Ledgerly Technical Documentation

> **Product Name**: Ledgerly
>
> **Version**: 0.1
>
> **Date**: 1/01/2025

About Ledgerly

Ledgerly is a smart, user-friendly platform designed to simplify personal expense management. Ideal for busy professionals and tech-savvy individuals, Ledgerly empowers users to upload monthly bank statements (CSV, XML, or PDF), automatically categorize transactions, and visualize spending habits through interactive dashboards. Whether you're a young professional striving to manage finances. With customizable categories, AI-driven savings insights, and seamless data export options, Ledgerly makes financial clarity accessible for everyone.

Core Features

1. Authentication: Login and Sign Up

Authentication in the application will be handled using Clerk, which simplifies the implementation process with prebuilt components for both email/password and Google authentication.

- Add **ClerkProvider** to your app by wrapping it in `layout.tsx` to provide global session and user context.

- Use Clerk's prebuilt components:
  - **SignInButton** and **UserButton** for authentication actions.
  - **SignedIn** and **SignedOut** components to control visibility of content based on user authentication state.
  - Strictly follow ## 1. Clerk Implementation example for implmentation.

2. File Upload

- Users should be able to upload one or more CSV, XLS, and PDF files from a popup “Upload files” form, accessible from a button “Upload” on the **/dashboard** or **/transactions** routes. This button will be placed on the far top right side of the page.
- Start extraction with a button “Submit.”
- **Default Schema (Database Fields and Types):**

Field Name	Data Type	Description
Transaction Date	`DATE`	The date of the transaction.
Transaction Amount	`FLOAT`	The monetary value of the transaction.
Transaction Name	`VARCHAR`	The name or description of the transaction.
Account Balance	`FLOAT`	The account balance after the transaction.
Transaction Description	`TEXT`	Additional details or notes about the transaction.

- Users must specify the transaction period covered in the document (e.g., January 1st to January 30th) using the shadcn **Date Range Picker** component.
- Uploaded files will be displayed in the form as items, showing the file name with a button to preview the file.
- Users can upload only one file per submission.
- Server-side file processing for efficient and accurate data handling.

Shadcn Components:

- **Dialog**: For the popup form.
- **Form**: For validation and field organization.

- **Input**: For file upload.
- **Date Range Picker**: For transaction period selection.
- **Button**: For submission and actions.
- **Table**: For listing uploaded files with preview options.
- **Alert**: For displaying error notifications during file upload.
- **Spinner**: For indicating loading states during file processing.

3. Text Extraction

- Use LlamaParser for PDF text extraction (server-side).
- For each file, combine all document chunks for complete text, make sure return full text of all documents, not just the first one documents [0].
- The LlamaParser text extraction should happen immediately after user upload files to UI, and not wait for button click.
- Strictly follow ## 1. Llama Parser documentation as code implementation example.
- After text extraction, parse the extracted text to identify transactions.
- Save the parsed transactions directly into the **Supabase PostgreSQL database**.
- Once saved, ensure the transactions are immediately **visible on the `/dashboard` and `/transactions` routes**.
- Associate uploaded documents with the respective user to ensure personalized data visibility.
- The extraction and storage pipeline must handle:
    - **Error Handling:** Use **shadcn Alert Dialog** for real-time error notifications.
        - Example errors:
            - Unsupported file format.
            - Text extraction failure.

        - The dialog should clearly state the error message and offer actionable options (e.g., "Retry" or "Cancel").
    - **Duplicate Checks:** Prevent duplicate transactions by checking for identical file uploads or entries with the same metadata.
- **Server-Side Implementation Only:**
    - Ensure all text extraction and processing tasks occur on the **server side** to maintain security and prevent sensitive data exposure on the client side.
- **Date Range Validation:**
    - Validate and store the **transaction period** (start and end dates) for every file, using user input from the **Upload Files** form.
    - Link extracted transactions with their respective transaction period to allow for accurate filtering and reporting.
- Show a **loading spinner** during file upload and extraction.
- Use **shadcn Alert Dialog** to notify users of:
    - Extraction success (e.g., "File processed successfully!").
    - Extraction failure or unsupported file format.
- **Shadcn Components**:
    - **Alert**: For error notifications during extraction.
    - **Spinner**: For loading states during text extraction.
    - **Table**: For displaying extracted data.
    - **Card**: For organizing extracted content.
- **Strictly follow**: ## 2. Llama Parser documentation for implementation.

4. Manual Transaction Entry

- **Adding Transactions:**

  Users can manually add individual transactions through a popup form accessible via a button labeled **"Add Transaction"** located at the top-right corner of the **/dashboard** and **/transactions** routes.

- **Mandatory Fields for Each Transaction:**
    - **Date:** Date of the transaction (use Shadcn's ## 12. Date Range Picker for seamless date selection).
    - **Amount:** Numeric input field for the transaction amount.
    - **Name:** Text field for the transaction description (e.g., "Grocery Shopping").
    - **Type:** Dropdown with options `Expense` and `Income`.
    - **Account Type:** Dropdown with options such as `Cash`, `Savings`, or `Checking`.
    - **Category:** Dropdown with predefined categories such as `Food`, `Transportation`, `Entertainment`, etc.
    - **Recurring Options:** A dropdown to select the recurrence pattern:
        - **Options:**
            - `Never Repeat` (default)
            - `Daily`
            - `Weekly`
            - `Monthly`
            - `Quarterly`
            - `Annually`
        - When a recurring option is selected:
            - The next transaction will occur on the same day of the selected recurrence period.
            - Users can adjust the date manually using the ## 12. date range picker field to set a custom recurrence day.
- **Editing and Deleting Transactions:**
    - Transactions can be edited inline or through the same popup form.
    - Users can delete individual transactions using a delete button displayed in the action column of the transaction table.

- **Recurring Options Behaviour:**
    - If a recurring option is selected (e.g., `Monthly`), the transaction will repeat on the same day of the next month by default.
    - Users can adjust the recurrence day by modifying the date in the **Date** field.
- **Server-Side Processing:**
    - Transaction Submission:
        - Each transaction submission is sent to the backend via a POST request.
    - Recurring Transactions:
        - If the transaction is recurring, the recurrence details are stored in the database with the selected pattern.
        - The backend generates the next transaction based on the recurrence setting and inserts it automatically.
- **Database Storage:**

  Transactions are stored in the Supabase PostgreSQL database.
- **Database Schema (Example):**

Field	Type	Description
id	VARCHAR	Unique alphanumeric ID for the transaction.
date	DATE	Transaction date.
amount	NUMERIC	Transaction amount.
name	TEXT	Transaction name or description.
type	ENUM	`Expense` or `Income`.
account_type	ENUM	`Cash`, `Savings`, or `Checking`.
category	VARCHAR	Category of the transaction.
recurring_frequency	ENUM	Frequency of recurrence (e.g., `Never`, `Monthly`).
created_at	TIMESTAMP	Timestamp when the transaction was added.

- **Shadcn Components**:

    - **Dialog**: For the popup form.
    - **Form**: For input fields and validation.
    - **Input**: For transaction details (e.g., amount, name).
    - **Select**: For dropdowns (e.g., transaction type, category).
    - **Date Range Picker**: For transaction date selection.
    - **Button**: For submission and actions.
    - **Alert**: For success/error notifications.
- **Strictly follow**: ## 13. Shadcn Popup Form for Adding Manual Transactions for form implementation and ## 12. Date Range Picker.

5. Data Processing:

- Upon clicking "Submit" in the **Import Transactions** form, all uploaded files are processed, and their extracted data is sent to OpenAI for structured information extraction.
- Utilize OpenAI's structured output for precise and consistent data extraction.
- Strictly follow **## 3. OpenAI Structured Data Extraction** as the code implementation example.
- Automatically categorize transactions using AI and default category templates.
- AI based analytics for user and suggestions according to their goals.
- Allow users to customize or create new categories via the **/categories** route.
- Enable manual reassignment of transactions to different categories for flexibility.
- **Shadcn Components**:
    - **Alert**: For success/error notifications.
- **Strictly follow:** ## 3. OpenAI Structured Data Extraction for implementation.

6. Manage Categories

Purpose:

Allow users to customize transaction categories.

Features:

- List of all categories (default and custom).
- Add, edit, or delete categories.

1. **Visualizations:**

- Editable list of categories with icons and color coding.
- Modal or inline form for adding/editing categories.
- **Shadcn Components**:
    - **Table**: For listing categories.
    - **Dialog**: For adding/editing categories.
    - **Form**: For category input fields.
    - **Input**: For category names.
    - **Button**: For submission and actions.
    - **Alert**: For success/error notifications.
- **Strictly follow**: ## 10. Shadcn Data Table Implementation for table setup.

7. Manage Transactions

Purpose:

Show all user transactions with filtering, sorting, and editing capabilities.

Features:

- Search bar for transactions.
- Filters for date range, category, and amount.
- Inline editing for transaction details (e.g., category reassignment).
- Batch actions (e.g., delete multiple transactions).
- export transactions
- upload

- add transaction

Visualizations:

- Table of all transactions with sortable columns.
- Filters as dropdowns or date pickers.
- **Shadcn Components**:
  - **Table**: For displaying transactions.
  - **Dropdown**: For filters and actions.
  - **Input**: For search functionality.
  - **Date Range Picker**: For date range filtering.
  - **Button**: For actions like export and upload.
  - **Alert**: For success/error notifications.
- **Strictly follow**: ## 10. Shadcn Data Table Implementation for table setup and ## 12. Shadcn Date Range Picker Implementation.

8. Recurring and Future Transactions Management

- Provide users with the ability to manage recurring financial transactions efficiently.
- Users can add new recurring transactions with predefined frequency options, such as:
  - Never repeat, daily, working days only, weekly, bi-weekly, tri-weekly, monthly, bi-monthly, quarterly, first/last day of the week, semi-annually, or annually.
  - And users can also specify day.
- Fetch all recurring transactions for a user, displaying them in a structured format for easy review.
- Also display a list of upcoming recurring transaction dates, showing the transaction name, amount, and options to edit the amount or delete any upcoming recurring transaction.

- Allow users to update existing recurring transactions by modifying key details, including amount, frequency, or applicable dates.
- Enable the deletion of recurring transactions when they are no longer needed.
- Transactions are stored in a **Supabase PostgreSQL database** to ensure secure and reliable management.
- Provide an interactive and user-friendly UI to manage recurring transactions with clear and organized options.
- Utilize **shadcn components** for inputs and modals to create a consistent and accessible user experience.
- **Shadcn Components**:
    - **Table**: For displaying recurring transactions.
    - **Dialog**: For adding/editing recurring transactions.
    - **Form**: For input fields.
    - **Select**: For recurrence frequency.
    - **Date Range Picker**: For start and end dates.
    - **Button**: For submission and actions.
    - **Alert**: For success/error notifications.
- **Strictly follow**: ## 10. Shadcn Data Table Implementation for table setup and ## 11. Shadcn Date Range Picker Implementation.

Pages

1. `/dashboard` - Dashboard Page

Purpose: Provide an overview of financial data, including key insights, trends, and recent transactions. This page serves as the central hub for users to monitor their financial health and make informed decisions.

