Lumo Savings Calculator — Technical Decision Log

1. Decision-Making Justification

When designing and building the Lumo Savings Calculator, our primary objective was to deliver a modern, fast, scalable, and developer-friendly solution that ensures long-term maintainability and flexibility. To meet these goals, we carefully selected each tool and framework based on industry standards, community support, future-proofing potential, and developer experience.

This document outlines the rationale behind our technical decisions, focusing on how each element of our stack contributes to performance, maintainability, developer velocity, and user satisfaction.

2. Technology Stack Overview

Frontend Framework: React 19.1

React continues to be the most popular and battle-tested frontend library in the web ecosystem. With the release of v19.1, it brings concurrent features, improved performance, and better developer ergonomics. It supports component-based architecture, enhances reusability, and has an enormous ecosystem and developer community behind it. This ensures that Lumo will benefit from ongoing improvements, third-party integrations, and extensive documentation.

Build Tool: Vite

Vite is the fastest build tool available today and has become the de-facto standard for modern frontend projects. It enables:

- Near-instant dev server startup
- Lightning-fast hot module replacement (HMR)
- Out-of-the-box TypeScript and JSX support
- Highly optimized production builds using Rollup under the hood

Vite ensures our dev loop is fast and smooth, while our production bundles are lean and performant.

Styling: Tailwind CSS

Tailwind CSS allows us to build consistent and maintainable styles rapidly using utility-first principles. Key benefits include:

- Faster development with reduced context switching
- Design system enforcement via custom Tailwind config
- Excellent compatibility with component-based architectures

UI component library: MaterialUI

We chose **Material UI** and **Material Icons** for standardized, accessible components and visually polished UI.

Component Routing: React Router

React Router provides robust and flexible routing that integrates seamlessly with React. It supports nested routes, lazy loading, and route-level error handling, allowing us to structure the application in a modular and scalable way.

Type System: TypeScript

TypeScript gives us static typing on top of JavaScript, catching bugs at compile time and greatly improving code maintainability and collaboration. It enhances developer productivity and provides self-documenting code, which is essential in a growing project like Lumo.

State Management: React Context API

For this project's scale, the React Context API offers a lightweight and performant solution for shared state management. It integrates naturally into the React ecosystem, reduces external dependencies, and simplifies debugging.

HTTP Client: Axios

Axios is a well-established HTTP client that supports interceptors, error handling, and cancellation, and is easier to test and extend than `fetch`. It's ideal for handling API communications in a predictable and scalable way.

Charting: Recharts

Recharts provides highly customizable, responsive charts built on React and D3 under the hood. It offers good accessibility and is flexible enough to meet all our visualization needs for presenting savings and financial figures to the end-user.

Code Quality Tools: ESLint + Prettier

To enforce consistency and reduce technical debt, we use ESLint with a custom rule set and Prettier for code formatting. These tools ensure that all code adheres to a defined standard and remains readable and clean across contributors.

Testing Stack: Vitest + React Testing Library

We chose **Vitest** for its seamless integration with Vite and superior performance over Jest. Combined with **React Testing Library (RTL)**, we ensure:

- Fast and reliable test execution
- Testing focused on user behavior, not implementation
- Excellent developer experience and IDE support

This setup guarantees a robust and maintainable test suite that helps catch regressions early.

3. Supported Devices & Browsers

The project is intentionally built with a **modern-only** approach to reduce bloat and technical debt. The following **browsers and devices are supported out-of-the-box**:

✅ Supported Browsers

Browser	Minimum Version
Chrome	110+
Edge (Chromium)	110+
Safari	16+
Firefox	115+
Samsung Internet	20+

We do **not** support legacy browsers such as **Internet Explorer**, **Opera Mini**, or outdated mobile browsers.

✅ Supported Devices

Device Type	Minimum Requirements
Mobile	iOS 16+ / Android 11+
Tablet	iPadOS 16+, modern Android tablets
Desktop	macOS, Windows 10+, Ubuntu 22+

The app is fully **responsive** and adapts to screen sizes from ~360px wide (mobile) up to 1440px+ (desktop HD screens).

4. Future improvements

4.1 Reusable Typography Components

To ensure consistent branding and UI quality across the calculator, we recommend creating a dedicated set of **typography components** that map directly to the text styles defined in Figma.

These components will:

- Guarantee **visual consistency**
- Allow easier **theme adjustments** in the future
- Encourage **semantic clarity** in JSX code

Component	Purpose
`HeadingXL`	Hero titles / prominent page headings
`HeadingM`	Section titles
`HeadingS`	Card titles / smaller headings
`TextL`	Emphasis text / callouts

`TextM`	Body copy
`TextS`	Fine print / secondary descriptions
`TextXS`	Labels, disclaimers, footnotes

Each of these will be styled using **Tailwind** and **MUI's theming** where needed, with class-based control and emotion-styled fallback for edge cases.

4.2 Swapping the JSON File for a Dynamic API Endpoint

The savings calculator currently fetches its permutations data from a static JSON URL (e.g., hosted on a CDN or mock service like `jsonplaceholder.typicode.com`). To maintain flexibility and avoid hardcoding assumptions into the app, we've isolated this logic in a single utility function:

How to Replace the Static URL with a Real API

To migrate from a hosted static JSON file to a real backend API:

1. **Locate this function**:
   `getSavingsPermutationsData()` is the only place where the data-fetching URL lives.
2. **Replace the URL**:
   Swap the `.get()` URL with your actual endpoint:
   `.get('https://api.lumo.energy/v1/savings/permutations')`
3. **Ensure the response shape** matches the expected `SavingsPermutations` type.
4. ✅ **No other changes** are needed in the app — all consumers of this data will remain unaffected as long as the returned format stays consistent.

4.3 Forwarding user selection to external form

Once the form that will receive these user inputs (solar panels, battery Size, number of bedrooms, deposit % and years financed) is created, follow:

1. **Locate this component**: `GetFreeQuoteButton`
2. **Replace the URL**: modify `const url = \`https://externalurl.com?${params.toString()}\`` where `https://externalurl.com` is replaced with the form url.

4.4 Whitelabelling to insert into third party providers

To make your React-based savings calculator app **whitelabel-friendly and embeddable** for third-party companies, you'll need to make it **highly configurable, brand-neutral, and easy to embed**.

- Decouple the Branding

  Use a `WhitelabelConfigContext` and read its content in the applicable parts of the app.

- Enable Style & Theme Customization

  Replace hardcoded styles with CSS vars and tailwind theme

- Support Runtime Configuration
- Build as Embeddable Widget