**Part 2: Design Alternatives**

**Project Description** This project is a multi-functional calculator application designed to support various real-world computational needs such as number conversions, loan computations, unit measurements, and more. It aims to assist students, professionals, and everyday users in performing essential calculations and conversions quickly and accurately through a mobile-friendly interface. By providing multiple specialized calculators in one unified app, the project simplifies access to tools that would otherwise require separate apps or manual methods, making it a practical companion for academic, personal, and work-related tasks.

**Requirements Summary**

The multi-calculator application must fulfill several core functional and non-functional requirements to ensure usability, flexibility, and efficiency for its intended users.

**Functional Requirements:**

**Modular Calculators** -The app must support multiple types of calculators, such as number conversions, loan calculators, unit or measurement converters, and other math-related tools.

**Dynamic Content Loading** -The main page must dynamically load each calculator’s HTML, CSS, and JavaScript when a button is clicked.

**Mobile Compatibility** -The application must be fully operational on Android devices using Capacitor integration via Android Studio.

**User Input Handling** -Calculators must accept and validate user input, display computed results, and handle invalid inputs gracefully.

**Offline Availability** -The app should be usable offline once installed, with all calculators embedded within the package.

**Non-Functional Requirements:**

**Performance** -Calculations and content loading should happen quickly, without noticeable lag, even on low-end mobile devices.

**Scalability** -The system design should allow for the easy addition of new calculators without major structural changes.

**Maintainability** -The codebase should be modular and well-documented to allow updates or bug fixes with minimal effort.

**Accessibility** - The interface must be intuitive and readable, with accessible font sizes, colors, and contrast for all users.

**Design Space**

In the development of our multi-calculator application, several design considerations and challenges arise based on both technical feasibility and user experience.

The most challenging requirement to implement is dynamic content loading in a mobile environment, especially when packaging the app using Capacitor for Android. Unlike standard web environments, ensuring that HTML, CSS, and JS files load correctly within the directory of an Android build can be difficult due to relative path issues and mobile file access limitations. Another complex requirement is adding offline capability while ensuring the app still dynamically loads different calculators modularly.

A key tradeoff involved choosing between embedding all calculators into a single HTML file versus loading them modularly. Embedding them all could simplify compatibility and performance, but at the cost of code maintainability and scalability. Conversely, dynamic modular loading makes the application more flexible and easier to expand, but it introduces complexity when working within Android Studio's build system and Capacitor's web asset packaging.

The easiest tasks to support are those involving static calculations like basic number conversions or unit conversions. These require minimal user input and logic and have straightforward interfaces. In contrast, more complex features like loan calculators or algebraic calculations are harder to implement. These require handling multiple inputs, validating user data, managing dynamic or symbolic expressions, and sometimes generating detailed outputs like amortization tables. Algebraic-related functions, such as solving equations or evaluating expressions, are particularly challenging due to the need for parsing symbolic input and providing accurate, step-by-step results. Tasks involving mobile integration and loading external calculator files dynamically also add to the overall complexity.

