IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, June 2026
Impact Factor: 8.2

ExpenseIQ: Design And Implementation of a Client-Side Web-Based Personal Expense Management System

Vikash Kumar[1], Vikash Kumar[2], Hemant Kumar[3], Rajendra Singh[4]
[1,2,] Department of Computer Science and Engineering
[3] Assistant Professor, Department of Computer Science and Engineering
[4] Dean, Department of Computer Science and Engineering
Raffles University, Neemrana, Rajasthan, India
[1]vikasdehmiwal.cse@gmail.com,[2]Vikash702337@gmail.com
[3]hemantkumar@rafflesuniversity.edu.in, rajendra.singh@rafflesuniversity.edu.in

Abstract: *Personal financial mismanagement among young adults and college students remains a persistent and widely documented problem across India and globally. The absence of lightweight, privacy- preserving, zero-setup expense tracking tools that work directly in a web browser represents a significant gap in the personal finance technology landscape. This paper presents ExpenseIQ, a client- side multi-page web application for personal expense management built entirely using HTML5, CSS3, and vanilla JavaScript ES6, with data persistence implemented through the browser's localStorage API. The system provides four functional modules: a Dashboard with real-time summary statistics and Chart.js-powered doughnut chart visualization, a History module with live keyword search, category filtering, and multi-mode sorting, a Budget Tracker with per-category monthly limit setting and three-tier color-coded progress bar alerts, and an About module for project documentation. The complete application is implemented in ten files totaling approximately 65 kilobytes, requires no installation, account creation, backend server, or internet connectivity beyond the initial CDN resource load, and stores all user data exclusively in the browser's localStorage, addressing the privacy concerns associated with cloud-based personal finance platforms. Functional testing across twelve test cases confirmed correct behavior for all features. Performance analysis confirmed initial page load times under one second and interaction response times under 16 milliseconds. The system demonstrates that a fully functional, visually polished, and analytically capable personal expense management tool can be built and deployed without any server infrastructure.*

Keywords: Personal Expense Management, Web Application, localStorage API, Client-Side Storage, Data Visualization, Chart.js, Budget Tracker, HTML5, CSS3, JavaScript ES6, Responsive Web Design, Personal Finance Technology

I. INTRODUCTION

The management of personal finances is a foundational life skill whose importance is broadly recognized yet rarely practiced with any rigor by young adults, particularly those in the college student demographic. The capacity to track daily expenditures, understand categorical spending patterns, and enforce self-defined monthly budget limits is directly correlated with financial wellbeing, the accumulation of savings, and the avoidance of financial stress [1]. Despite widespread awareness of these benefits, surveys consistently show that the majority of college students and young professionals in India do not maintain any structured record of their personal expenses. The SEBI

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, June 2026 Impact Factor: 8.2

Investor Survey of 2021 found that fewer than 30% of surveyed urban individuals in India maintained any form of personal expense record [2].

The primary barriers to adoption of expense tracking practices are not motivational but technological and ergonomic: the tools available are either too complex, require account creation and bank linkage, raise privacy concerns through cloud storage of sensitive financial data, or are mobile-only applications that require installation and specific device compatibility [3]. There exists a clear and unmet need for a tool that is immediately accessible from any web browser without any setup, stores data locally on the user's device for privacy, provides visual spending analysis, and enforces budget limits — all without any server, database, subscription, or installation.

This paper presents ExpenseIQ, a web-based personal expense management system designed to address exactly this need. The system is implemented as a ten-file multi-page web application using only HTML5, CSS3, and JavaScript, with all data stored in the browser's built-in localStorage API. The paper makes the following specific contributions. First, it demonstrates a clean three-layer client- side architecture (presentation, logic, and localStorage data layer) for a multi-module web application without any framework. Second, it presents a complete localStorage-based data model for personal expense and budget management. Third, it demonstrates the integration of Chart.js for real-time spending visualization in a zero-dependency web application. Fourth, it provides a comprehensive functional test case matrix and performance analysis for a client-side personal finance application.

The remainder of this paper is organized as follows. Section 2 reviews related work in personal finance technology, client-side web storage, and data visualization. Section 3 describes the system architecture and design. Section 4 presents the implementation details. Section 5 presents test results and performance analysis. Section 6 discusses the findings and limitations. Section 7 concludes the paper and outlines future work.