Visuals and Data

1. Interactive Charts

The dashboard will feature multiple interactive charts to visualize financial data. Each chart will serve a specific analytical purpose:

1. **Bar Chart - Interactive**:
    - **Type of Analytics**: **Spending Trends Over Time**.
    - **Purpose**: Display daily, weekly, or monthly spending and income trends.
    - **Data**:
        - X-axis: Time (e.g., days, weeks, months).
        - Y-axis: Amount (e.g., expenses, income).
    - **Features**:
        - Toggle between **Expenses** and **Income** to compare trends.
        - Hover over bars to view detailed transaction amounts for a specific date.
    - **Shadcn Components**:
        - **BarChart**: For visualizing trends. Strictly follow ## 5. Bar Chart - Interactive.
        - **Card**: For chart container.
        - **Button**: For toggling between expenses and income.
2. **Pie Chart - Donut with Text**:
    - **Type of Analytics**: **Spending by Category**.
    - **Purpose**: Show the distribution of expenses across different categories (e.g., Food, Transportation, Entertainment).
    - **Data**:
        - Categories: Labels for each spending category.
        - Values: Percentage of total spending per category.
    - **Features**:
        - Display the total spending amount in the center of the donut chart.

    - Hover over segments to view the exact amount spent in each category.
  - **Shadcn Components**:
    - **PieChart**: For visualizing category distribution. Strictly follow ## 6. Shadcn Pie Chart - Donut with Text.
    - **Card**: For chart container.
3. **Line Chart - Dots Colors**:
  - **Type of Analytics**: **Net Balance Over Time**.
  - **Purpose**: Track the net balance (income minus expenses) over a selected time period.
  - **Data**:
    - X-axis: Time (e.g., days, weeks, months).
    - Y-axis: Net balance (currency).
  - **Features**:
    - Color-coded dots to highlight positive (green) and negative (red) net balances.
    - Hover over dots to view the exact net balance for a specific date.
  - **Shadcn Components**:
    - **LineChart**: For visualizing net balance trends. Strictly follow ## 8. Shadcn Line Chart - Dots Colors.
    - **Card**: For chart container.
4. **Radar Chart - Dots**:
  - **Type of Analytics**: **Spending Distribution by Category Over Time**.
  - **Purpose**: Compare spending across categories over multiple time periods (e.g., months).
  - **Data**:
    - Axes: Categories (e.g., Food, Transportation).
    - Values: Spending amounts for each category over time.

  - **Features**:
    - Multiple radar lines to compare spending across different months.
    - Hover over dots to view exact spending amounts.
  - **Shadcn Components**:
    - **RadarChart**: For visualizing category distribution over time. Strictly follow ## 9. Radar Chart - Dots.
    - **Card**: For chart container.

2. Summary Cards

- **Primary Visual**: A set of cards displaying key financial metrics at a glance.
- **Metrics**:
  - **Total Expenses**: Sum of all expenses for the selected time period.
  - **Total Income**: Sum of all income for the selected time period.
  - **Net Balance**: Difference between total income and total expenses.
  - **Most Spent Category**: The category with the highest spending.
- **Features**:
  - Dynamic updates based on selected filters (e.g., date range, category).
  - Color-coded badges for positive (green) and negative (red) net balance.
- **Shadcn Components**:
  - **Card**: For each summary metric.
  - **Badge**: For highlighting positive/negative net balance.
  - **Tooltip**: For hover explanations of metrics.

3. Filters

- **Primary Visual**: A toolbar above the charts for filtering data.
- **Filters**:
  - **Date Range**: Use **Date Range Picker** to filter data by a specific date range.
  - **Category**: Use **Dropdown** to filter by spending category.

    - **Account**: Use **Dropdown** to filter by account (e.g., Checking, Savings).
- **Features**:
    - Dynamic updates: Charts and summary cards update in real-time based on selected filters.
    - Reset button to clear all filters.
- **Shadcn Components**:
    - **Date Range Picker**: For date range selection.
    - **Dropdown**: For category and account filters.
    - **Button**: For applying or resetting filters.

4. Table of Recent Transactions

- **Primary Visual**: A compact table displaying the most recent transactions.
- **Columns**:
    - **Date**: The date of the transaction.
    - **Amount**: The transaction amount (formatted as currency).
    - **Name**: The name or description of the transaction.
    - **Category**: The category of the transaction.
    - **Account**: The account associated with the transaction.
- **Features**:
    - Pagination for navigating through transactions.
    - Click on a transaction to view/edit details.
- **Shadcn Components**:
    - **Table**: For displaying recent transactions. Strictly follow ## 8. Shadcn Data Table Implementation.
    - **Button**: For pagination and actions.

5. Options to Upload or Add Transactions

- **Primary Visual**: Buttons for uploading transactions or adding them manually.

- **Features**:
    - **Upload Transactions**: Open a modal to upload CSV, XLS, or PDF files.
    - **Add Transaction**: Open a modal to manually add a transaction.
- **Shadcn Components**:
    - **Button**: For upload and add actions.
    - **Dialog**: For the upload and add transaction forms. Strictly follow ## 2. Shadcn Popup Form for Adding Manual Transactions.

Shadcn Components Used

Component	Usage
BarChart	Display spending trends over time.
PieChart	Visualize spending distribution by category.
LineChart	Track net balance over time.
RadarChart	Compare spending across categories over time.
Card	Container for charts, summary cards, and recent transactions.
Table	Display recent transactions.
Date Range Picker	Filter data by date range.
Dropdown	Filter data by category and account.
Button	Actions like upload, add transaction, and reset filters.
Badge	Highlight positive/negative net balance.
Tooltip	Hover explanations for metrics and actions.
Dialog	Modal for uploading transactions or adding them manually.

Data Flow

1. **Fetch Data**: On page load, fetch transaction data from the database.
2. **Apply Filters**: Update charts, summary cards, and the recent transactions table based on selected filters.
3. **Add/Upload Transactions**:
    - Open the modal form.

- Validate and submit the form.
- Update the dashboard dynamically after saving.

4. **Edit Transaction**:
    - Open the modal form with pre-filled data.
    - Validate and submit the form.
    - Update the dashboard dynamically after saving.

Error Handling

- **Validation Errors**: Show inline errors for invalid inputs (e.g., empty transaction name).
- **API Errors**: Display an **Alert** if the server fails to fetch or save data.
- **Empty State**: Show a friendly message if no transactions exist.

2. `/transactions` - Transactions Page

Purpose: Manage and review all transactions, including viewing, filtering, sorting, editing, and exporting transaction data. This page serves as the central hub for users to track their financial activities.

Visuals and Data

1. Data Table for Transactions

- **Primary Visual**: A responsive, paginated table displaying all transactions with relevant details.
- **Table Columns**:
    - **Date**: The date of the transaction.
    - **Amount**: The transaction amount (formatted as currency).
    - **Name**: The name or description of the transaction (e.g., "Grocery Shopping").
    - **Type**: The transaction type (e.g., Expense, Income).
    - **Category**: The category of the transaction (e.g., Food, Transportation).

- **Account**: The account associated with the transaction (e.g., Checking, Savings).
    - **Actions**: Buttons for editing or deleting the transaction.
- **Features**:
    - **Sorting**: Allow users to sort transactions by date, amount, or name.
    - **Pagination**: Display transactions in pages for better performance and usability.
    - **Inline Editing**: Enable users to edit transaction details directly in the table.
- **Strictly follow**: ## 8. Shadcn Data Table Implementation for table setup.

2. Filters and Search

- **Primary Visual**: A toolbar above the table for filtering and searching transactions.
- **Filters**:
    - **Date Range**: Use **Date Range Picker** to filter transactions by a specific date range.
    - **Category**: Use a **Dropdown** to filter by category.
    - **Type**: Use a **Dropdown** to filter by transaction type (Expense, Income).
    - **Account**: Use a **Dropdown** to filter by account.
- **Search**:
    - A search bar to quickly find transactions by name or description.
- **Features**:
    - **Dynamic Filtering**: Filters update the table in real-time as users make selections.
    - **Reset Filters**: A button to clear all filters and reset the table to its default state.
- **Strictly follow**: Use **Dropdown** and **Date Range Picker** components for filtering.

3. Add/Edit Transaction Form

- **Primary Visual**: A modal dialog that opens when the user clicks "Add Transaction" or "Edit" on an existing transaction.
- **Form Fields**:
  - **Date**: A **Date Picker** for selecting the transaction date.
  - **Amount**: A numeric input for the transaction amount.
  - **Name**: A text input for the transaction name or description.
  - **Type**: A **Dropdown** for selecting the transaction type (Expense, Income).
  - **Category**: A **Dropdown** for selecting the category.
  - **Account**: A **Dropdown** for selecting the account.
  - **Recurring Options**: A **Dropdown** for setting recurrence (e.g., Never, Daily, Monthly).
- **Form Actions**:
  - **Save**: To save the new or updated transaction.
  - **Cancel**: To close the modal without saving.
- **Features**:
  - **Validation**: Ensure all required fields are filled before submission.
  - **Dynamic Updates**: Update the table immediately after saving a transaction.
- **Strictly follow**: ## 2. Shadcn Popup Form for Adding Manual Transactions for form implementation.

4. Bulk Actions

- **Primary Visual**: A toolbar above the table for performing bulk actions on transactions.
- **Actions**:
  - **Delete Multiple**: Delete selected transactions.
  - **Recategorize**: Assign a new category to selected transactions.
  - **Export Transactions**: Export the list of transactions as a CSV file.

- **Features**:
    - **Selection**: Allow users to select multiple transactions using checkboxes.
    - **Confirmation**: Show a confirmation dialog before performing destructive actions like deletion.
- **Strictly follow**: Use **Dropdown** and **Alert Dialog** components for bulk actions.

---

5. Transaction Insights

- **Primary Visual**: A small summary section above the table showing key insights.
- **Data**:
    - **Total Expenses**: The sum of all expense transactions.
    - **Total Income**: The sum of all income transactions.
    - **Net Balance**: The difference between total income and total expenses.
- **Features**:
    - **Dynamic Updates**: Update insights in real-time as transactions are added, edited, or deleted.
    - **Visual Indicators**: Use color-coded badges or icons to highlight positive/negative net balance.
- **Strictly follow**: Use **Card** and **Badge** components for displaying insights.

6. Empty State

- **Primary Visual**: A friendly message and call-to-action when no transactions exist.
- **Content**:
    - A message like "No transactions found. Start by adding your first transaction!"
    - A button to open the "Add Transaction" form.
- **Features**:

    - **Call-to-Action**: Encourage users to add their first transaction with a prominent button.
    - **Illustration**: Include an optional placeholder image or icon for visual appeal.
- **Strictly follow**: Use **Card** and **Button** components for the empty state.

Shadcn Components Used

Component	Usage
Table	Display the list of transactions with columns for date, amount, name, etc.
Dialog	Modal for adding/editing transactions.
Form	Form fields for transaction details.
Input	Text input for transaction name and search functionality.
Select	Dropdowns for type, category, and account selection.
Date Range Picker	For selecting transaction date ranges.
Button	Actions like Save, Cancel, Edit, Delete, and Add Transaction.
Alert	Notifications for success/error messages.
Badge	Display transaction type (Expense, Income).
Tooltip	Hover explanations for actions and insights.
Card	Container for summary insights and empty state messages.
Dropdown	Bulk actions and additional options.

Data Flow

1. **Fetch Transactions**: On page load, fetch the list of transactions from the database.
2. **Display Transactions**: Render the transactions in the table.
3. **Add/Edit Transaction**:
    - Open the modal form.
    - Validate and submit the form.

    - Update the table dynamically after saving.
4. **Delete Transaction**:
    - Show a confirmation dialog.
    - Remove the transaction from the table and database.
5. **Bulk Actions**:
    - Perform actions like delete or recategorize on selected transactions.
    - Update the table and database accordingly.