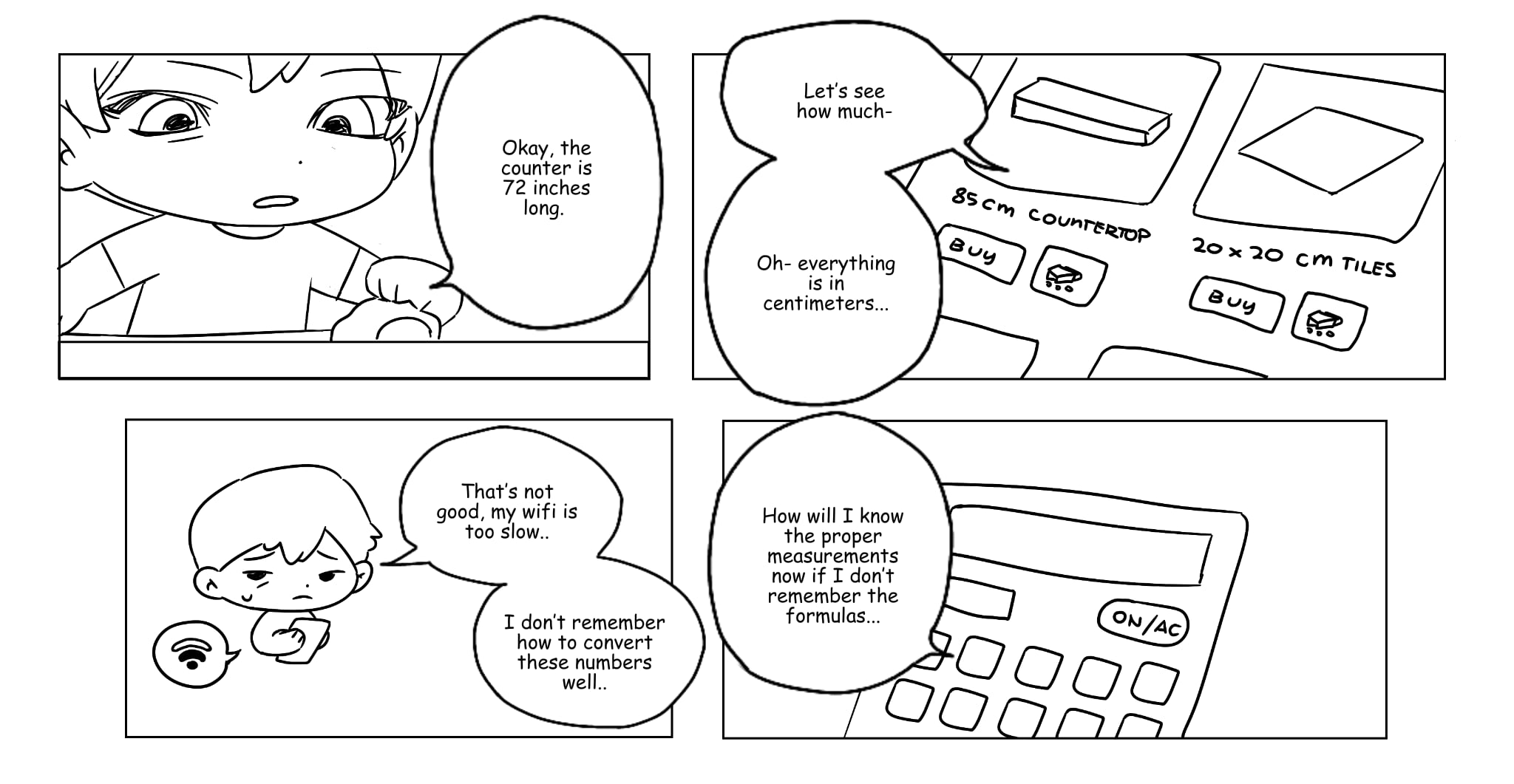
**Design Summary**

In designing our calculator application, we considered several interface approaches. One alternative was to create a single-page app where all calculator tools or functions would exist on one HTML page with dynamic sections shown or hidden based on user input. While this method would have simplified navigation and loading, it could have led to a cluttered interface and slower initial load time. Another option was to create separate calculator pages with full navigation, but this approach added overhead to user experience and mobile optimization. Ultimately, we pursued the modular design of loading external calculator HTML and JS files dynamically into a main container. This choice balanced performance, organization, and flexibility, allowing easy updates or additions without affecting the core layout. We also considered using external frameworks but opted for a lightweight vanilla JavaScript approach to maintain control and simplicity, especially for deployment in mobile environments like Android Studio.