II. RELATED WORK

2.1 Personal Finance Management Tools

The history of digital personal finance tools begins with Quicken (1983) and Microsoft Money (1991), desktop applications that brought automated calculation and graphical charts to household budgeting for the first time [4]. These tools established the core feature set — expense categorization, monthly totals, and spending charts — that remains the reference design for personal finance applications today. However, their constraint to a single installed desktop device limited their practical utility as mobile computing became ubiquitous.

The web-based personal finance era was defined by Mint.com (2006), which introduced automatic bank transaction import and cloud-based multi-device access [5]. Mint's approach achieved mass adoption but introduced two significant drawbacks that are directly relevant to the design of ExpenseIQ: the requirement to share banking credentials with a third-party service, and the storage of detailed transaction-level spending data on remote servers controlled by a corporation. Multiple studies have documented user concerns about both of these issues [6], [7].

In the Indian mobile app context, applications such as Walnut, ET Money, and Money View have achieved significant userbases by combining SMS transaction parsing with manual expense entry, but all require account creation, SMS access permissions, and cloud data storage [8]. The LocalCircles survey of 2023 found that 58% of Indian smartphone users expressed discomfort with sharing financial transaction data with third-party applications, establishing a clear market need for locally- stored alternatives.

2.2 Client-Side Web Storage Technologies

The Web Storage API, standardized by the W3C in 2009, provides two persistent storage mechanisms: localStorage and sessionStorage [9]. The key distinction is persistence duration: sessionStorage clears when the browser tab is closed, while localStorage persists indefinitely across sessions until explicitly cleared. The localStorage API provides four core methods — setItem(), getItem(), removeItem(), and clear() — and supports string values only, requiring JSON serialization for structured data objects.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, June 2026 Impact Factor: 8.2

Bidelman [10] established the foundational best practices for localStorage usage in web applications, including JSON serialization patterns, quota management, and key namespace conventions. These practices are directly applied in ExpenseIQ's shared utility layer. The 5 MB minimum localStorage quota across all major browsers is more than sufficient for personal expense data: a user recording five expenses per day for an entire year would accumulate approximately 1,825 records occupying approximately 274 KB — well under 6% of the available quota.

IndexedDB, the more powerful client-side storage API supporting structured data, indexing, and transactions, was considered for this project but rejected in favor of localStorage. The simpler localStorage API is entirely sufficient for the expected data volumes of an individual user, provides a significantly simpler programming model, and is universally familiar to developers. The architectural abstraction of all storage access through utility functions ensures that a migration to IndexedDB in a future version would require changes only in the data access layer.

2.3 Data Visualization for Personal Finance

The role of data visualization in personal financial management goes beyond aesthetics. Treisman and Gelade [11] established that preattentive visual features — color, size, shape — are processed automatically in under 200 milliseconds, far faster than analytical reading of numerical tables. This means a well-designed spending chart communicates categorical distribution to the viewer almost instantaneously, providing a qualitatively different level of financial insight than a list of numbers.

Kirk [12] identifies proportional area charts (pie charts and doughnut charts) as the optimal visualization for categorical composition data — data that shows how a whole is divided among named categories. The doughnut variant is specifically recommended for dashboard contexts where the hollow center can be used to reduce visual weight and where the reduced ink-to-data ratio improves clarity. Both considerations apply directly to the ExpenseIQ dashboard chart design.

Chart.js, introduced by Downie in 2013, has become one of the most widely adopted open-source charting libraries for web applications [13]. Its Canvas-based rendering provides smooth animation, consistent cross-browser appearance, and responsive sizing without any server-side processing. The library's simple configuration API makes it well-suited to academic projects where development efficiency and code clarity are as important as visual output quality.

2.4 Responsive Web Design

Marcotte [14] introduced the concept of responsive web design in 2010, establishing the three- component model — fluid grids, flexible images, and CSS media queries — that remains the foundation of multi-device web design. The subsequent introduction of CSS Grid (CSS Grid Layout Module Level 1, W3C 2017) and the widespread browser adoption of CSS custom properties have significantly simplified the implementation of responsive layouts, making it practical to build production-quality responsive interfaces without any CSS framework.

The ExpenseIQ interface uses CSS Grid for its major layout structures (the summary card grid, the two-column main content layout) and CSS Flexbox for component-level alignment (sidebar navigation items, expense item rows, form row pairs). Media queries at 900px and 680px handle the transition from four-column to two-column cards and from persistent sidebar to hamburger menu navigation respectively.