Error Handling

- **Validation Errors**: Show inline errors for invalid inputs (e.g., empty transaction name).
- **API Errors**: Display an **Alert** if the server fails to save or fetch transactions.
- **Empty State**: Show a friendly message if no transactions exist.

---

3. `/categories` - Categories Page

Purpose: Allow users to manage their transaction categories, including creating, editing, and deleting categories. This page provides a centralized location for users to customize their financial tracking by organizing transactions into meaningful groups.

Visuals and Data

1. Editable List of Categories

- **Primary Visual**: A clean, interactive table displaying all categories with their associated metadata (e.g., name, icon, color, and usage count).
- **Table Columns**:
    - **Name**: The name of the category (e.g., "Food", "Transportation").
    - **Icon**: A visual representation of the category (e.g., a fork for "Food").
    - **Color**: A color swatch to visually distinguish categories.
    - **Usage Count**: The number of transactions associated with the category.

    - **Actions**: Buttons for editing or deleting the category.

2. Add/Edit Category Form

- **Primary Visual**: A modal dialog that opens when the user clicks "Add Category" or "Edit" on an existing category.
- **Form Fields**:
    - **Name**: A text input for the category name.
    - **Icon**: A dropdown or icon picker for selecting an icon.
    - **Color**: A color picker for selecting a category color.
    - **Description** (optional): A text area for additional notes about the category.
- **Form Actions**:
    - **Save**: To save the new or updated category.
    - **Cancel**: To close the modal without saving.

3. Category Usage Insights

- **Primary Visual**: A small chart or summary card showing the distribution of transactions across categories.
- **Data**:
    - A pie chart or bar chart displaying the percentage of transactions in each category.
    - A summary card showing the total number of categories and the most used category.

4. Bulk Actions

- **Primary Visual**: A toolbar above the table for performing bulk actions on categories.
- **Actions**:
    - **Delete Multiple**: Delete selected categories.

5. Empty State

- **Primary Visual**: A friendly message and call-to-action when no categories exist.
- **Content**:
    - A message like "No categories found. Start by adding your first category!"
    - A button to open the "Add Category" form.

Shadcn Components Summary

Component	Usage
Table	Display the list of categories with columns for name, icon, color, etc.
Dialog	Modal for adding/editing categories.
Form	Form fields for category name, icon, color, and description.
Input	Text input for category name and description.
Select	Dropdown for selecting icons.
Button	Actions like Save, Cancel, Edit, Delete, and Add Category.
Alert	Notifications for success/error messages.
Badge	Display usage count for each category.
Tooltip	Hover explanations for icons and actions.
PieChart	Visualize the distribution of transactions across categories.
Card	Container for summary insights and empty state messages.
Dropdown	Bulk actions and additional options.

Data Flow

1. **Fetch Categories**: On page load, fetch the list of categories from the database.
2. **Display Categories**: Render the categories in the table.
3. **Add/Edit Category**:
    - Open the modal form.
    - Validate and submit the form.
    - Update the table dynamically after saving.

4. **Delete Category**:
    - Show a confirmation dialog.
    - Remove the category from the table and database.
5. **Bulk Actions**:
    - Perform actions like delete or merge on selected categories.
    - Update the table and database accordingly.

Error Handling

- **Validation Errors**: Show inline errors for invalid inputs (e.g., empty category name).
- **API Errors**: Display an **Alert** if the server fails to save or fetch categories.
- **Empty State**: Show a friendly message if no categories exist.

Strictly Follow

- **Table Implementation**: ## 8. Shadcn Data Table Implementation.
- **Form Implementation**: ## 2. Shadcn Popup Form for Adding Manual Transactions.
- **Chart Implementation**: ## 4. Shadcn Pie Chart - Donut with Text.

---

4. `/recurring-transactions` - Recurring Transactions Management Page

Purpose:

The **Recurring Transactions Management** page offers users a centralized space to add, modify, or remove recurring transactions. This page is designed to help users track and automate payments or income entries, ensuring better financial management and reducing manual efforts.

Features:

1. **Add Recurring Transactions**:

- Accessible via an **"Add Transaction" button** located at the top-right corner of the page.
- Users can define:
    - Transaction details such as name, amount, and type (income/expense).
    - Frequency of recurrence using a dropdown (e.g., monthly, weekly).
    - Applicable date range for the recurring transaction.
- Include a preview of the upcoming transaction schedule.

2. **View Recurring Transactions**:
    - Display all recurring transactions in a structured **data table** with columns for:
        - Name
        - Frequency
        - Amount
        - Start Date
        - Next Transaction Date
    - Include sorting and filtering options to refine the displayed transactions.
    - **View upcoming recurring transaction dates**, displaying:
        - Transaction name
        - Amount
        - Editable amount field for quick updates.
        - Delete any upcoming recurring transaction.
3. **Update Recurring Transactions**:
    - Users can click on a transaction in the table to open a **popup modal** for editing.
    - Allow updates to any field, including amount, frequency, or applicable dates.

    - **Quickly edit transaction amounts** for upcoming transactions directly from the table view.

4. **Delete Recurring Transactions**:
    - Provide a delete option within the transaction table, accompanied by a **confirmation dialog** using a **shadcn Alert Dialog**.
5. **Visual Feedback**:
    - Highlight upcoming transactions in a dedicated data table.
6. Follow **## 8. Shadcn data table implementation** for data table implementation.

UI Visualization:

Add Transaction Form:

- **Popup Modal** (using shadcn form components):
    - Contains input fields for:
        - **Name** (text field).
        - **Amount** (number input).
        - **Type** (dropdown: Expense/Income).
        - **Frequency** (dropdown: Never Repeat, Daily, Weekly, Monthly, etc.).
        - **Date Range** (## 12. date range picker component for start and end dates).
    - Frequency dropdown includes all predefined options (e.g., Never Repeat, Weekly, Monthly).
    - "Save" and "Cancel" buttons at the bottom of the modal.

Transaction Table:

- **shadcn Data Table** for displaying all recurring transactions:
    - Columns:
        - **Name** (transaction name).
        - **Frequency** (e.g., Weekly, Monthly).

        - **Amount**.
        - **Start Date**.
        - **Next Transaction Date**.
    - Inline actions:
        - Edit: Opens a **Popup Modal** for modifying the transaction details.
        - Delete: Triggers a **shadcn Alert Dialog** for confirmation.
    - Filters:
        - Dropdown filters for transaction type and frequency.
        - Date range filter (## 12. date range picker component).
    - **Upcoming Transactions**:
        - Show transaction name, amount, and next transaction date directly in a seperate data table.
        - Allow inline editing of the **amount field** for upcoming transactions.

Error and Confirmation Dialogs:

- Use **shadcn Alert Dialog** for:
    - Errors:
        - Display error messages when an action (e.g., adding, editing, or deleting) fails.
    - Confirmations:
        - Deletion confirmation with a message like:*"Are you sure you want to delete this recurring transaction? This action cannot be undone."*

Tech Stack

Core Frameworks and Libraries

1. **Next.js**: For server-side rendering, routing, and React-based development.
2. **React**: Core library for building the user interface.

3. **Tailwind CSS**: Utility-first CSS framework for styling.
4. **Shadcn**: For pre-built, accessible, and customizable UI components and charts.
5. **Lucid Icons**: Icon library for modern and clean visuals.

State Management

1. **Redux Toolkit**: For managing global state, particularly for transactions, categories, and user-related data.

Authentication

1. **Clerk Auth**: Clerk provides an all-in-one authentication solution with built-in components for **email/password** and **Google authentication**. It simplifies implementation by offering prebuilt, styled components for login, sign-up, and user management, eliminating the need for custom component creation. Clerk also includes seamless session management, error handling, and password recovery flows. Integration is straightforward using the `ClerkProvider`, enabling global session and user context. Strictly follow **## 1. Clerk implementation example** for detailed setup instructions.

File Parsing

1. **Llama Parser**: For extracting content from CSV, XML, and PDF documents uploaded by users.

Data Visualization

1. **Shadcn Charts**: Native charting solution provided by Shadcn for seamless integration and visualizations.

Form Handling

1. **React Hook Form**: For handling forms and validation efficiently.

Utilities

1. **Date-fns**: For managing and formatting dates.
2. **Axios**: For making API requests to Supabase and Llama Parser.

Backend instructions

Tables & Buckets already created

Supabase storage Bucket: “files”

```
 CREATE TABLE upcoming_transactions (
    id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    recurring_transaction_id BIGINT REFERENCES recurring_transac
    user_id BIGINT REFERENCES profiles (id) ON DELETE CASCADE,
    date DATE NOT NULL,
    amount NUMERIC NOT NULL,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);
```

```
CREATE TABLE recurring_transactions (
    id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    user_id BIGINT REFERENCES profiles (id) ON DELETE CASCADE,
    name TEXT NOT NULL,
    amount NUMERIC NOT NULL,
    type TEXT CHECK (type IN ('Expense', 'Income')) NOT NULL,
    account_type TEXT CHECK (account_type IN ('Cash', 'Savings',
    category_id BIGINT REFERENCES categories (id) ON DELETE SET
    frequency TEXT CHECK (
        frequency IN (
            'Never', 'Daily', 'Weekly', 'Bi-Weekly', 'Tri-Weekly
            'Bi-Monthly', 'Quarterly', 'Semi-Annually', 'Annuall
            'Working Days Only', 'First Day of Week', 'Last Day
        )
    ) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE, -- NULL means no end date
```

```sql
    description TEXT,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);
```

```sql
CREATE TABLE transactions (
    id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    user_id BIGINT REFERENCES profiles (id) ON DELETE CASCADE,
    date DATE NOT NULL,
    file_id BIGINT REFERENCES files (id) ON DELETE SET NULL,
    amount NUMERIC NOT NULL,
    name TEXT NOT NULL,
    description TEXT,
    type TEXT CHECK (type IN ('Expense', 'Income')),
    account_type TEXT CHECK (account_type IN ('Cash', 'Savings',
    category_id BIGINT REFERENCES categories (id) ON DELETE SET
    recurring_frequency TEXT CHECK (
        recurring_frequency IN (
            'Never',
            'Daily',
            'Weekly',
            'Bi-Weekly',
            'Tri-Weekly',
            'Monthly',
            'Bi-Monthly',
            'Quarterly',
            'Semi-Annually',
            'Annually',
            'Working Days Only',
            'First Day of Week',
            'Last Day of Week'
        )
    ),
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);
```

```
CREATE TABLE categories (
    id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    name TEXT NOT NULL UNIQUE, -- Ensures unique category names
    description TEXT, -- Optional description for the category
    icon TEXT, -- Stores the name of the icon (optional, can be
    color TEXT, -- Stores color code for the category (e.g., #F
    is_default BOOLEAN DEFAULT FALSE, -- Indicates if the categ
    user_id BIGINT REFERENCES profiles (id) ON DELETE CASCADE,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);
```

```
CREATE TABLE profiles (
    id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    email TEXT UNIQUE NOT NULL,
    name TEXT NOT NULL,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);
```

```
CREATE TABLE files (
    id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    user_id BIGINT REFERENCES profiles (id) ON DELETE CASCADE,
    file_name TEXT NOT NULL,
    file_url TEXT NOT NULL,
    uploaded_at TIMESTAMPTZ DEFAULT NOW()
);
```

```
CREATE TABLE analytics (
    id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    user_id BIGINT REFERENCES profiles (id) ON DELETE CASCADE,
    file_id BIGINT REFERENCES files (id) ON DELETE CASCADE,
    total_income NUMERIC,
    total_expenses NUMERIC,
    spending_by_category JSONB,
    net_balance NUMERIC,
    unusual_transactions JSONB,
    created_at TIMESTAMPTZ DEFAULT NOW()
);
```

Requirements

1. User Management

1. **Create User in Profiles Table**

    - Retrieve the `userId` from Clerk after user signs in.
    - Check if this `userId` exists in the `profiles` table.
    - If not, create a new record with the required fields (e.g., `user_id`, `email`, `name`).
    - If it exists, proceed with backend operations, passing the `user_id` to other functions.