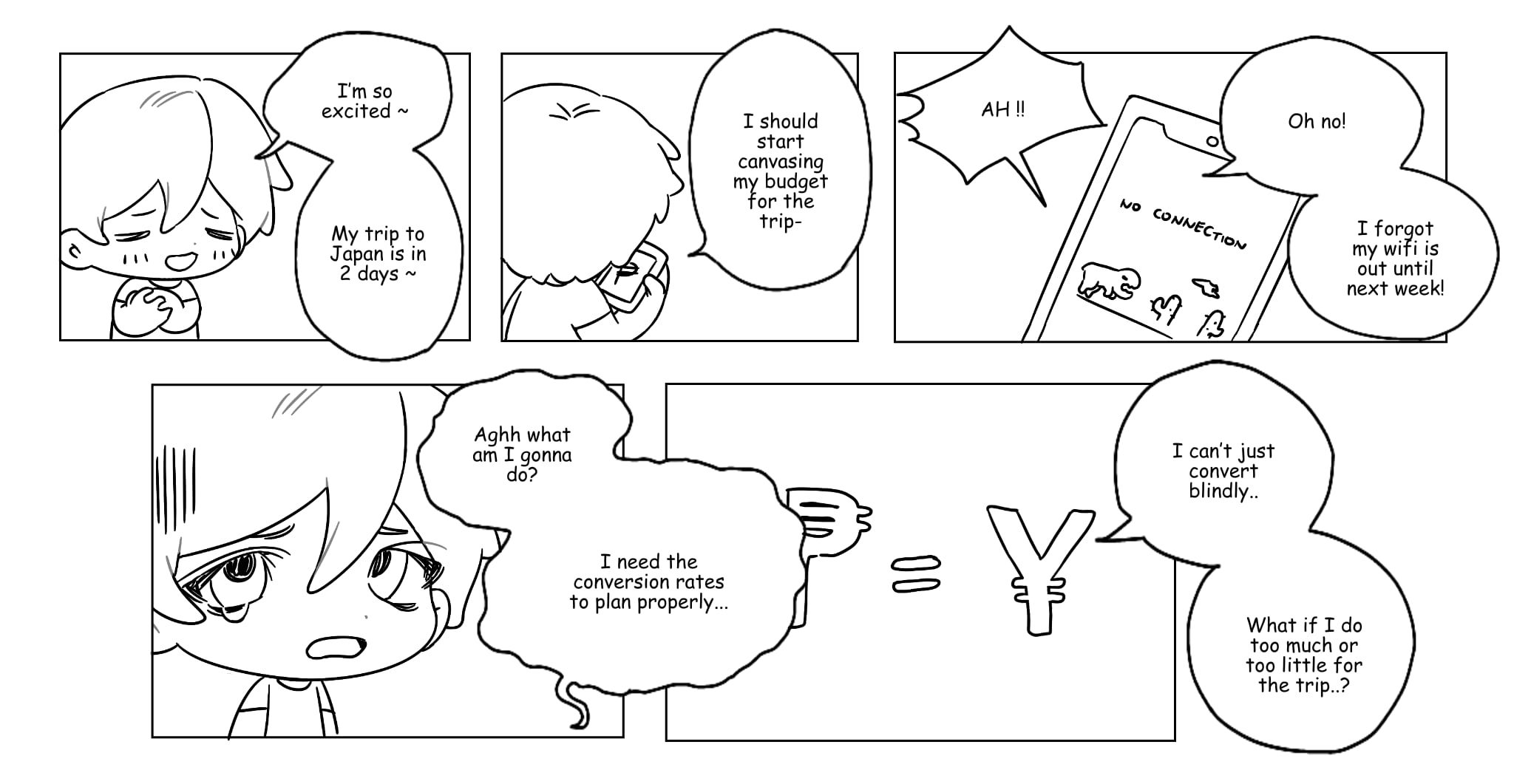
**Scenarios**



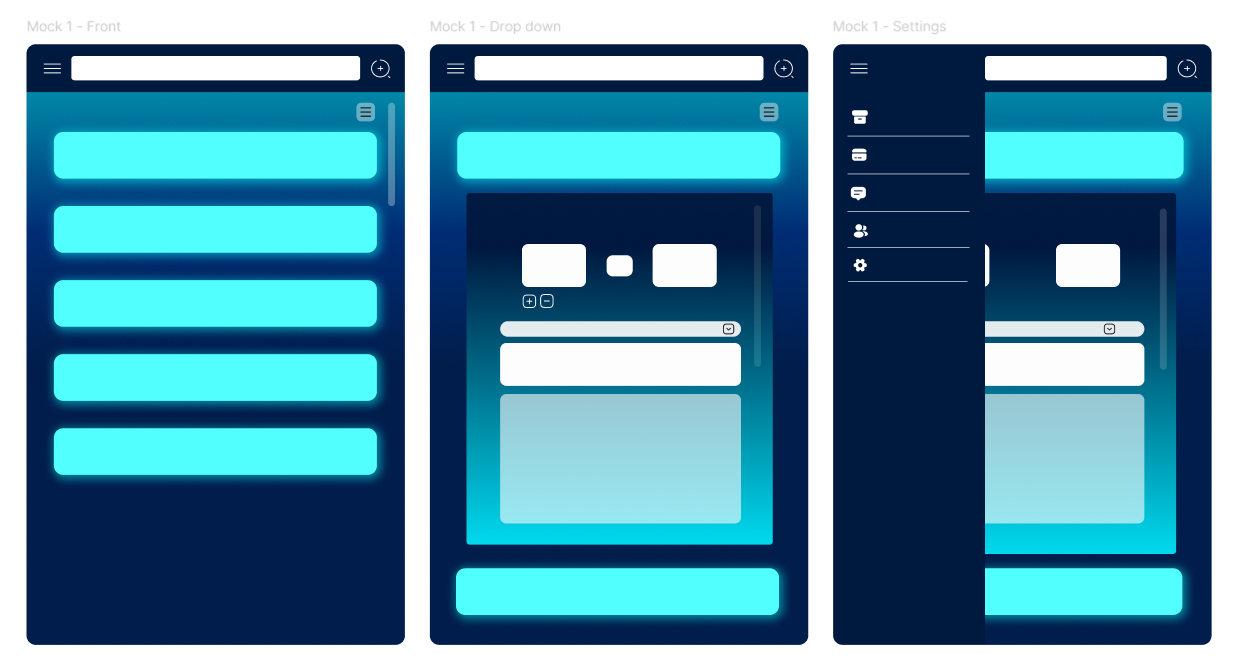
**Scenario 1 Storyboard**



**Scenario 2 Storyboard**



**Scenario 3 Storyboard**

****

**Design 1**

**Overview**

Design 1 uses a bright neon blue theme (can be changed) with a vertical layout. The home screen features large, glowing labeled buttons for each calculator, while the dropdown and settings pages maintain a minimalist design. It relies on icons for navigation and simple screen transitions.

**Scene**

A college student opens the app to calculate her GPA. She taps the glowing GPA calculator button from the main screen, shows the GPA calculator from the