ExTrack - Expense Tracker

1.Project Overview

1.1 Project Title & Logo

Project Title: ExTrack- – Expense Tracker

1.2 Author / Team Name

Author: Memon Furkan

1.3 Date & Version

- Date: 9 October 2025
- Version: v1.0.0

1.4 One-line Pitch

ExTrack : “A fast, secure web-based tool to record, analyze, and track your daily income and expenses for smarter budget management.”

2.Executive Summary

> 2.1-2.4

2.1 Purpose & Goals

Purpose : To help users easily track their income and expenses for better money management.

Goals:

- Add and view income or expense records quickly
- Categorize transactions for clarity
- Display total balance and monthly summaries
- Allow users to edit or delete entries easily
- Keep all data safe and user-friendly

2.2 Key Features

- Easy transaction management (add, edit, delete)
- Separate categories for income and expenses
- Summary reports by day, week, and month
- Auto-updated total balance display

2.3 Target Users

- Students who want to manage their pocket money
- Working professionals tracking monthly expenses
- Families managing household budgets
- Small business owners monitoring daily transactions

3.Problem Statement & Solution

> 3.1 - 3.3

3.1 Why Build Another Expense Tracker

- Current apps can be confusing or overloaded with features
- Many require internet accounts or paid subscriptions
- Users prefer a lightweight, easy-to-use, and secure local tracker

3.2 Pain Points of Existing Solutions

- Overloaded with unnecessary features, making them confusing
- Require registration, subscriptions, or internet access
- Slow performance and complex interfaces
- Limited control over personal data and privacy
- Hard to get clear summaries or insights quickly

3.3 Solution Overview

- ExTrack provides a simple and intuitive interface
- Users can add, edit, categorize, and delete transactions easily
- Automatic summaries and charts give quick spending insights
- All data is stored locally for privacy and security

4.Scope & Objectives

> 4.1 - 4.3

4.1 In-Scope (MVP)

- Quick entry of income and expense records

- Assign categories to each transaction
- Display balance and transaction summaries
- View reports by day, week, and month
- Mobile-friendly and clean desig

4.2 Out-of-Scope (Future Enhancements)

- Online account login and cloud backup
- Multi-user or collaborative budgeting
- Complex financial reports and analytics
- Exporting or importing data

4.3 Acceptance Criteria (Definition of Done)

- Users can record income and expenses without errors.
- Transaction categories and summaries display correctly.
- Total balance reflects all transactions accurately.
- App works offline with data stored securely in the browser

5.User-Stories & Use-Cases

> 5.1 - 5.3

5.1 Personas

- Student: Track pocket money, save for goals
- Professional: Monitor salary and monthly expenses
- Small Business Owner: Record daily income and expenses
- Family Manager: Manage household budget and plan spending

5.2 User Journey Maps

1. Open App: Access the tracker in the browser
2. Add Transaction: Enter amount, category, and date
3. View Summary: Check daily, weekly, monthly totals
4. Edit/Delete: Correct or remove entries
5. Analyze: Review spending and plan budge

5.3 Use-Case Diagram

Use Cases :

- Record income/expense
- Update transaction
- Remove transaction
- View filtered transactions
- Export reports (future)

6.Functional Requirements

> 6.1 - 6.5

6.1 CRUD Operations

- Add, edit, and delete income & expense transactions

6.2 Filtering & Sorting

- Filter transactions by type (income/expense) or category
- Sort transactions by newest, oldest, or highest/lowest amount

6.3 Summary Cards & Totals

- Display total income, expenses, and current balance
- Show quick overviews for today, this week, and this month

6.4 Export CSV

- Allow users to download all transaction data as a CSV file

6.5 Dark-Mode Toggle

- Allow users to download all transaction data as a CSV file

7.Technology Stack

> 7.1 - 7.5

7.1 Front-End

- Built with HTML, CSS, and Tailwind CSS
- Responsive design for desktop and mobile
- Interactive UI for adding, editing, and deleting transactions
- Displays summaries, balances, and filtered views in real-time

7.2 Scripting

- Pure JavaScript (no frameworks) for all functionality

7.3 Storage

- Browser Local Storage for secure, offline data

7.4 Build Tools

- VS Code, ESLint, Prettier, Lighthouse

7.5 Deployment

- Hosted on GitHub Pages or Netlify

8.System Architecture

> 8.1 - 8.3

8.1 High-Level Diagram

- User interacts with the Front-End (HTML, CSS, JavaScript) to manage transactions.
- Data is stored and retrieved from Browser Local Storage for summaries and balances.