---

2. File Upload Management

1. **Upload Files to Supabase Storage**:

    - Accept user-uploaded files.
    - Send the file to **Replicate** for processing.
    - Upload the processed file to the **Supabase `files` storage bucket**.
    - Store file metadata in the `files` table, linked to the `user_id`.

---

3. Extract and Categorize Transactions

1. **Use Llama Parser for Transaction Extraction**:

    - Extract raw text data from uploaded files.
    - Parse structured transaction fields: `date`, `amount`, `name`, `description`.
2. **Categorize Transactions with GPT-4o**:
    - Categorize transactions into predefined or custom categories.
    - Save categorized transactions in the `transactions` table.

---

4. Analyze Transactions by Timestamp

1. **Link Transactions to File Upload**:
    - Add `file_id` to the `transactions` table to link transactions to uploaded files.
2. **Analyze Transactions**:
    - Provide insights such as:
        - Total income and expenses.
        - Spending trends.
        - Duplicate transactions.

---

5. Suggestions and Analytics

1. **Generate Suggestions**:
    - Highlight key spending categories.
    - Suggest budgeting limits.
    - Identify unusual transactions.
2. **Provide Analytics**:
    - Create visual reports: bar charts, pie charts, and trend lines.
    - Endpoint: **GET /analytics**.

---

6. Transaction Management

1. **Create Transactions**:
    - Endpoint: **POST /transactions**.

- Input:
    - `{ date, amount, name, description, type, account_type, category_id, recurring_frequency }`.
- Validate required fields and save to the `transactions` table.

---

7. Manage Categories

1. **CRUD Operations for Categories**:
    - Add, edit, or delete predefined and custom categories.
    - Endpoints:
        - **POST /categories** (create).
        - **GET /categories** (fetch all).
        - **PUT /categories/:id** (update).
        - **DELETE /categories/:id** (delete).

---

8. Recurring Transactions Management

1. **Handle Recurring Transactions**:
    - Allow users to:
        - Add new recurring transactions.
        - Edit/delete existing ones.
        - View upcoming recurring transactions.
2. **Automate Recurring Transactions**:
    - Background job generates recurring transactions for the next occurrence.
    - Update the `next_transaction_date` field accordingly.

---

9. Dashboard and Insights

1. **Dashboard Data**:

- Fetch summarized data for total income, expenses, net balance, spending by category, and recurring transaction summaries.
- Endpoint: **GET /dashboard**.

---

10. Backend Workflow

1. **File Upload and Processing**:
   - Parse files using Llama Parser.
   - Categorize transactions using GPT-4o.
   - Save to the `transactions` table.
2. **Analytics Generation**:
   - Generate analytics for uploaded files and transactions.
3. **Suggestions and Insights**:
   - Generate actionable insights for the user using GPT-4o.

Code documentations

## 1. Clerk Implementation example

Add ClerkProvider to your app

The ClerkProvidercomponent provides session and user context to Clerk's hooks and components. It's recommended to wrap your entire app at the entry point with ClerkProvider to make authentication globally accessible. See the reference docs for other configuration options.

You can control which content signed-in and signed-out users can see with Clerk's prebuilt components.

/src/app/layout.tsx

```
import {
  ClerkProvider,
  SignInButton,
  SignedIn,
  SignedOut,
  UserButton
} from '@clerk/nextjs'
import './globals.css'
export default function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
    <ClerkProvider>
      <html lang="en">
        <body>
          <SignedOut>
            <SignInButton />
          </SignedOut>
          <SignedIn>
            <UserButton />
          </SignedIn>
          {children}
        </body>
      </html>
    </ClerkProvider>
  )
}
```

## 2. Llama Parser documentation for implementating with typescript

```
import {
  LlamaParseReader,
  // Optionally include additional utilities
} from "llamaindex";
import 'dotenv/config'; // Ensure environment variables are l
oaded
```

```
async function processUploadedFile(filePath: string) {
  try {
    // Initialize LlamaParser reader
    const reader = new LlamaParseReader({ resultType: "text"
});

    // Parse the uploaded document
    const documents = await reader.loadData(filePath);

    // Combine all document chunks into a single text output
    const fullText = documents.map((doc) => doc.text).join
("\n");

    // Log the full extracted text
    console.log("Extracted Text:", fullText);

    // Custom logic to parse transactions from text (implemen
t parseTransactions)
    const transactions = parseTransactions(fullText);

    // Save transactions to Supabase PostgreSQL database
    await saveTransactionsToDatabase(transactions);

    return { success: true };
  } catch (error) {
    console.error("Error processing file:", error);

    // Show error message using shadcn Alert Dialog
    showErrorDialog({
      title: "Extraction Failed",
      description: `An error occurred while processing the fi
le: ${error.message}. Please try again.`,
      actions: [
        { label: "Retry", onClick: () => retryFileUpload(file
Path) },
        { label: "Cancel", onClick: () => console.log("Operat
ion canceled.") },
      ],
    });

    throw new Error("Text extraction failed.");
  }
}

// Example function for saving transactions to the database
async function saveTransactionsToDatabase(transactions) {
  // Replace with database integration logic (e.g., Supabase
client)
  console.log("Saving transactions:", transactions);
}

// Example implementation of shadcn Alert Dialog
function showErrorDialog({ title, description, actions }) {
  return (
    <AlertDialog>
      <AlertDialogTrigger asChild>
        <button>Error</button>
      </AlertDialogTrigger>
      <AlertDialogContent>
        <AlertDialogHeader>
          <AlertDialogTitle>{title}</AlertDialogTitle>
          <AlertDialogDescription>{description}</AlertDialogD
escription>
        </AlertDialogHeader>
        <AlertDialogFooter>
          {actions.map((action) => (
            <button onClick={action.onClick} key={action.labe
l}>
              {action.label}
            </button>
          ))}
        </AlertDialogFooter>
      </AlertDialogContent>
    </AlertDialog>
  );
}

// Example parsing function
function parseTransactions(text: string) {
  // Custom parsing logic to extract transactions
  // Example: Split text by lines and extract specific fields
  const transactions = text.split("\n").map((line) => {
    const [date, amount, name, type] = line.split(",");
    return {
      date,
      amount,
      name,
      type,
      transactionId: generateUniqueId(), // Generate unique t
ransaction ID
    };
  });
  return transactions;
}

// Helper function to generate unique IDs
function generateUniqueId() {
  return `txn-${Math.random().toString(36).substr(2, 9)}`;
}
```

Workflow Summary:

1. **File Upload**:
    - Users upload a file via the UI.
    - The file is processed immediately on the server.
2. **Text Extraction**:

- LlamaParser extracts and combines the full text from all document chunks.

3. **Transaction Parsing**:
    - Parse extracted text to identify and structure transactions.
4. **Error Handling**:
    - Use **shadcn Alert Dialog** to handle errors during extraction or parsing.
5. **Data Storage**:
    - Save parsed transactions to the database and display them in the user's dashboard.
6. **Data Validation**:
    - Ensure transactions fall within the user-specified date range.

This implementation ensures a seamless user experience while maintaining data security and accuracy.

## 3. OpenAI Structured data extraction

Make sure you use the gpt-4o model and zod for defining data structures.

```
import OpenAI from "openai";
import { z } from "zod";
import { zodResponseFormat } from "openai/helpers/zod";

const openai = new OpenAI();

const ResearchPaperExtraction = z.object({
  title: z.string(),
  authors: z.array(z.string()),
  abstract: z.string(),
  keywords: z.array(z.string()),
});

const completion = await openai.beta.chat.completions.parse({
  model: "gpt-4o-2024-08-06",
  messages: [
    { role: "system", content: "You are an expert at structured
    { role: "user", content: "..." },
  ],
  response_format: zodResponseFormat(ResearchPaperExtraction, "
});

const research_paper = completion.choices[0].message.parsed;
```

## 4. Supabase Implementation Example

Query Supabase data from Next.js

Create a new file at `app/countries/page.tsx` and populate with the following.

This will select all the rows from the `countries` table in Supabase and render them on the page.

app/countries/page.tsx

```
  import { createClient } from '@/utils/supabase/server';

  export default async function Countries() {
    const supabase = await createClient();
    const { data: countries } = await supabase.from("countries"

    return <pre>{JSON.stringify(countries, null, 2)}</pre>
  }
```

utils/supabase/server.ts

```
  import { createServerClient } from '@supabase/ssr'
  import { cookies } from 'next/headers'

  export async function createClient() {
    const cookieStore = await cookies()
    return createServerClient(
      process.env.NEXT_PUBLIC_SUPABASE_URL!,
      process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!,
      {
        cookies: {
          getAll() {
            return cookieStore.getAll()
          },
          setAll(cookiesToSet) {
            try {
              cookiesToSet.forEach(({ name, value, options }) =>
                cookieStore.set(name, value, options)
              )
            } catch {
              // The `setAll` method was called from a Server Co
              // This can be ignored if you have middleware refr
              // user sessions.
            }
          },
        },
      }
    )
  }
```

Adding files to supabase storage example

```
import { createClient } from '@supabase/supabase-js'

// Create Supabase client
const supabase = createClient('your_project_url', 'your_supabase

// Upload file using standard upload
async function uploadFile(file) {
  const { data, error } = await supabase.storage.from('bucket_na
  if (error) {
    // Handle error
  } else {
    // Handle success
  }
}
```