dropdown, and inputs her grades. She can get her calculated GPA and can continue scrolling down if she wants something else calculated that is academic related.

**Assessment**

**Advantages:**

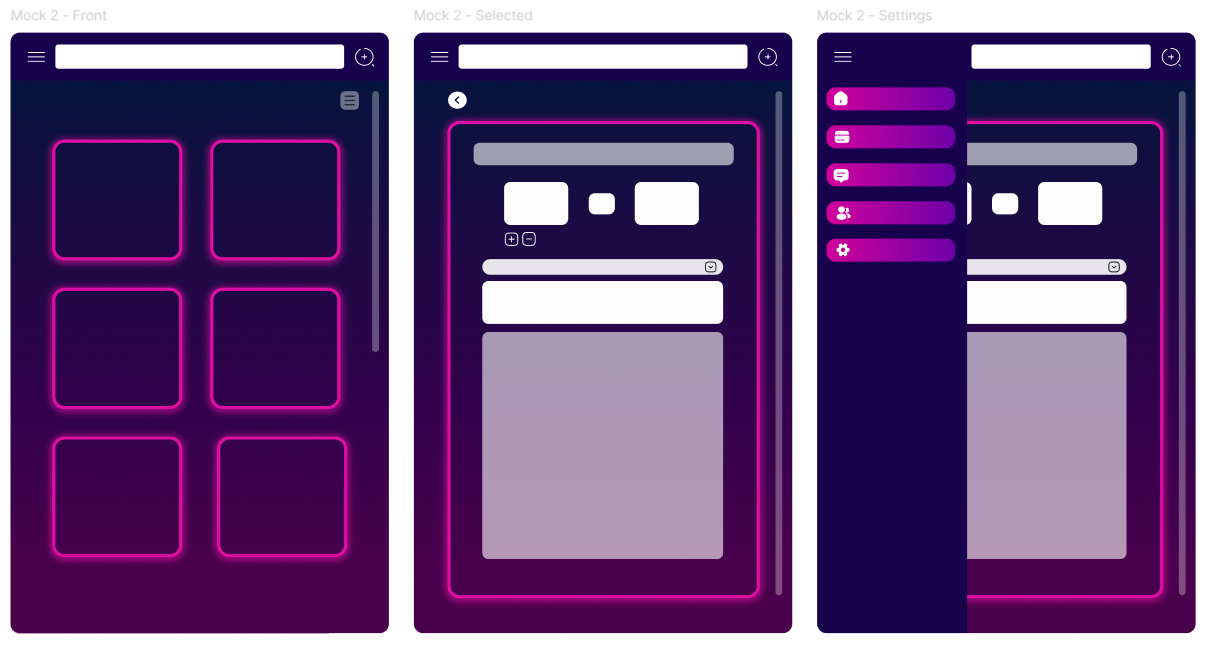
* **Straightforward navigation:** The vertical button list layout is intuitive, especially for users looking for quick access to individual calculators.
* **Dropdown integration:** The dropdown in the selected calculator view supports a clean and compact way to switch between subcategories or calculator variants.
* **Side menu layout:** The slide-out settings menu is familiar and efficient, minimizing distractions from the main workspace.