III. SYSTEM ARCHITECTURE AND DESIGN

3.1 Overall Architecture

ExpenseIQ is implemented as a client-side multi-page application. The architecture consists of three layers: a Presentation Layer of four HTML files, an Application Logic Layer of five JavaScript files, and a Data Layer implemented in the browser's localStorage API. Figure 1 shows the relationship between these layers.

The four HTML files — index.html (Dashboard), history.html (History), budget.html (Budget), about.html (About) — share a single CSS stylesheet (style.css) ensuring visual consistency. Each HTML file loads two

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, June 2026 Impact Factor: 8.2

JavaScript files: the shared utility module (shared.js) and a page-specific controller. Navigation between pages is implemented through standard HTML anchor links, requiring no JavaScript router and ensuring correct operation when files are opened directly from the filesystem.

All data access is abstracted through four utility functions in shared.js: getExpenses(), saveExpenses(), getBudgets(), and saveBudgets(). This abstraction ensures that if the storage mechanism is changed in a future version, only these four functions need modification.

3.2 Data Model

The system uses two primary localStorage keys. The key 'expenses' stores a JSON-serialized array of expense objects. Each expense object contains five properties: id (integer, assigned as Date.now() for uniqueness), name (string, user-provided description), amount (float, positive non-zero value in Indian Rupees), category (string, one of seven defined values), and date (string, YYYY-MM-DD format). The key 'budgets' stores a JSON-serialized object mapping category name strings to monthly budget limit integers. Two secondary keys, 'dev-name' and 'dev-roll', store the developer's display name and roll number for the About page.

The seven expense categories are: Food, Transport, Shopping, Entertainment, Health, Education, and Other. Each category is associated with a specific hex color code used consistently across the doughnut chart, expense item badge, and budget progress bar displays.

3.3 Module Design

The Dashboard module provides the primary user interaction point. Its responsibilities include displaying four real-time summary metric cards (Total Spent, This Month, Total Entries, Top Category), rendering the Chart.js doughnut chart, providing the expense entry form with validation, and displaying the five most recently added expenses.

The History module provides comprehensive data access with filtering, search, and sort capabilities. It applies user-selected filters (keyword search, category filter, sort mode) to the expense array in memory andre-renders the filtered list and derived statistics on every filter change event.

The Budget module enables per-category monthly budget management. It computes the current month's spending per category by filtering the expense array to the current calendar month and year, compares these figures to stored budget limits, and renders progress bars with threshold-based color alerts.

The About module displays project metadata, an editable developer profile stored in localStorage, a technology stack overview, and a feature summary — serving as an embedded project documentation page for academic evaluation.

3.4 User Interface Design

The visual design system is defined through CSS custom properties in the :root selector, establishing a dark-theme color palette anchored by a deep near-black background (#0f0f13), a warm gold accent color (#f5c542) for monetary figures and interactive elements, and coral red (#ff6b6b) for delete actions and over-budget alerts. The typography system uses DM Serif Display for titles and monetary amounts — providing visual authority and distinctiveness — and DM Sans for all body text and interface labels, providing excellent small-size readability.

The layout uses CSS Grid with grid-template-columns: repeat(4, 1fr) for the summary card row, collapsing to two columns at 900px, and grid-template-columns: 1fr 1fr for the two-column main content area, collapsing to a single column at 900px. The sidebar navigation occupies a fixed 240px on the left side on desktop screens above 680px and converts to a hamburger-triggered slide-in panel on smaller screens, implemented through a CSS transform transition.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, June 2026
Impact Factor: 8.2

IV. IMPLEMENTATION

4.1 Shared Utility Layer

The shared.js file provides eight functions and one constant used across all four page controllers. The getExpenses() and saveExpenses() functions abstract all expense array localStorage access: getExpenses() calls localStorage.getItem('expenses'), parses the JSON result, and returns the array or an empty array if the key is absent; saveExpenses() serializes the expense array to JSON and calls localStorage.setItem(). The same pattern is applied in getBudgets() and saveBudgets() for the budget object.

The fmt() utility formats numeric values as Indian Rupee strings (□X.XX). The formatDate() utility converts YYYY-MM-DD strings to human-readable 'en-IN' locale date strings. The showToast() function manages the toast notification display, adding and removing a CSS class to trigger a CSS transition-based fade-in/fade-out animation. The expenseItemHTML() function generates reusable expense item HTML strings used in both the Dashboard and History page renderers.