8.2 Module Overview

- Transactions: Add, edit, delete, categorize
- Summary: Show totals and balance
- Filter & Sort: Organize transactions by date, category, or amount
- Storage: Save and load data locally
- UI: Display data in a responsive, user-friendly interface\

8.3 Data Flow

- Add transaction
- Validate input
- Store in LocalStorage
- Update UI / Re-render

9.Database Design

> 9.1 - 9.4

9.1 Entity: Expense

- Represents a single transaction, either income or expense
- Stores details like amount, category, date, and description

9.2 Attributes

- id – Unique ID for each transaction
- amount – Money added or spent
- category – Classification (e.g., Food, Salary, Transport)
- date – When the transaction occurred
- note – Additional details (optional)

9.3 JSON Schema Sample

```
{
  "id": "TX1001",
  "amount": 1200,
  "category": "Transport",
  "date": "2025-10-09",
  "note": "Taxi fare"
}
```

9.4 Local-Storage Keys & Max Size Strateg

- Storage Key: ExTrack_data
- Stored Value: JSON array containing all transaction objects
- Handling Large Data: Break into multiple keys if a single key exceeds 5MB

10.User Interface Design

> 10.1 - 10.3

10.1 Wireframes

- Dashboard with balance, summaries, and transaction list
- Forms for adding/editing transactions, plus filter and sort options

10.2 Style Guide

- Colors: Light and clean palette for readability
- Typography: Clear, legible fonts for headings and body
- Buttons & Inputs: Rounded, consistent sizing for usability
- Layout: Responsive design for desktop and mobile
- Icons & Visuals: Minimal and intuitive to support functionality

10.3 Component Library

- Buttons for add, edit, delete, and save; inputs for text, number, date, and category.
- Cards for summaries and tables/lists for transactions with sortable columns.
- Modals for confirmations and filters for date/category selection

---

11.API Specification (For Future Backend)

> 11.1 - 11.3

11.1 Endpoints (REST)

- GET /expenses – Fetch all expense and income records
- POST /expenses – Create a new transaction
- PUT /expenses/:id – Modify an existing transaction
- DELETE /expenses/:id – Delete a transaction

11.2 Request / Response Formats

- Request: amount, category, date, note for a new transaction
- Response: status, message, and transaction details including id, amount, category, date, and note

11.3 Status Codes & Error Handling

- 200 OK: Request succeeded, data returned
- 201 Created: Transaction successfully added
- 400 Bad Request: Missing or invalid input
- 404 Not Found: Transaction ID not found
- 500 Internal Server Error: Server or storage failure

12. Algorithm & Business Logic

> 12.1 - 12.3

12.1 CRUD Flowcharts

- Create: User adds transaction → Validate input → Save to LocalStorage → Update UI

- Read: Load transactions from LocalStorage → Display in list and summary cards
- Update: Select transaction → Edit details → Validate → Save changes → Refresh UI
- Delete: Select transaction → Confirm deletion → Remove from LocalStorage → Update UI

12.2 Filter & Sort Algorithms

- Filter: Select criteria → Keep matching transactions → Display results
- Sort: Choose key → Compare and order → Update displayed list

12.3 Summary Calculations

- Total Income: Sum of all income transactions
- Total Expenses: Sum of all expense transactions
- Balance: Total Income − Total Expenses
- Period Summaries: Calculate totals for daily, weekly, and monthly ranges

---

13.Folder Structure & Installation Guide

> 13.1 - 13.3

13.1 Project Structure

```
/trackify
 |
 ├── index.html        # Main HTML file
 ├── style.css         # Global CSS
 ├── script.js         # Main JavaScript file
 ├── /assets           # Images, icons, fonts
 ├── /components       # Reusable UI components (cards, modals, button
s)
```

```
├── /data            # Sample or JSON data files
└── /utils           # Helper functions (filter, sort, calculations)
```

13.2 Naming Conventions

- Files & Folders: lowercase with hyphens
- CSS Classes: kebab-case
- JavaScript Variables: camelCase
- Functions: camelCase and descriptive
- Constants: UPPERCASE with underscores

13.3 Installation & Setup

Prerequisites: Node ≥ 18, Git

Steps:

1. Clone repo: `git clone <repo-link>`
2. Install dependencies
3. Start dev server: Live Server / VS Code Go-Live
4. Build for production: minify JS/CSS, optimize assets

