is 0%, payment is principal divided by months
- This is an estimator, not a loan offer

Copy code

#### ## Out of scope for v1

- User accounts, saving profiles
- Backend APIs
- Credit checks or eligibility logic
- Amortization schedule export

#### ## Success criteria (v1)

- UI closely matches the reference layout and interaction model
- Results update correctly and consistently with the defined triggers
- Input validation prevents misleading outputs
- Works well on mobile with stacked layout

## CONTRIBUTING.md

md

Copy code

## # Contributing

### ## Prereqs

- Node.js LTS
- npm (or pnpm if the project is configured that way)

### ## Getting started

- Install: npm install
- Run dev server: npm run dev
- Run tests: npm run test
- Lint: npm run lint
- Format: npm run format
- Build: npm run build

### ## Branching and PRs

- Create a feature branch from main: feature/<short-name>
- Keep PRs small and focused
- Include screenshots for UI changes
- Ensure CI is green (lint + test + build)

### ## Coding guidelines

- Use TypeScript strictly, avoid `any`
- Keep domain logic in `src/domain` as pure functions with unit tests
- UI components should be presentational where possible, with logic in hooks
- Prefer composition over deeply nested props drilling

### ## Testing guidelines

- Unit test domain functions (loan math, formatting, validation)
- Component tests for critical interactions:
  - Calculate commits amount
  - APR changes update table instantly
  - Invalid amount disables Calculate and shows error

### ## Accessibility

- Inputs must have labels (visible or aria-label)
- Slider must be keyboard accessible
- Ensure visible focus styles

## ## Commit messages

- Use clear, imperative style: "Add APR control", "Fix rounding for payments"
- 

## custom-instructions.md

md

 Copy code

### # GitHub Copilot Agent - Project Instructions (Persistent)

#### ## Goals

- Replicate the reference UI and interaction model precisely.
- Keep business logic testable and isolated from UI.
- Prefer simple, readable code over abstractions.

#### ## Tech constraints

- React + TypeScript + Vite
- Tailwind CSS for layout and spacing
- MUI for UI primitives (Input, Button, Slider, Tabs) with styling to match the reference UI
- Vitest + React Testing Library for tests

#### ## Architecture rules

- Put pure loan math and formatting in `src/domain/\*`:
  - `loanMath.ts` must export pure functions with no React imports.
  - Include unit tests for payment calculations and edge cases.
- UI components live in `src/components/\*` and should be reusable.
- App composition and state orchestration live in `src/app/App.tsx` (or `src/app/`)

#### ## Product rules

- Amount input:
  - Accept currency-like typing and paste.
  - Validate min \$1500, max \$100000.
  - Show inline error message and disable Calculate when invalid.
- APR:
  - Range 0 to 36.
  - Minus/plus buttons change APR by 1.
  - Slider changes update results instantly.
- Recalculation:
  - Amount changes only update the table after Calculate is clicked.
- Table:

- Show 3 amount columns: committed-1000, committed, committed+1000 (clamped).
- Rows: 24/36/48/60 months.
- Display monthly payments as whole dollars.

### ## Code style

- TypeScript:
  - Use explicit types for public functions.
  - Avoid `any`, prefer `unknown` with narrowing.
- React:
  - Prefer functional components, hooks.
  - Keep derived data computed via memoization where helpful, but do not overuse
- Formatting:
  - Prettier enforced.
  - ESLint warnings should be fixed, not ignored.

### ## Testing expectations

- Add tests for:
  - `calculateMonthlyPayment` for typical APR, 0% APR, and rounding behavior.
  - Amount clamping and compare amounts generation.
  - Calculate-button commit behavior versus instant APR updates.

### ## Deliverables order

- 1) Domain functions + unit tests
- 2) UI skeleton matching layout
- 3) Wiring state and interactions
- 4) Styling polish to match reference
- 5) Accessibility pass

If you want, I can also include a minimal `README.md` and a GitHub Pages workflow YAML that matches the architecture above.



Yes please, generate an initial `README.md` file also

Here is a clean, initial `README.md` you can drop straight into the repo. It is aligned with the requirements, architecture, and Copilot instructions we defined.

md

 Copy code

## # Personal Loan Calculator

A simple, client-side web application for estimating monthly payments on a personal loan. Built as an internal demo/prototype with a clean UI and instant feedback when adjusting inputs.

### ## Overview

The Personal Loan Calculator helps users understand how loan amount, APR, and repayment period affect monthly payments.

Users can:

- Enter a desired loan amount
- Adjust APR using a slider and plus/minus controls
- Compare monthly payments across multiple loan terms
- See total paid and total interest for a selected scenario

All calculations run entirely in the browser. No backend or user accounts are required.

### ## Features

- Currency-aware loan amount input
- APR slider with step controls
- Standard amortized loan calculations
- Comparison table with:
  - 3 loan amount variants
  - 24, 36, 48, and 60 month terms
- Instant updates when APR changes
- Explicit Calculate action for committing loan amount changes
- Responsive layout for desktop and mobile
- Accessible inputs and keyboard-friendly controls

### ## Tech Stack

- React
- TypeScript
- Vite
- Tailwind CSS
- Material UI (unstyled primitives, customized styling)
- Vitest + React Testing Library

### ## Getting Started

### ### Prerequisites

- Node.js (LTS)
- npm

### ### Install dependencies

```
```bash
npm install
```

## Run locally

bash

 Copy code

```
npm run dev
```

The app will be available at:

arduino

 Copy code

```
http://localhost:5173
```

## Run tests

bash

 Copy code

```
npm run test
```

## Lint and format

bash

 Copy code

```
npm run lint
```

```
npm run format
```

## Build for production

bash

 Copy code

```
npm run build
```

## Project Structure

pgsql

 Copy code

```
src/
  app/          Application composition and state
  components/   UI components
  domain/       Pure loan calculation and formatting logic
  styles/        Global styles
```

## Loan Calculation Assumptions

- Fixed-rate, fully amortizing personal loan
- Nominal APR, compounded monthly (APR / 12)
- Monthly payments rounded to the nearest whole dollar
- If APR is 0%, payment equals principal divided by term

This tool provides estimates only and does not represent a loan offer.

## Deployment

The project is designed to be deployed to GitHub Pages using a Vite build output.

## Contributing

See `CONTRIBUTING.md` for development guidelines and best practices.

## License

Internal use only.

javascript

 Copy code

```
If you want, next I can:
- Generate a GitHub Pages workflow (`.github/workflows/deploy.yml`)
- Generate the initial Vite + React + Tailwind project skeleton
- Create the first domain function (`calculateMonthlyPayment`) with tests
- Break this into Copilot-friendly implementation tasks
```