## 5. Bar chart - Interactive

This will serve as our main chart, occupying three grid spaces on /dashboard route, while all other charts will take up one grid space each. Each row will contain a single chart.

```
"use client"

import * as React from "react"
import { Bar, BarChart, CartesianGrid, XAxis } from "recharts"

import {
  Card,
  CardContent,
  CardDescription,
  CardHeader,
  CardTitle,
} from "@/components/ui/card"
import {
  ChartConfig,
  ChartContainer,
  ChartTooltip,
  ChartTooltipContent,
} from "@/components/ui/chart"

export const description = "An interactive bar chart"

const chartData = [
  { date: "2024-04-01", desktop: 222, mobile: 150 },
  { date: "2024-04-02", desktop: 97, mobile: 180 },
  { date: "2024-04-03", desktop: 167, mobile: 120 },
  { date: "2024-04-04", desktop: 242, mobile: 260 },
  { date: "2024-04-05", desktop: 373, mobile: 290 },
  { date: "2024-04-06", desktop: 301, mobile: 340 },
  { date: "2024-04-07", desktop: 245, mobile: 180 },
  { date: "2024-04-08", desktop: 409, mobile: 320 },
  { date: "2024-04-09", desktop: 59, mobile: 110 },
  { date: "2024-04-10", desktop: 261, mobile: 190 },
  { date: "2024-04-11", desktop: 327, mobile: 350 },
  { date: "2024-04-12", desktop: 292, mobile: 210 },
  { date: "2024-04-13", desktop: 342, mobile: 380 },
  { date: "2024-04-14", desktop: 137, mobile: 220 },
  { date: "2024-04-15", desktop: 120, mobile: 170 },
  { date: "2024-04-16", desktop: 138, mobile: 190 },
  { date: "2024-04-17", desktop: 446, mobile: 360 },
  { date: "2024-04-18", desktop: 364, mobile: 410 },
  { date: "2024-04-19", desktop: 243, mobile: 180 },
  { date: "2024-04-20", desktop: 89, mobile: 150 },
  { date: "2024-04-21", desktop: 137, mobile: 200 },
  { date: "2024-04-22", desktop: 224, mobile: 170 },
  { date: "2024-04-23", desktop: 138, mobile: 230 },
  { date: "2024-04-24", desktop: 387, mobile: 290 },
  { date: "2024-04-25", desktop: 215, mobile: 250 },
  { date: "2024-04-26", desktop: 75, mobile: 130 },
  { date: "2024-04-27", desktop: 383, mobile: 420 },
  { date: "2024-04-28", desktop: 122, mobile: 180 },
  { date: "2024-04-29", desktop: 315, mobile: 240 },
  { date: "2024-04-30", desktop: 454, mobile: 380 },
  { date: "2024-05-01", desktop: 165, mobile: 220 },
  { date: "2024-05-02", desktop: 293, mobile: 310 },
  { date: "2024-05-03", desktop: 247, mobile: 190 },
  { date: "2024-05-04", desktop: 385, mobile: 420 },
  { date: "2024-05-05", desktop: 481, mobile: 390 },
  { date: "2024-05-06", desktop: 498, mobile: 520 },
  { date: "2024-05-07", desktop: 388, mobile: 300 },
  { date: "2024-05-08", desktop: 149, mobile: 210 },
  { date: "2024-05-09", desktop: 227, mobile: 180 },
  { date: "2024-05-10", desktop: 293, mobile: 330 },
  { date: "2024-05-11", desktop: 335, mobile: 270 },
  { date: "2024-05-12", desktop: 197, mobile: 240 },
  { date: "2024-05-13", desktop: 197, mobile: 160 },
  { date: "2024-05-14", desktop: 448, mobile: 490 },
  { date: "2024-05-15", desktop: 473, mobile: 380 },
  { date: "2024-05-16", desktop: 338, mobile: 400 },
  { date: "2024-05-17", desktop: 499, mobile: 420 },
  { date: "2024-05-18", desktop: 315, mobile: 350 },
  { date: "2024-05-19", desktop: 235, mobile: 180 },
  { date: "2024-05-20", desktop: 177, mobile: 230 },
  { date: "2024-05-21", desktop: 82, mobile: 140 },
  { date: "2024-05-22", desktop: 81, mobile: 120 },
  { date: "2024-05-23", desktop: 252, mobile: 290 },
  { date: "2024-05-24", desktop: 294, mobile: 220 },
  { date: "2024-05-25", desktop: 201, mobile: 250 },
  { date: "2024-05-26", desktop: 213, mobile: 170 },
  { date: "2024-05-27", desktop: 420, mobile: 460 },
  { date: "2024-05-28", desktop: 233, mobile: 190 },
  { date: "2024-05-29", desktop: 78, mobile: 130 },
  { date: "2024-05-30", desktop: 340, mobile: 280 },
  { date: "2024-05-31", desktop: 178, mobile: 230 },
  { date: "2024-06-01", desktop: 178, mobile: 200 },
  { date: "2024-06-02", desktop: 470, mobile: 410 },
  { date: "2024-06-03", desktop: 103, mobile: 160 },
  { date: "2024-06-04", desktop: 439, mobile: 380 },
  { date: "2024-06-05", desktop: 88, mobile: 140 },
  { date: "2024-06-06", desktop: 294, mobile: 250 },
  { date: "2024-06-07", desktop: 323, mobile: 370 },
  { date: "2024-06-08", desktop: 385, mobile: 320 },
  { date: "2024-06-09", desktop: 438, mobile: 480 },
  { date: "2024-06-10", desktop: 155, mobile: 200 },
  { date: "2024-06-11", desktop: 92, mobile: 150 },
  { date: "2024-06-12", desktop: 492, mobile: 420 },
  { date: "2024-06-13", desktop: 81, mobile: 130 },
  { date: "2024-06-14", desktop: 426, mobile: 380 },
  { date: "2024-06-15", desktop: 307, mobile: 350 },
  { date: "2024-06-16", desktop: 371, mobile: 310 },
  { date: "2024-06-17", desktop: 475, mobile: 520 },
  { date: "2024-06-18", desktop: 107, mobile: 170 },
  { date: "2024-06-19", desktop: 341, mobile: 290 },
  { date: "2024-06-20", desktop: 408, mobile: 450 },
  { date: "2024-06-21", desktop: 169, mobile: 210 },
  { date: "2024-06-22", desktop: 317, mobile: 270 },
  { date: "2024-06-23", desktop: 480, mobile: 530 },
  { date: "2024-06-24", desktop: 132, mobile: 180 },
  { date: "2024-06-25", desktop: 141, mobile: 190 },
  { date: "2024-06-26", desktop: 434, mobile: 380 },
  { date: "2024-06-27", desktop: 448, mobile: 490 },
  { date: "2024-06-28", desktop: 149, mobile: 200 },
  { date: "2024-06-29", desktop: 103, mobile: 160 },
  { date: "2024-06-30", desktop: 446, mobile: 400 },
]

const chartConfig = {
  views: {
    label: "Page Views",
  },
  desktop: {
    label: "Desktop",
    color: "hsl(var(--chart-1))",
  },
  mobile: {
    label: "Mobile",
    color: "hsl(var(--chart-2))",
  },
} satisfies ChartConfig

export function Component() {
  const [activeChart, setActiveChart] =
    React.useState<keyof typeof chartConfig>("desktop")
```

```
const total = React.useMemo(
  () => ({
    desktop: chartData.reduce((acc, curr) => acc + curr.deskto
    mobile: chartData.reduce((acc, curr) => acc + curr.mobile,
  }),
  []
)

return (
  <Card>
    <CardHeader className="flex flex-col items-stretch space-y
      <div className="flex flex-1 flex-col justify-center gap-
        <CardTitle>Bar Chart - Interactive</CardTitle>
        <CardDescription>
          Showing total visitors for the last 3 months
        </CardDescription>
      </div>
      <div className="flex">
        {["desktop", "mobile"].map((key) => {
          const chart = key as keyof typeof chartConfig
          return (
            <button
              key={chart}
              data-active={activeChart === chart}
              className="relative z-30 flex flex-1 flex-col ju
              onClick={() => setActiveChart(chart)}
            >
              <span className="text-xs text-muted-foreground">
                {chartConfig[chart].label}
              </span>
              <span className="text-lg font-bold leading-none
                {total[key as keyof typeof total].toLocaleStri
              </span>
            </button>
          )
        })}
      </div>
    </CardHeader>
    <CardContent className="px-2 sm:p-6">
      <ChartContainer
        config={chartConfig}
        className="aspect-auto h-[250px] w-full"
      >
        <BarChart
          accessibilityLayer
          data={chartData}
          margin={{
            left: 12,
            right: 12,
          }}
        >
          <CartesianGrid vertical={false} />
          <XAxis
            dataKey="date"
            tickLine={false}
            axisLine={false}
            tickMargin={8}
            minTickGap={32}
            tickFormatter={(value) => {
              const date = new Date(value)
              return date.toLocaleDateString("en-US", {
                month: "short",
                day: "numeric",
              })
            }}
          />
          <ChartTooltip
            content={
              <ChartTooltipContent
                className="w-[150px]"
                nameKey="views"
```

```
                    labelFormatter={(value) => {
                      return new Date(value).toLocaleDateString("e
                        month: "short",
                        day: "numeric",
                        year: "numeric",
                      })
                    }}
                  />
                }
              />
              <Bar dataKey={activeChart} fill={`var(--color-${acti
            </BarChart>
          </ChartContainer>
        </CardContent>
      </Card>
    )
  }
```

## 6. Shadcn Pie Chart - Donut with Text

```
"use client"

import * as React from "react"
import { TrendingUp } from "lucide-react"
import { Label, Pie, PieChart } from "recharts"

import {
  Card,
  CardContent,
  CardDescription,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card"
import {
  ChartConfig,
  ChartContainer,
  ChartTooltip,
  ChartTooltipContent,
} from "@/components/ui/chart"
const chartData = [
  { browser: "chrome", visitors: 275, fill: "var(--color-chrome
  { browser: "safari", visitors: 200, fill: "var(--color-safari
  { browser: "firefox", visitors: 287, fill: "var(--color-firef
  { browser: "edge", visitors: 173, fill: "var(--color-edge)" },
  { browser: "other", visitors: 190, fill: "var(--color-other)"
]

const chartConfig = {
  visitors: {
    label: "Visitors",
  },
  chrome: {
    label: "Chrome",
    color: "hsl(var(--chart-1))",
  },
  safari: {
    label: "Safari",
    color: "hsl(var(--chart-2))",
  },
  firefox: {
    label: "Firefox",
    color: "hsl(var(--chart-3))",
  },
  edge: {
    label: "Edge",
    color: "hsl(var(--chart-4))",
  },
  other: {
    label: "Other",
    color: "hsl(var(--chart-5))",
  },
} satisfies ChartConfig

export function Component() {
  const totalVisitors = React.useMemo(() => {
    return chartData.reduce((acc, curr) => acc + curr.visitors,
  }, [])

  return (
    <Card className="flex flex-col">
      <CardHeader className="items-center pb-0">
        <CardTitle>Pie Chart - Donut with Text</CardTitle>
        <CardDescription>January - June 2024</CardDescription>
      </CardHeader>
      <CardContent className="flex-1 pb-0">
        <ChartContainer
          config={chartConfig}
          className="mx-auto aspect-square max-h-[250px]"
        >
          <PieChart>
            <ChartTooltip
              cursor={false}
              content={<ChartTooltipContent hideLabel />}
            />
            <Pie
              data={chartData}
              dataKey="visitors"
              nameKey="browser"
              innerRadius={60}
              strokeWidth={5}
            >
              <Label
                content={({ viewBox }) => {
                  if (viewBox && "cx" in viewBox && "cy" in view
                    return (
                      <text
```

```
                        x={viewBox.cx}
                        y={viewBox.cy}
                        textAnchor="middle"
                        dominantBaseline="middle"
                      >
                        <tspan
                          x={viewBox.cx}
                          y={viewBox.cy}
                          className="fill-foreground text-3xl fo
                        >
                          {totalVisitors.toLocaleString()}
                        </tspan>
                        <tspan
                          x={viewBox.cx}
                          y={(viewBox.cy || 0) + 24}
                          className="fill-muted-foreground"
                        >
                          Visitors
                        </tspan>
                      </text>
                    )
                  }
                }}
              />
            </Pie>
          </PieChart>
        </ChartContainer>
      </CardContent>
      <CardFooter className="flex-col gap-2 text-sm">
        <div className="flex items-center gap-2 font-medium lead
          Trending up by 5.2% this month <TrendingUp className='
        </div>
        <div className="leading-none text-muted-foreground">
          Showing total visitors for the last 6 months
        </div>
      </CardFooter>
```

```
    </Card>
  )
}
```