**Disadvantages:**

* **Limited scalability:** As more calculators are added, the button list could become too long, leading to excessive scrolling. Although it can be solved with the help of the search bar.
* **Minimal information at a glance:** The layout doesn't provide previews or descriptions of calculators, which could make it harder for users to choose the right tool, especially if they’re unsure what the tool does.

**Requirements Met:**

* Supports requirements such as quick navigation, basic tool selection, and lightweight settings access.
* Less effective for information-dense tools, such as those needing multi-step input/output or visual results (e.g., graphs or amortization tables).

****

**Design 2**

**Overview**

Design 2 introduces a boxed layout with a vibrant magenta-to-purple color scheme (can be changed). The front screen presents calculators in a grid view with individual tiles each with a small label, and the side menu uses stylized, rounded buttons.

**Scene**

An engineering student needs to convert meters to feet for his physics assignment. He quickly scrolls through the tile grid, identifies the unit converter by its icon, and taps it. The app transitions to the unit conversion interface, where he selects “Length,” inputs the value in meters, and instantly gets the result in feet.

**Assessment**

**Advantages:**

* **Good discoverability:** The tile/grid layout allows users to visually scan and browse multiple tools at once, which encourages exploration.
* **Structured grouping:** It naturally supports grouping calculators by type or purpose, which aligns well with user expectations (e.g., academic, financial, everyday).
* **Moderate adaptability:** This layout adjusts reasonably well for both mobile and desktop views, especially when paired with collapsible categories.

**Disadvantages:**

* **Can feel cluttered:** If too many tiles are visible, it may overwhelm users or create decision fatigue, particularly if descriptions aren’t shown. Altho
* **Not ideal for form-based tasks:** Detailed inputs like equation solvers or interactive loan forms might not display well in a tile-focused structure.

**Requirements Met:**

* Efficiently supports browsing across a wide range of calculators, especially for users who want to discover what the app offers.
* Struggles slightly with supporting tools that require larger or more complex interfaces, unless paired with a detailed subview.

****

**Design 3**

**Overview**

Design 3 offers a cleaner, split-view layout with a violet and gradient pink aesthetic (can be changed). The layout favors list-based navigation, with calculators arranged vertically and simple dropdown transitions. It balances aesthetic and utility.

**Scene**

A small business owner is reviewing her budget and wants to estimate monthly payments for a business loan. She opens the app and taps the "Loan Calculator" tab at the top. The tab highlights, and the loan calculator interface loads instantly below. She enters the loan amount, interest rate, and term, then presses "Calculate." The results including monthly payments and a summary appear immediately.

**Assessment**

**Advantages:**

* **High clarity and focus:** A clean list format helps users quickly focus on one calculator at a time without distractions.
* **Scales well for complexity:** This structure supports calculators with multi-step inputs or outputs, and can easily accommodate additional features such as save and load input, expandable sections, or help popups.
* **Consistent UX:** The sidebar navigation keeps key sections accessible while leaving most of the screen available for the main tool.

**Disadvantages:**

* **Slower discovery**: Users may not explore new tools as easily since fewer options are visible at once.
* **Repetitive structure**: Without additional visual or interactive distinctions, users may find it harder to tell calculators apart by function or type until they click into them.

**Requirements Met:**

* Best supports advanced calculators (e.g., algebra, financial), task-based flows, and context-sensitive features like dynamic result previews.
* Slightly less optimized for exploration or casual users browsing without a specific tool in mind.

**Requirements changes**

Throughout the design and prototyping stages of our calculator app, several changes to our initial requirements and usability criteria naturally emerged. Originally, we intended to have all calculator tools on a single page for simplicity. However, during prototype making and usability testing, we realized this approach could lead to possible excessive clutter, slow initial load times, and confusion when transitioning between different tools. This led us to introduce modular loading using JavaScript fetch() method, which was not part of our original plan.

We also initially assumed all calculators would share a uniform layout, but during development, we found that certain calculators (e.g., loan and algebraic calculators) needed more space, unique input styles, or specific user instructions. This led to a requirement change to support tool-specific layouts and scripts, making each calculator more tailored and intuitive. These changes arose from a combination of technical feasibility discussions, observed user behavior, and direct feedback, allowing us to better align the app’s design with real-world usage and expectations.