4.2 Dashboard Implementation

The dashboard.js controller implements the addExpense() function, which validates the four form inputs, constructs the expense object with a Date.now() id, prepends it to the expense array using Array.unshift(), saves to localStorage, resets the form, and calls renderDashboard(). The renderDashboard() function computes the four summary statistics through array operations: total via Array.reduce(), monthly total via Array.filter() and Array.reduce(), top category via grouping and Object.entries().sort(), and entry count via Array.length. It then updates the four card elements, calls renderChart() with the category totals, and renders the five mostrecent expenses.

The renderChart() function initializes a Chart.js doughnut chart instance with category-specific colors from the CAT_COLORS constant, custom tooltip formatting showing rupee amounts, and a bottom- positioned legend styled to match the dark theme. The chart instance is destroyed and recreated on every render call to prevent the Chart.js "canvas already in use" error.

4.3 History Implementation

The history.js renderHistory() function reads filter values from the three filter DOM elements on every call. It applies them to the expense array through chained Array.filter() and Array.sort() calls: keyword search applies a case-insensitive String.includes() test on the name property; category filter applies a strict equality test on the category property; the four sort modes use comparison functions for date string comparison (ascending and descending) and numeric amount comparison (ascending and descending). The four statistics cards are computed from the filtered array through reduce() operations for total and max and standard division for average.

4.4 Budget Implementation

The budget.js renderBudget() function filters all expenses to the current calendar month using Date object month and year comparisons, then groups the filtered expenses by category using an accumulator object pattern. For each entry in the budgets object, it computes pct = Math.min((spent / limit) * 100, 100) for the progress bar fill width, determines the bar color using a conditional expression (red if spent > limit, orange if pct >= 80, category color otherwise), and generates the progress bar item HTML as a template literal string. The overview panel aggregates total budget set, total spent across budgeted categories, and total remaining through Object.keys().reduce() operations.

4.5 Implementation Summary

The complete implementation consists of 10 files: 4 HTML files (~250 lines total), 1 CSS file (~300 lines), and 5 JavaScript files (~350 lines). The total codebase size is approximately 900 lines across all files. External dependencies are limited to Chart.js 3.x (loaded from jsDelivr CDN, ~200 KB, Dashboard page only) and Google Fonts (DM

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, June 2026 Impact Factor: 8.2

Serif Display and DM Sans, ~65 KB combined, cached after first load). All other functionality is implemented in vanilla HTML, CSS, and JavaScript with no framework, build tool, or package manager dependency.

V. TESTING AND RESULTS

5.1 Functional Test Results

The system was evaluated against a structured set of twelve functional test cases covering all major features and edge cases. Tests were performed manually in Google Chrome 120 on Windows 11 desktop and in Chrome's mobile device emulation mode at 375px viewport width.

Table 1 summarizes the complete test results. All twelve test cases passed, confirming correct implementation of expense addition with all four validation rules, expense deletion with statistics update, History page filtering, searching, and sorting, Budget page progress bar rendering including the three threshold-based color states, localStorage persistence across page reloads, and About page developer info editing with localStorage persistence.

Table 1: Functional Test Case Results

TC-ID	Feature Tested	Action	Expected	Result
TC-01	Valid expense addition	All fields filled,clickAdd	Expense added, stats update	PASS
TC-02	Empty name validation	Nameblank, clickAdd	Error toast, no addition	PASS
TC-03	Zero amount validation	Amount = 0,clickAdd	Error toast, no addition	PASS
TC-04	Negativeamount validation	Amount = -50,clickAdd	Error toast, no addition	PASS
TC-05	Expense deletion	Click× onexpenseitem	Expenseremoved,stats update	PASS
TC-06	Category filter in History	Select "Food"filter	Only Food expenses shown	PASS
TC-07	Keyword search inHistory	Type"lunch" insearch	Matching expenses shown	PASS
TC-08	Sort by HighestAmount	Selectsort mode	Expenses sorted correctly	PASS
TC-09	Budget setting	Enter Food budget □3000	Progress bar renders correctly	PASS
TC-10	Over-budget detection	Food spend > Food budget	Red bar, overageshown	PASS
TC-11	localStorage persistence	Addexpenses, reload page	All data retained	PASS
TC-12	Developer info editing	Edit name onAbout page	Nameupdates, persists	PASS

5.2 Performance Analysis

Performance was measured across four dimensions. For application size, the total size of all ten application files is approximately 65 KB, providing an extremely fast filesystem load time under 500 milliseconds. The Chart.js CDN dependency adds approximately 200 KB on first load, cached by the browser for all subsequent visits.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, June 2026 Impact Factor: 8.2

For interaction response time, all user interactions — expense addition, deletion, filter application, sort change, budget save — produce visible UI updates in under 16 milliseconds on modern desktop hardware, meeting the threshold for "instant" response as defined by Nielsen [15]. These fast response times are a direct consequence of the purely in-memory nature of all data operations: there are no network requests, no file I/O, and no computationally complex algorithms.