## 7. Shadcn Bar Chart - Multiple

```
"use client"

import { TrendingUp } from "lucide-react"
import { Bar, BarChart, CartesianGrid, XAxis } from "recharts"

import {
  Card,
  CardContent,
  CardDescription,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card"
import {
  ChartConfig,
  ChartContainer,
  ChartTooltip,
  ChartTooltipContent,
} from "@/components/ui/chart"
const chartData = [
  { month: "January", desktop: 186, mobile: 80 },
  { month: "February", desktop: 305, mobile: 200 },
  { month: "March", desktop: 237, mobile: 120 },
  { month: "April", desktop: 73, mobile: 190 },
  { month: "May", desktop: 209, mobile: 130 },
  { month: "June", desktop: 214, mobile: 140 },
]

const chartConfig = {
  desktop: {
    label: "Desktop",
    color: "hsl(var(--chart-1))",
  },
  mobile: {
    label: "Mobile",
    color: "hsl(var(--chart-2))",
  },
} satisfies ChartConfig

export function Component() {
  return (
    <Card>
      <CardHeader>
        <CardTitle>Bar Chart - Multiple</CardTitle>
        <CardDescription>January - June 2024</CardDescription>
      </CardHeader>
      <CardContent>
        <ChartContainer config={chartConfig}>
          <BarChart accessibilityLayer data={chartData}>
            <CartesianGrid vertical={false} />
            <XAxis
              dataKey="month"
              tickLine={false}
              tickMargin={10}
              axisLine={false}
              tickFormatter={(value) => value.slice(0, 3)}
            />
            <ChartTooltip
              cursor={false}
              content={<ChartTooltipContent indicator="dashed"
            />
            <Bar dataKey="desktop" fill="var(--color-desktop)"
            <Bar dataKey="mobile" fill="var(--color-mobile)" ra
          </BarChart>
        </ChartContainer>
      </CardContent>
      <CardFooter className="flex-col items-start gap-2 text-sm'
        <div className="flex gap-2 font-medium leading-none">
          Trending up by 5.2% this month <TrendingUp className='
        </div>
        <div className="leading-none text-muted-foreground">
          Showing total visitors for the last 6 months
        </div>
      </CardFooter>
    </Card>
  )
}
```

## 8. Shadcn Line Chart - Dots Colors

```
"use client"

import { TrendingUp } from "lucide-react"
import { CartesianGrid, Dot, Line, LineChart } from "recharts"

import {
  Card,
  CardContent,
  CardDescription,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card"
import {
  ChartConfig,
  ChartContainer,
  ChartTooltip,
  ChartTooltipContent,
} from "@/components/ui/chart"
const chartData = [
  { browser: "chrome", visitors: 275, fill: "var(--color-chrome
  { browser: "safari", visitors: 200, fill: "var(--color-safari
  { browser: "firefox", visitors: 187, fill: "var(--color-firef
  { browser: "edge", visitors: 173, fill: "var(--color-edge)" }
  { browser: "other", visitors: 90, fill: "var(--color-other)"
]

const chartConfig = {
  visitors: {
    label: "Visitors",
    color: "hsl(var(--chart-2))",
  },
  chrome: {
    label: "Chrome",
    color: "hsl(var(--chart-1))",
  },
  safari: {
    label: "Safari",
    color: "hsl(var(--chart-2))",
  },
  firefox: {
    label: "Firefox",
    color: "hsl(var(--chart-3))",
  },
  edge: {
    label: "Edge",
    color: "hsl(var(--chart-4))",
  },
  other: {
    label: "Other",
    color: "hsl(var(--chart-5))",
  },
} satisfies ChartConfig

export function Component() {
  return (
      <Card>
        <CardHeader>
          <CardTitle>Line Chart - Dots Colors</CardTitle>
          <CardDescription>January - June 2024</CardDescription>
        </CardHeader>
        <CardContent>
          <ChartContainer config={chartConfig}>
            <LineChart
              accessibilityLayer
              data={chartData}
              margin={{
                top: 24,
                left: 24,
                right: 24,
              }}
            >
              <CartesianGrid vertical={false} />
              <ChartTooltip
                cursor={false}
                content={
                  <ChartTooltipContent
                    indicator="line"
                    nameKey="visitors"
                    hideLabel
                  />
                }
              />
              <Line
                dataKey="visitors"
                type="natural"
                stroke="var(--color-visitors)"
                strokeWidth={2}
                dot={({ payload, ...props }) => {
                  return (
                    <Dot
                      key={payload.browser}
                      r={5}
                      cx={props.cx}
                      cy={props.cy}
                      fill={payload.fill}
                      stroke={payload.fill}
                    />
                  )
                }}
              />
            </LineChart>
          </ChartContainer>
        </CardContent>
        <CardFooter className="flex-col items-start gap-2 text-sm'
          <div className="flex gap-2 font-medium leading-none">
            Trending up by 5.2% this month <TrendingUp className='
          </div>
          <div className="leading-none text-muted-foreground">
            Showing total visitors for the last 6 months
          </div>
        </CardFooter>
      </Card>
    )
  }
```

## 9. Radar Chart - Dots

```
"use client"

import { TrendingUp } from "lucide-react"
import { PolarAngleAxis, PolarGrid, Radar, RadarChart } from "re

import {
  Card,
  CardContent,
  CardDescription,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card"
import {
  ChartConfig,
  ChartContainer,
  ChartTooltip,
  ChartTooltipContent,
} from "@/components/ui/chart"
const chartData = [
  { month: "January", desktop: 186 },
  { month: "February", desktop: 305 },
  { month: "March", desktop: 237 },
  { month: "April", desktop: 273 },
  { month: "May", desktop: 209 },
  { month: "June", desktop: 214 },
]

const chartConfig = {
  desktop: {
    label: "Desktop",
    color: "hsl(var(--chart-1))",
  },
} satisfies ChartConfig

export function Component() {
  return (
    <Card>
      <CardHeader className="items-center">
        <CardTitle>Radar Chart - Dots</CardTitle>
        <CardDescription>
          Showing total visitors for the last 6 months
        </CardDescription>
      </CardHeader>
      <CardContent className="pb-0">
```

```
          <ChartContainer
            config={chartConfig}
            className="mx-auto aspect-square max-h-[250px]"
          >
            <RadarChart data={chartData}>
              <ChartTooltip cursor={false} content={<ChartTooltipC
              <PolarAngleAxis dataKey="month" />
              <PolarGrid />
              <Radar
                dataKey="desktop"
                fill="var(--color-desktop)"
                fillOpacity={0.6}
                dot={{
                  r: 4,
                  fillOpacity: 1,
                }}
              />
            </RadarChart>
          </ChartContainer>
        </CardContent>
        <CardFooter className="flex-col gap-2 text-sm">
          <div className="flex items-center gap-2 font-medium lead
            Trending up by 5.2% this month <TrendingUp className="
          </div>
          <div className="flex items-center gap-2 leading-none tex
            January - June 2024
          </div>
        </CardFooter>
      </Card>
    )
  }
```

## 10. Shadcn data table implementation

```
"use client"

import * as React from "react"
import {
  ColumnDef,
  ColumnFiltersState,
  SortingState,
  VisibilityState,
  flexRender,
  getCoreRowModel,
  getFilteredRowModel,
  getPaginationRowModel,
  getSortedRowModel,
  useReactTable,
} from "@tanstack/react-table"
import { ArrowUpDown, ChevronDown, MoreHorizontal } from "lucide

import { Button } from "@/components/ui/button"
import { Checkbox } from "@/components/ui/checkbox"
import {
  DropdownMenu,
  DropdownMenuCheckboxItem,
  DropdownMenuContent,
  DropdownMenuItem,
  DropdownMenuLabel,
  DropdownMenuSeparator,
  DropdownMenuTrigger,
} from "@/components/ui/dropdown-menu"
import { Input } from "@/components/ui/input"
import {
  Table,
  TableBody,
  TableCell,
  TableHead,
  TableHeader,
  TableRow,
} from "@/components/ui/table"

const data: Payment[] = [
  {
    id: "m5gr84i9",
    amount: 316,
    status: "success",
    email: "ken99@yahoo.com",
  },
  {
    id: "3u1reuv4",
    amount: 242,
    status: "success",
    email: "Abe45@gmail.com",
  },
  {
    id: "derv1ws0",
    amount: 837,
    status: "processing",
    email: "Monserrat44@gmail.com",
  },
  {
    id: "5kma53ae",
    amount: 874,
    status: "success",
    email: "Silas22@gmail.com",
  },
  {
    id: "bhqecj4p",
    amount: 721,
    status: "failed",
    email: "carmella@hotmail.com",
  },
]
```

```
export type Payment = {
  id: string
  amount: number
  status: "pending" | "processing" | "success" | "failed"
  email: string
}

export const columns: ColumnDef<Payment>[] = [
  {
    id: "select",
    header: ({ table }) => (
      <Checkbox
        checked={
          table.getIsAllPageRowsSelected() ||
          (table.getIsSomePageRowsSelected() && "indeterminate"
        }
        onCheckedChange={(value) => table.toggleAllPageRowsSele
        aria-label="Select all"
      />
    ),
    cell: ({ row }) => (
      <Checkbox
        checked={row.getIsSelected()}
        onCheckedChange={(value) => row.toggleSelected(!!value)
        aria-label="Select row"
      />
    ),
    enableSorting: false,
    enableHiding: false,
  },
  {
    accessorKey: "status",
    header: "Status",
    cell: ({ row }) => (
      <div className="capitalize">{row.getValue("status")}</div
    ),
  },
  {
    accessorKey: "email",
    header: ({ column }) => {
      return (
        <Button
          variant="ghost"
          onClick={() => column.toggleSorting(column.getIsSorte
        >
          Email
          <ArrowUpDown />
        </Button>
      )
    },
    cell: ({ row }) => <div className="lowercase">{row.getValue
  },
  {
    accessorKey: "amount",
    header: () => <div className="text-right">Amount</div>,
    cell: ({ row }) => {
      const amount = parseFloat(row.getValue("amount"))

      // Format the amount as a dollar amount
      const formatted = new Intl.NumberFormat("en-US", {
        style: "currency",
        currency: "USD",
      }).format(amount)

      return <div className="text-right font-medium">{formatted
    },
  },
  {
    id: "actions",
    enableHiding: false,
    cell: ({ row }) => {
      const payment = row.original
      return (
        <DropdownMenu>
          <DropdownMenuTrigger asChild>
            <Button variant="ghost" className="h-8 w-8 p-0">
              <span className="sr-only">Open menu</span>
              <MoreHorizontal />
            </Button>
          </DropdownMenuTrigger>
          <DropdownMenuContent align="end">
            <DropdownMenuLabel>Actions</DropdownMenuLabel>
            <DropdownMenuItem
              onClick={() => navigator.clipboard.writeText(payme
            >
              Copy payment ID
            </DropdownMenuItem>
            <DropdownMenuSeparator />
            <DropdownMenuItem>View customer</DropdownMenuItem>
            <DropdownMenuItem>View payment details</DropdownMenu
          </DropdownMenuContent>
        </DropdownMenu>
      )
    },
  },
]

export function DataTableDemo() {
  const [sorting, setSorting] = React.useState<SortingState>([]
  const [columnFilters, setColumnFilters] = React.useState<Colur
    []
  )
  const [columnVisibility, setColumnVisibility] =
    React.useState<VisibilityState>({})
  const [rowSelection, setRowSelection] = React.useState({})

  const table = useReactTable({
    data,
    columns,
    onSortingChange: setSorting,
    onColumnFiltersChange: setColumnFilters,
    getCoreRowModel: getCoreRowModel(),
    getPaginationRowModel: getPaginationRowModel(),
    getSortedRowModel: getSortedRowModel(),
    getFilteredRowModel: getFilteredRowModel(),
    onColumnVisibilityChange: setColumnVisibility,
    onRowSelectionChange: setRowSelection,
    state: {
      sorting,
      columnFilters,
      columnVisibility,
      rowSelection,
    },
  })

  return (
    <div className="w-full">
      <div className="flex items-center py-4">
        <Input
          placeholder="Filter emails..."
          value={(table.getColumn("email")?.getFilterValue() as
          onChange={(event) =>
            table.getColumn("email")?.setFilterValue(event.targe
          }
          className="max-w-sm"
        />
        <DropdownMenu>
          <DropdownMenuTrigger asChild>
            <Button variant="outline" className="ml-auto">
              Columns <ChevronDown />
            </Button>
          </DropdownMenuTrigger>
          <DropdownMenuContent align="end">
```

```
            {table
              .getAllColumns()
              .filter((column) => column.getCanHide())
              .map((column) => {
                return (
                  <DropdownMenuCheckboxItem
                    key={column.id}
                    className="capitalize"
                    checked={column.getIsVisible()}
                    onCheckedChange={(value) =>
                      column.toggleVisibility(!!value)
                    }
                  >
                    {column.id}
                  </DropdownMenuCheckboxItem>
                )
              })}
          </DropdownMenuContent>
        </DropdownMenu>
      </div>
      <div className="rounded-md border">
        <Table>
          <TableHeader>
            {table.getHeaderGroups().map((headerGroup) => (
              <TableRow key={headerGroup.id}>
                {headerGroup.headers.map((header) => {
                  return (
                    <TableHead key={header.id}>
                      {header.isPlaceholder
                        ? null
                        : flexRender(
                            header.column.columnDef.header,
                            header.getContext()
                          )}
                    </TableHead>
                  )
                })}
              </TableRow>
            ))}
          </TableHeader>
          <TableBody>
            {table.getRowModel().rows?.length ? (
              table.getRowModel().rows.map((row) => (
                <TableRow
                  key={row.id}
                  data-state={row.getIsSelected() && "selected"
                >
                  {row.getVisibleCells().map((cell) => (
                    <TableCell key={cell.id}>
                      {flexRender(
                        cell.column.columnDef.cell,
                        cell.getContext()
                      )}
                    </TableCell>
                  ))}
                </TableRow>
              ))
            ) : (
              <TableRow>
                <TableCell
                  colSpan={columns.length}
                  className="h-24 text-center"
                >
                  No results.
                </TableCell>
              </TableRow>
            )}
          </TableBody>
        </Table>
      </div>
      <div className="flex items-center justify-end space-x-2 p
        <div className="flex-1 text-sm text-muted-foreground">
```

```
          {table.getFilteredSelectedRowModel().rows.length} of{'
          {table.getFilteredRowModel().rows.length} row(s) selec
        </div>
        <div className="space-x-2">
          <Button
            variant="outline"
            size="sm"
            onClick={() => table.previousPage()}
            disabled={!table.getCanPreviousPage()}
          >
            Previous
          </Button>
          <Button
            variant="outline"
            size="sm"
            onClick={() => table.nextPage()}
            disabled={!table.getCanNextPage()}
          >
            Next
          </Button>
        </div>
      </div>
    </div>
  )
}
```