14.Configuration

> 14.1 - 14.3

14.1 Environment Variables

- Placeholder keys for integrating APIs or backend services in the future
- Configuration for currency, storage key, and app mode

14.2 Tailwind Config

- Custom colors, fonts, and spacing defined for consistency
- Extended theme for responsive breakpoints and utility classes
- Purge unused CSS for optimized production build

14.3 ESLint & Prettier Rules

- ESLint: Detect errors and enforce coding standards
- Prettier: Auto-format code for consistent style

15.Usage Manual

> 15.1 - 15.5

15.1 Adding an Expense

Open Trackify → Click Add → Enter details → Save

15.2 Editing / Deleting

Click Edit → Update → Save

Click Delete → Confirm removal

15.3 Filtering by Date & Category

Select date range or category → View filtered transactions

15.4 Exporting CSV

Click Export → Download data (future feature)

15.5 Switching Dark-Mode

Toggle Dark/Light mode button for better visibility

16.Testing Strategy

> 16.1 - 16.4

##

16.1 Unit Tests

- Verify CRUD operations function correctly across all modules
- Test adding, editing, and deleting transactions updates LocalStorage correctly
- Validate filter, sort, and summary calculations return accurate results
- Ensure UI reflects all changes and calculations properly

16.2 Integration Tests

- Check data flows properly from UI to LocalStorage and updates summaries

16.3 Manual QA Checklist

- Test app on multiple browsers for compatibility
- Check responsiveness on mobile and tablet devices
- Verify offline functionality and consistent UI behavior

16.4 Lighthouse Report

- Evaluate performance, accessibility, SEO, and PWA compliance
- Highlight areas for optimization and better user experience

---

17.Deployment Guide

> 17.1 - 17.4

17.1 GitHub Pages

- Upload the project to a GitHub repository
- Activate GitHub Pages in repository settings
- Access the live project via `https://username.github.io/project-name`

17.2 Netlify

- Upload the project folder to Netlify

- Set up a custom domain if required
- Netlify manages hosting, SSL, and automatic builds

17.3 CI/CD with GitHub Actions

- Automate linting, building, and deployment on each push
- Workflow sequence: lint → build → deploy
- Maintains code quality and ensures quick updates

17.4 Monitoring

- Monitor site uptime with Uptime Robot
- Track user activity and behavior using Google Analytics

18.Performance Optimization

> 18.1 - 18.4

18.1 Image Optimization

- Convert images to WebP format for smaller file size
- Implement lazy-loading to improve page load performance

18.2 CSS/JS Minification & Gzip

- Minify CSS and JavaScript for faster page loads
- Enable Gzip compression to reduce file size and improve performance

18.3 Purge Unused Tailwind Classes

- Remove unused Tailwind CSS classes to shrink bundle size
- Enhances page speed and overall performance

18.4 Lazy-load Charts

- Load charts only when they appear in the viewport

- Reduces initial load time
- Improves overall page performance

19.Maintenance & Troubleshooting

> 19.1 - 19.4

19.1 Common Build Errors & Fixes

- CSS/JS build or minification errors
- Missing dependencies or wrong file paths
- Fix by reviewing logs, reinstalling dependencies, and correcting paths

19.2 Browser Console Debug Tips

- Use `console.log` to monitor variable values
- Inspect DOM elements and attached events
- Track errors and warnings via browser developer tools

19.3 Updating Dependencies Safely

- Review available updates before applying them
- Update packages incrementally rather than all at once
- Test the application thoroughly after each update
- Lock versions in configuration files to maintain stability

19.4 Backup / Restore Local-Storage Data

- Export transaction data as JSON before clearing LocalStorage
- Import JSON to restore data into the app
- Ensures no data is lost during updates or testing

20.Future Enhancements (Road-map)

> 20.1 - 20.4

20.1 Backend & MongoDB Integration

- Shift data storage from LocalStorage to a backend database
- Enable multi-device access and real-time data syncing

20.2 User Accounts & JWT Auth

- Implement user registration and login systems
- Use JWT tokens for secure authentication and session manageme

20.3 Charts & Budget Alerts

- Integrate Chart.js to display spending trends and summaries
- Allow users to set budget limits and receive alerts when exceeded

20.4 Push Notifications

- Send alerts for overspending or upcoming bills
- Help users stay aware and manage finances effectively

20.5 Export to PDF / Excel

- Enable users to export transaction reports
- Support record-keeping and financial analysis