For storage efficiency, a single expense record serialized to JSON occupies approximately 150 bytes. A user recording five expenses per day for a full year generates approximately 274 KB of localStorage data, representing under 6% of the 5 MB minimum localStorage quota guaranteed by all major browsers.

For cross-browser compatibility, the application was confirmed to function correctly on Google Chrome 120, Mozilla Firefox 121, Microsoft Edge 120 (all on Windows 11 desktop), Safari 17 on macOS Sonoma, and mobile Chrome and Safari on Android 13 and iOS 17 respectively. No functional differences were observed across any tested platform.

Table 2: Performance Metrics

Metric	Measured Value	Threshold	Status
Total application file size	~65 KB	< 500 KB	Good
Initial page load (local filesystem)	<500 ms	<1000 ms	Good
Initialpageload (CDN, firstvisit)	800-1200 ms	< 2000 ms	Good
Expense addition response time	<16 ms	<100 ms	Excellent
Filter100 expenses in History	< 5 ms	<100 ms	Excellent
localStorage for1 year ofdaily data	~274 KB	<5000 KB	Excellent
Cross-browser compatibility	All tested	Chrome, Firefox, Edge,Safari	Pass

VI. DISCUSSION

6.1 Achievements and Strengths

The ExpenseIQ system successfully demonstrates that a fully functional, visually polished, and analytically capable personal expense management system can be implemented without any server infrastructure, framework dependency, or external database. The ten-file implementation is self- contained, immediately runnable, and comprehensible as a complete codebase — all properties that are valuable both for academic demonstration and for practical adoption as a personal finance tool.

The localStorage-based data persistence approach addresses the privacy concerns that are the primary barrier to adoption of cloud-based personal finance tools among privacy-conscious users. All financial data remains exclusively on the user's device and is never transmitted to any server, making the system appropriate for users who are unwilling to share spending data with third-party platforms.

The Chart.js doughnut chart integration provides the instant categorical spending visualization that is qualitatively unavailable from numerical records alone. The chart updates automatically on every expense addition and deletion, providing continuous visual feedback that reinforces spending awareness — the primary mechanism through which expense tracking is believed to influence financial behavior [1].

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, June 2026 Impact Factor: 8.2

The three-tier budget alert system (green, orange, red) implements the behavioral finance principle that visual feedback calibrated to urgency is more effective at influencing behavior than neutral numerical displays. The color progression from green through orange to red exploits universal color- danger associations to communicate budget status without requiring the user to interpret numerical percentages.

6.2 Limitations

The most significant technical limitation of the current implementation is the device-specificity of localStorage storage. Expense data recorded in one browser on one device is not accessible from other browsers or devices. This means a user who switches from their laptop to their smartphone, or who clears their browser data, loses access to their expense history. For a personal expense tracker intended to be used across multiple devices or over extended time periods, this is a meaningful functional limitation.

The second limitation is the absence of data backup capability. There is currently no export feature that would allow users to create a backup of their expense data to an external file. A browser data clear or device failure would result in permanent loss of all recorded expense data. A CSV export feature, implementable entirely in client-side JavaScript, would address this limitation with minimal additional complexity.

The third limitation is the single-user, single-device architecture. The current data model has no concept of a user identity, making it unsuitable for shared use by multiple household members or for use across different browser profiles on the same device.

6.3 Comparison with Existing Tools

Compared to cloud-based personal finance platforms such as Mint or ET Money, ExpenseIQ trades multi-device synchronization and automatic transaction import for complete privacy, zero setup friction, and offline functionality. This trade-off is appropriate for the target user — a student or individual user who wants a private, immediately usable expense tracker rather than a comprehensive connected finance platform.

Compared to spreadsheet-based expense tracking, ExpenseIQ provides significantly faster data entry (5–10 seconds per expense versus 30–60 seconds), automatic visualization (no chart setup required), and budget enforcement (not possible in a basic spreadsheet without formulas and conditional formatting). The trade-off is that a spreadsheet provides more flexibility for custom analysis, but this flexibility is rarely exercised by casual expense trackers.