## 11. Shadcn dashbaord with sidebar that collapses to icons

```
import { AppSidebar } from "@/components/app-sidebar"
import {
  Breadcrumb,
  BreadcrumbItem,
  BreadcrumbLink,
  BreadcrumbList,
  BreadcrumbPage,
  BreadcrumbSeparator,
} from "@/components/ui/breadcrumb"
import { Separator } from "@/components/ui/separator"
import {
  SidebarInset,
  SidebarProvider,
  SidebarTrigger,
} from "@/components/ui/sidebar"

export default function Page() {
  return (
    <SidebarProvider>
      <AppSidebar />
      <SidebarInset>
        <header className="flex h-16 shrink-0 items-center gap-2
          <div className="flex items-center gap-2 px-4">
            <SidebarTrigger className="-ml-1" />
            <Separator orientation="vertical" className="mr-2 h-
            <Breadcrumb>
              <BreadcrumbList>
                <BreadcrumbItem className="hidden md:block">
                  <BreadcrumbLink href="#">
                    Building Your Application
                  </BreadcrumbLink>
                </BreadcrumbItem>
                <BreadcrumbSeparator className="hidden md:block"
                <BreadcrumbItem>
                  <BreadcrumbPage>Data Fetching</BreadcrumbPage>
                </BreadcrumbItem>
              </BreadcrumbList>
            </Breadcrumb>
          </div>
        </header>
        <div className="flex flex-1 flex-col gap-4 p-4 pt-0">
          <div className="grid auto-rows-min gap-4 md:grid-cols-
            <div className="aspect-video rounded-xl bg-muted/50"
            <div className="aspect-video rounded-xl bg-muted/50'
            <div className="aspect-video rounded-xl bg-muted/50'
          </div>
          <div className="min-h-[100vh] flex-1 rounded-xl bg-mu
        </div>
      </SidebarInset>
    </SidebarProvider>
  )
}
```

```
"use client"

import * as React from "react"
import {
  AudioWaveform,
  BookOpen,
  Bot,
  Command,
  Frame,
  GalleryVerticalEnd,
  Map,
  PieChart,
  Settings2,
  SquareTerminal,
} from "lucide-react"

import { NavMain } from "@/components/nav-main"
import { NavProjects } from "@/components/nav-projects"
import { NavUser } from "@/components/nav-user"
import { TeamSwitcher } from "@/components/team-switcher"
import {
  Sidebar,
  SidebarContent,
  SidebarFooter,
  SidebarHeader,
  SidebarRail,
} from "@/components/ui/sidebar"

// This is sample data.
const data = {
  user: {
    name: "shadcn",
    email: "m@example.com",
    avatar: "/avatars/shadcn.jpg",
  },
  teams: [
    {
      name: "Acme Inc",
      logo: GalleryVerticalEnd,
      plan: "Enterprise",
    },
    {
      name: "Acme Corp.",
      logo: AudioWaveform,
      plan: "Startup",
    },
    {
      name: "Evil Corp.",
      logo: Command,
      plan: "Free",
    },
  ],
  navMain: [
    {
      title: "Playground",
      url: "#",
      icon: SquareTerminal,
      isActive: true,
      items: [
        {
          title: "History",
          url: "#",
        },
        {
          title: "Starred",
          url: "#",
        },
        {
          title: "Settings",
          url: "#",
        },
      ],
    },
    {
      title: "Models",
      url: "#",
      icon: Bot,
      items: [
        {
          title: "Genesis",
          url: "#",
        },
        {
          title: "Explorer",
          url: "#",
        },
        {
          title: "Quantum",
          url: "#",
        },
      ],
    },
    {
      title: "Documentation",
      url: "#",
      icon: BookOpen,
      items: [
        {
          title: "Introduction",
          url: "#",
        },
        {
          title: "Get Started",
          url: "#",
        },
        {
          title: "Tutorials",
          url: "#",
        },
        {
          title: "Changelog",
          url: "#",
        },
      ],
    },
    {
      title: "Settings",
      url: "#",
      icon: Settings2,
      items: [
        {
          title: "General",
          url: "#",
        },
        {
          title: "Team",
          url: "#",
        },
        {
          title: "Billing",
          url: "#",
        },
        {
          title: "Limits",
          url: "#",
        },
      ],
    },
  ],
  projects: [
    {
      name: "Design Engineering",
      url: "#",
      icon: Frame,
    },
    {
      name: "Sales & Marketing",
      url: "#",
      icon: PieChart,
    },
    {
      name: "Travel",
      url: "#",
      icon: Map,
    },
  ],
}

export function AppSidebar({ ...props }: React.ComponentProps<ty
  return (
    <Sidebar collapsible="icon" {...props}>
      <SidebarHeader>
        <TeamSwitcher teams={data.teams} />
      </SidebarHeader>
      <SidebarContent>
        <NavMain items={data.navMain} />
        <NavProjects projects={data.projects} />
      </SidebarContent>
      <SidebarFooter>
        <NavUser user={data.user} />
      </SidebarFooter>
      <SidebarRail />
    </Sidebar>
  )
}
```

```
"use client"

import { ChevronRight, type LucideIcon } from "lucide-react"

import {
  Collapsible,
  CollapsibleContent,
  CollapsibleTrigger,
} from "@/components/ui/collapsible"
import {
  SidebarGroup,
  SidebarGroupLabel,
  SidebarMenu,
  SidebarMenuButton,
  SidebarMenuItem,
  SidebarMenuSub,
  SidebarMenuSubButton,
  SidebarMenuSubItem,
} from "@/components/ui/sidebar"

export function NavMain({
  items,
}: {
  items: {
    title: string
    url: string
    icon?: LucideIcon
    isActive?: boolean
    items?: {
      title: string
      url: string
    }[]
  }[]
}) {
  return (
    <SidebarGroup>
      <SidebarGroupLabel>Platform</SidebarGroupLabel>
      <SidebarMenu>
        {items.map((item) => (
          <Collapsible
            key={item.title}
            asChild
            defaultOpen={item.isActive}
            className="group/collapsible"
          >
            <SidebarMenuItem>
              <CollapsibleTrigger asChild>
                <SidebarMenuButton tooltip={item.title}>
                  {item.icon && <item.icon />}
                  <span>{item.title}</span>
                  <ChevronRight className="ml-auto transition-t
                </SidebarMenuButton>
              </CollapsibleTrigger>
              <CollapsibleContent>
                <SidebarMenuSub>
                  {item.items?.map((subItem) => (
                    <SidebarMenuSubItem key={subItem.title}>
                      <SidebarMenuSubButton asChild>
                        <a href={subItem.url}>
                          <span>{subItem.title}</span>
                        </a>
                      </SidebarMenuSubButton>
                    </SidebarMenuSubItem>
                  ))}
                </SidebarMenuSub>
              </CollapsibleContent>
            </SidebarMenuItem>
          </Collapsible>
        ))}
      </SidebarMenu>
    </SidebarGroup>
  )
}
```

```
"use client"

import {
  Folder,
  Forward,
  MoreHorizontal,
  Trash2,
  type LucideIcon,
} from "lucide-react"

import {
  DropdownMenu,
  DropdownMenuContent,
  DropdownMenuItem,
  DropdownMenuSeparator,
  DropdownMenuTrigger,
} from "@/components/ui/dropdown-menu"
import {
  SidebarGroup,
  SidebarGroupLabel,
  SidebarMenu,
  SidebarMenuAction,
  SidebarMenuButton,
  SidebarMenuItem,
  useSidebar,
} from "@/components/ui/sidebar"

export function NavProjects({
  projects,
}: {
  projects: {
    name: string
    url: string
    icon: LucideIcon
  }[]
}) {
  const { isMobile } = useSidebar()

  return (
    <SidebarGroup className="group-data-[collapsible=icon]:hidde
      <SidebarGroupLabel>Projects</SidebarGroupLabel>
      <SidebarMenu>
        {projects.map((item) => (
          <SidebarMenuItem key={item.name}>
            <SidebarMenuButton asChild>
              <a href={item.url}>
                <item.icon />
                <span>{item.name}</span>
              </a>
            </SidebarMenuButton>
            <DropdownMenu>
              <DropdownMenuTrigger asChild>
                <SidebarMenuAction showOnHover>
                  <MoreHorizontal />
                  <span className="sr-only">More</span>
                </SidebarMenuAction>
              </DropdownMenuTrigger>
              <DropdownMenuContent
                className="w-48 rounded-lg"
                side={isMobile ? "bottom" : "right"}
                align={isMobile ? "end" : "start"}
              >
                <DropdownMenuItem>
                  <Folder className="text-muted-foreground" />
                  <span>View Project</span>
                </DropdownMenuItem>
                <DropdownMenuItem>
                  <Forward className="text-muted-foreground" />
                  <span>Share Project</span>
                </DropdownMenuItem>
                <DropdownMenuSeparator />
                <DropdownMenuItem>
                  <Trash2 className="text-muted-foreground" />
                  <span>Delete Project</span>
                </DropdownMenuItem>
              </DropdownMenuContent>
            </DropdownMenu>
          </SidebarMenuItem>
        ))}
        <SidebarMenuItem>
          <SidebarMenuButton className="text-sidebar-foreground/
            <MoreHorizontal className="text-sidebar-foreground/
            <span>More</span>
          </SidebarMenuButton>
        </SidebarMenuItem>
      </SidebarMenu>
    </SidebarGroup>
  )
}
```

```
"use client"

import {
  BadgeCheck,
  Bell,
  ChevronsUpDown,
  CreditCard,
  LogOut,
  Sparkles,
} from "lucide-react"

import {
  Avatar,
  AvatarFallback,
  AvatarImage,
} from "@/components/ui/avatar"
import {
  DropdownMenu,
  DropdownMenuContent,
  DropdownMenuGroup,
  DropdownMenuItem,
  DropdownMenuLabel,
  DropdownMenuSeparator,
  DropdownMenuTrigger,
} from "@/components/ui/dropdown-menu"
import {
  SidebarMenu,
  SidebarMenuButton,
  SidebarMenuItem,
  useSidebar,
} from "@/components/ui/sidebar"

export function NavUser({
  user,
}: {
  user: {
    name: string
    email: string
    avatar: string
  }
}) {
  const { isMobile } = useSidebar()

  return (
    <SidebarMenu>
      <SidebarMenuItem>
        <DropdownMenu>
          <DropdownMenuTrigger asChild>
            <SidebarMenuButton
              size="lg"
              className="data-[state=open]:bg-sidebar-accent dat
            >
              <Avatar className="h-8 w-8 rounded-lg">
                <AvatarImage src={user.avatar} alt={user.name} /
                <AvatarFallback className="rounded-lg">CN</Avata
              </Avatar>
              <div className="grid flex-1 text-left text-sm lead
                <span className="truncate font-semibold">{user.n
                <span className="truncate text-xs">{user.email}<
              </div>
              <ChevronsUpDown className="ml-auto size-4" />
            </SidebarMenuButton>
          </DropdownMenuTrigger>
          <DropdownMenuContent
            className="w-[--radix-dropdown-menu-trigger-width] m
            side={isMobile ? "bottom" : "right"}
            align="end"
            sideOffset={4}
          >
            <DropdownMenuLabel className="p-0 font-normal">
              <div className="flex items-center gap-2 px-1 py-1
                <Avatar className="h-8 w-8 rounded-lg">
                  <AvatarImage src={user.avatar} alt={user.name}
                  <AvatarFallback className="rounded-lg">CN</Ava
                </Avatar>
                <div className="grid flex-1 text-left text-sm le
                  <span className="truncate font-semibold">{use
                  <span className="truncate text-xs">{user.emai
                </div>
              </div>
            </DropdownMenuLabel>
            <DropdownMenuSeparator />
            <DropdownMenuGroup>
              <DropdownMenuItem>
                <Sparkles />
                Upgrade to Pro
              </DropdownMenuItem>
            </DropdownMenuGroup>
            <DropdownMenuSeparator />
            <DropdownMenuGroup>
              <DropdownMenuItem>
                <BadgeCheck />
                Account
              </DropdownMenuItem>
              <DropdownMenuItem>
                <CreditCard />
                Billing
              </DropdownMenuItem>
              <DropdownMenuItem>
                <Bell />
                Notifications
              </DropdownMenuItem>
            </DropdownMenuGroup>
            <DropdownMenuSeparator />
            <DropdownMenuItem>
              <LogOut />
              Log out
            </DropdownMenuItem>
          </DropdownMenuContent>
        </DropdownMenu>
      </SidebarMenuItem>
    </SidebarMenu>
```

```
  )
}
```

```
"use client"

import * as React from "react"
import { ChevronsUpDown, Plus } from "lucide-react"

import {
  DropdownMenu,
  DropdownMenuContent,
  DropdownMenuItem,
  DropdownMenuLabel,
  DropdownMenuSeparator,
  DropdownMenuShortcut,
  DropdownMenuTrigger,
} from "@/components/ui/dropdown-menu"
import {
  SidebarMenu,
  SidebarMenuButton,
  SidebarMenuItem,
  useSidebar,
} from "@/components/ui/sidebar"

export function TeamSwitcher({
  teams,
}: {
  teams: {
    name: string
    logo: React.ElementType
    plan: string
  }[]
}) {
  const { isMobile } = useSidebar()
  const [activeTeam, setActiveTeam] = React.useState(teams[0])
  return (
    <SidebarMenu>
      <SidebarMenuItem>
        <DropdownMenu>
          <DropdownMenuTrigger asChild>
            <SidebarMenuButton
              size="lg"
              className="data-[state=open]:bg-sidebar-accent dat
            >
              <div className="flex aspect-square size-8 items-ce
                <activeTeam.logo className="size-4" />
              </div>
              <div className="grid flex-1 text-left text-sm lead
                <span className="truncate font-semibold">
                  {activeTeam.name}
                </span>
                <span className="truncate text-xs">{activeTeam.p
              </div>
              <ChevronsUpDown className="ml-auto" />
            </SidebarMenuButton>
          </DropdownMenuTrigger>
          <DropdownMenuContent
            className="w-[--radix-dropdown-menu-trigger-width] m
            align="start"
            side={isMobile ? "bottom" : "right"}
            sideOffset={4}
          >
            <DropdownMenuLabel className="text-xs text-muted-for
              Teams
            </DropdownMenuLabel>
            {teams.map((team, index) => (
              <DropdownMenuItem
                key={team.name}
                onClick={() => setActiveTeam(team)}
                className="gap-2 p-2"
              >
                <div className="flex size-6 items-center justify
                  <team.logo className="size-4 shrink-0" />
                </div>
                {team.name}
                <DropdownMenuShortcut>⌘{index + 1}</DropdownMen
              </DropdownMenuItem>
            ))}
            <DropdownMenuSeparator />
            <DropdownMenuItem className="gap-2 p-2">
              <div className="flex size-6 items-center justify-
                <Plus className="size-4" />
              </div>
              <div className="font-medium text-muted-foreground"
            </DropdownMenuItem>
          </DropdownMenuContent>
        </DropdownMenu>
      </SidebarMenuItem>
    </SidebarMenu>
  )
}
```

