1. App Summary

Saving for a house down payment is often a slow, daunting process that lacks immediate rewards, causing many people to lose motivation. Our primary users are prospective first-time homebuyers who struggle to stay engaged with traditional, spreadsheet-based saving methods.

To solve this, our app gamifies the financial journey by turning savings milestones into a highly visual and interactive experience. As users save money toward their goal, they unlock pieces of a digital house that is progressively built on their screen. By transforming abstract numbers into tangible progress, our product provides continuous positive reinforcement to make achieving homeownership fun and rewarding.

2. Tech Stack

Frontend Framework and Tooling

Core Framework: React (v19) combined with React DOM.

Build Tool/Bundler: Vite (for fast development and built via @vitejs/plugin-react and @tailwindcss/vite).

Routing: Wouter (a lightweight file-size routing solution for React).

Styling & UI System:

Tailwind CSS (v4) for utility-first styling.
Radix UI primitives (@radix-ui/react-*) for accessible, unstyled interactive components (which strongly suggests the use of shadcn/ui given the presence of class-variance-authority, taildwind-merge, and a `components.json` file).
Framer Motion for animations and transitions.
Lucide React for iconography.

State Management & Data Fetching: TanStack React Query (@tanstack/react-query) for asynchronous state management and fetch request caching.

Forms & Validation: React Hook Form paired with Zod (@hookform/resolvers, zod) for schema-based form validation.

Charts/Visualizations: Recharts for rendering dynamic charts.

Backend Framework

Framework: Express.js (v5) running on Node.js.

Language: TypeScript (executed via tsx during development and compiled with esbuild/tsc for production).

WebSockets: ws for integrated real-time capabilities if applicable.

Database

Database Engine: PostgreSQL (interfaced through the pg driver).

ORM (Object-Relational Mapper): Drizzle ORM (drizzle-orm). Migrations and schema scaffolding are handled by Drizzle Kit.

Schema validation: Drizzle Zod (drizzle-zod) to automatically bridge the database schema and API validation logic.

Authentication

System Check: Local Email/Password authentication.

Libraries: Passport.js utilizing the passport-local strategy.

Session Management: Relies on Express Sessions (express-session) populated in cookies, with server-side session persistence stored in the PostgreSQL database using connect-pg-simple (and an alternative memorystore utility).

External Services or APIs

None are present, the system is fully self-contained. There are no external third-party service SDKs installed (e.g., Stripe for payments, SendGrid for emails, AWS S3 for storage, or Firebase Auth).

3. Architecture Diagram

4. Prerequisites.

To run this project locally, you must install the following software. Ensure these are available in your system's PATH.

- Node.js (v20 or higher recommended)
  - Download & Install: Node.js Official Website
  - Verify installation by running in your terminal: `node -v` and `npm -v`
- PostgreSQL
  - Download & Install: PostgreSQL Official Website
  - During installation on Windows, remember the `postgres` user password you set.
  - Make sure `psql` (the command-line tool) is added to your system PATH (usually `C:\\Program Files\\PostgreSQL\\<version>\\bin` or `C:\\Program Files\\PostgreSQL\\<version>\\bin`).
  - Verify installation by running in your terminal: `psql -V`
- Git
  - Download & Install: Git Official Website
  - Verify installation by running in your terminal: `git --version`

5. Installation and Setup

Follow these steps to get the project running locally:

Step 1: Install Dependencies
Open your terminal in the root folder of the project (`Design-Prototype`) and run the following command to install all necessary Node modules:
npm install
Step 2: Configure Environment Variables

1. Duplicate the `.env.example` file and rename it to `.env`.
2. Open the new `.env` file and update the `DATABASE_URL` with your local PostgreSQL credentials.
   For example, if your PostgreSQL password is `mypassword` and you want to name your database `homeadv`, your URL should look like:
   `DATABASE_URL=postgres://postgres:mypassword@localhost:5432/homeadv`

Step 3: Create the Database
You need to create the actual database in PostgreSQL before the app can connect to it. Run this command in your terminal (assuming you named it `homeadv`):
psql -U postgres -c "CREATE DATABASE homeadv;"

(You will be prompted to enter your PostgreSQL password)

Step 4: Run the Database Schema and Seed Scripts
The project provides SQL files to structure your database and add initial data. Run the following commands to execute them against your new database:
psql -U postgres -d homeadv -f db/schema.sql
psql -U postgres -d homeadv -f db/seed.sql
(Note: If you plan to use the ORM directly instead of the SQL files, you can alternately run `npm run db:push` to sync the Drizzle schema).

Step 5: Start the Development Server
Once the dependencies are installed and the database is completely set up, you can start the frontend and backend servers concurrently:
npm run dev
The application should now be accessible in your web browser, typically at `http://localhost:5000` (or whatever PORT is configured in your `.env`).

6. Running the Application

To run the application, you can use the following commands:

- Start the development server: `npm run dev`
- Start the backend server: `npm run server`
- Start the frontend server: `npm run client`
- Stop the development server: `Ctrl + C`

7. Verifying the Vertical Slice

For our vertical slice, we implemented the functionality to add funds toward a budget goal using the "Add" button. Follow these steps to verify it works from end-to-end:

Triggering the Feature

1. Ensure your development server is running (`npm run dev`) and navigate to the application in your browser (e.g., `http://localhost:5000`).
2. Navigate to your Savings Goal dashboard.
3. Click the **"Add"** button to open the contribution form.
4. Enter an amount to add to your savings goal and click **"Submit"**.
5. The UI should immediately update to reflect your newly added funds in the progress bar or total amount.

Verifying Database Update
To guarantee the data was actually saved on the backend, check the database directly:

1. Open your terminal and connect to your database using `psql`:
   psql -U postgres -d homeadv
2. Run the following SQL query to view the latest entries in the contributions table:
   SELECT * FROM savings_contributions ORDER BY created_at DESC LIMIT 5;
   (Note: Depending on your exact schema, the table name might also be `SavingsEntry`. Ensure you see your newly submitted amount in the results).

Verifying Persistence
To ensure the backend API and frontend state are properly synchronized, simulate a fresh session:

1. Go back to your browser window where the application is running.
2. Hard refresh the page (Ctrl + F5 or Cmd + Shift + R).
3. Verify that the updated savings amount is still completely intact and rendered faithfully on the screen without needing to re-add the money.