VII. CONCLUSION AND FUTURE WORK

This paper has presented ExpenseIQ, a client-side web-based personal expense management system that addresses the fundamental limitations of existing personal finance tools for individual users who prioritize simplicity, privacy, and zero setup friction. The system provides a complete four-module personal finance workflow — expense recording, historical analysis, budget enforcement, and project documentation — in a ten-file, 65 KB web application that runs entirely in any modern browser without installation, account creation, or internet connectivity.

The implementation demonstrates several technically significant results. The three-layer client-side architecture with localStorage data abstraction provides a clean, maintainable, and extensible structure for a multi-page web application without any framework. The localStorage-based persistence approach is shown to be technically and practically adequate for individual user data volumes. The Chart.js integration provides professional-quality spending visualization with fewer than 30 lines of configuration code. All twelve functional test cases passed and all performance metrics met their thresholds.

Future development directions include backend integration with a relational or document database to enable multi-device data access, user authentication for multi-user support, income tracking to enable net savings calculation, CSV and PDF export for data backup and external analysis, Progressive Web App conversion for offline capability and home screen installation, and an advanced analytics module with month-over-month comparison charts and spending trend analysis. These enhancements would extend ExpenseIQ from a personal utility tool into a production-quality

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, June 2026
Impact Factor: 8.2

personal finance platform while preserving the simplicity and privacy-first design philosophy of the current implementation.

REFERENCES

1. E. T. Garman and R. Forgue, Personal Finance, 12th ed. Cengage Learning, 2014.
2. Securities and Exchange Board of India, "SEBI Investor Survey 2021," SEBI Publications, Mumbai, India, 2021.
3. National Institute of Financial Education, "Youth Financial Literacy in India: Survey Report," NIFE Publications, New Delhi, India, 2022.
4. S. Kiplinger, "The history of personal finance software," Kiplinger Personal Finance, vol. 48, no. 3,pp. 72–75, 1994.
5. J. Kagan, "Digital personal finance tools and user retention: A comparative study," Journal of Personal Finance, vol. 19,no. 1,pp. 45–62, 2020.
6. R. Garg and S. Williams, "Visual design and perceived stress in personal finance applications," International Journal of Human-Computer Studies, vol. 154,p. 102677, 2021.
7. LocalCircles, "Indian Consumer Data Privacy Survey," LocalCircles Research Publications, New Delhi, India, 2023.
8. P. Sharma, "Mobile personal finance applications in India: Adoption barriers and user behavior," Journal of Financial Technology, vol. 7,no. 2,pp. 113–128, 2022.
9. World Wide Web Consortium, "Web Storage Specification," W3C Recommendation, 2009. [Online]. Available:https://www.w3.org/TR/webstorage/
10. E. Bidelman, "The past, present and future of local storage for web applications," HTML5 Rocks, 2010. [Online]. Available:https://www.html5rocks.com/en/tutorials/offline/storage/
11. A. M. Treisman and G. Gelade, "A feature-integration theory of attention," Cognitive Psychology, vol. 12,no. 1,pp. 97–136, 1980.
12. A. Kirk, Data Visualisation: A Handbook for Data Driven Design, 2nd ed. SAGE Publications, 2019.
13. N. Downie, "Chart.js: Simple yet flexible JavaScript charting," 2013. [Online]. Available: https://www.chartjs.org/
14. [E. Marcotte, "Responsive web design," A List Apart, no. 306, 2010. [Online]. Available: https://alistapart.com/article/responsive-web-design/
15. J. Nielsen, "Response times: The three important limits," in Usability Engineering. Academic Press, 1993,pp. 135-137.
16. World Wide Web Consortium, "CSS Grid Layout Module Level 1," W3C Recommendation, 2017. [Online]. Available:https://www.w3.org/TR/css-grid-1/
17. Mozilla Developer Network, "Web Storage API — localStorage," MDN Web Docs, 2024. [Online]. Available:https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
18. P. M. Fitts, "The information capacity of the human motor system in controlling the amplitude of movement," Journal of Experimental Psychology, vol. 47,no. 6,pp. 381-391, 1954.
19. J. Nielsen, "Ten usability heuristics," Nielsen Norman Group, 1994. [Online]. Available:
20. https://www.nngroup.com/articles/ten-usability-heuristics/