## 12. Date Range Picker

```
"use client"

import * as React from "react"
import { addDays, format } from "date-fns"
import { CalendarIcon } from "lucide-react"
import { DateRange } from "react-day-picker"

import { cn } from "@/lib/utils"
import { Button } from "@/components/ui/button"
import { Calendar } from "@/components/ui/calendar"
import {
  Popover,
  PopoverContent,
  PopoverTrigger,
} from "@/components/ui/popover"

export function DatePickerWithRange({
  className,
}: React.HTMLAttributes<HTMLDivElement>) {
  const [date, setDate] = React.useState<DateRange | undefined>(
    from: new Date(2022, 0, 20),
    to: addDays(new Date(2022, 0, 20), 20),
  })

  return (
    <div className={cn("grid gap-2", className)}>
      <Popover>
        <PopoverTrigger asChild>
          <Button
            id="date"
            variant={"outline"}
            className={cn(
              "w-[300px] justify-start text-left font-normal",
              !date && "text-muted-foreground"
            )}
          >
            <CalendarIcon />
            {date?.from ? (
              date.to ? (
                <>
                  {format(date.from, "LLL dd, y")} -{" "}
                  {format(date.to, "LLL dd, y")}
                </>
              ) : (
                format(date.from, "LLL dd, y")
              )
            ) : (
              <span>Pick a date</span>
            )}
          </Button>
        </PopoverTrigger>
        <PopoverContent className="w-auto p-0" align="start">
          <Calendar
            initialFocus
            mode="range"
            defaultMonth={date?.from}
            selected={date}
            onSelect={setDate}
            numberOfMonths={2}
          />
        </PopoverContent>
      </Popover>
    </div>
  )
}
```

## 13. Shadcn Popup Form for Adding Manual Transactions

```
"use client";

import { Button } from "@/components/ui/button";
import { Dialog, DialogContent, DialogHeader, DialogTitle, Dial
import { Input } from "@/components/ui/input";
import { Select, SelectItem, SelectTrigger, SelectValue } from '
import { useForm } from "react-hook-form";

export function AddTransactionForm() {
  const { register, handleSubmit, setValue } = useForm();

  const onSubmit = (data) => {
    console.log("Transaction Data:", data);
  };
```

```
return (
  <Dialog>
    <DialogTrigger asChild>
      <Button>Add Transaction</Button>
    </DialogTrigger>
    <DialogContent>
      <DialogHeader>
        <DialogTitle>Add a New Transaction</DialogTitle>
      </DialogHeader>
      <form onSubmit={handleSubmit(onSubmit)} className="spac
        <div>
          <label>Date</label>
          <Input type="date" {...register("date", { required:
        </div>
        <div>
          <label>Amount</label>
          <Input type="number" placeholder="Enter amount" {..
        </div>
        <div>
          <label>Name</label>
          <Input type="text" placeholder="Transaction name" {
        </div>
        <div>
          <label>Type</label>
          <Select onValueChange={(value) => setValue("type",
            <SelectTrigger>
              <SelectValue placeholder="Select Type" />
            </SelectTrigger>
            <SelectItem value="expense">Expense</SelectItem>
            <SelectItem value="income">Income</SelectItem>
          </Select>
        </div>
        <div>
          <label>Account Type</label>
          <Select onValueChange={(value) => setValue("account
            <SelectTrigger>
              <SelectValue placeholder="Select Account Type" /
            </SelectTrigger>
            <SelectItem value="cash">Cash</SelectItem>
            <SelectItem value="savings">Savings</SelectItem>
            <SelectItem value="checking">Checking</SelectItem
          </Select>
        </div>
        <div>
          <label>Category</label>
          <Select onValueChange={(value) => setValue("category
            <SelectTrigger>
              <SelectValue placeholder="Select Category" />
            </SelectTrigger>
            <SelectItem value="food">Food</SelectItem>
            <SelectItem value="transportation">Transportation<
            <SelectItem value="entertainment">Entertainment</S
          </Select>
        </div>
        <div>
          <label>Recurring Options</label>
          <Select onValueChange={(value) => setValue("recurri
            <SelectTrigger>
              <SelectValue placeholder="Select Recurrence" />
            </SelectTrigger>
            <SelectItem value="never">Never Repeat</SelectIter
            <SelectItem value="daily">Daily</SelectItem>
            <SelectItem value="weekly">Weekly</SelectItem>
            <SelectItem value="monthly">Monthly</SelectItem>
            <SelectItem value="quarterly">Quarterly</SelectIte
            <SelectItem value="annually">Annually</SelectItem
          </Select>
        </div>
        <Button type="submit">Submit</Button>
      </form>
    </DialogContent>
```

```
    </Dialog>
  );
}
```