

# TrackXpense: A Secure, Multilingual, and Insight-Driven Personal Expense Tracking Web Application

Mr. V.V. Mali, Chaitanya D. Mane, Sumit L. Jagdale, Saniya N. Mujawar, Sakshi S. Pawar

1. *Lecturer, computer technology, AITRC vita*
2. *student, computer technology, AITRC vita*
3. *student, computer technology, AITRC vita*
4. *student, computer technology, AITRC vita*
5. *student, computer technology, AITRC vita*

\*\*\*

**Abstract:** Keeping track of personal expenses sounds simple, but for most people it's either too tedious to do manually or too risky when handed off to a cloud service they don't fully trust. TrackXpense is a web-based expense tracking application we built to bridge that gap — it runs on PHP and MySQL with a plain JavaScript frontend, and it's designed to be secure, easy to use, and genuinely useful from day one. Unlike most published systems in this space, we treated security as a core design requirement rather than something added at the end. That means proper CSRF protection, a dual-vector rate limiter that blocks both targeted account attacks and broad IP-based ones, and session fixation prevention on every login. On the analytics side, we wanted the system to give users meaningful spending insights without needing machine learning or a large dataset to warm up on — so we built recurring expense detection and month-over-month surge alerts entirely in SQL. The app also supports three languages — English, Hindi, and Marathi — which matters a lot for the users we built this for. We tested all of these features thoroughly and confirmed they hold up both under normal use and under deliberate attack scenarios.

**Keywords** — Expense Tracker, Personal Finance Management, Web Security, CSRF, Rate Limiting, SQL Analytics, Multilingual UI, PHP, MySQL, JavaScript, Budget Tracking

## I.INTRODUCTION

Most people know they should track their spending. The tricky part is finding a tool that actually fits into daily life — one that's fast to use, doesn't require handing over financial data to some third-party cloud platform, and gives back something useful in return. Spreadsheets work for a while, but they don't scale. Commercial apps are often feature-heavy but subscription-gated, or they quietly sync everything to remote servers.

TrackXpense started from a simple premise: a self-hosted PHP and MySQL web application, with no JavaScript framework involved, should be able to do everything a good personal finance tracker needs — secure login, categorized expense recording, visual reports, data export, and spending

insights — without requiring a machine learning pipeline, a cloud subscription, or a build system.

The problems fell into a few clear groups. Security — most reviewed systems are vulnerable to CSRF attacks, brute-force

login, and cross-user data access by design, not by accident. Analytics — the papers that do attempt smart insights require ML infrastructure that takes weeks to set up and months of user data before it produces anything useful. Accessibility — almost no published implementation handles anything other than English.

We built TrackXpense to address all four of these areas in a single, self-contained application. The rest of this paper walks through what we built, how it compares to existing work, the design decisions behind each feature, and what we learned along the way.

## II.BODY OF PAPER

TrackXpense is a secure and user-friendly web-based personal expense tracking application developed using PHP, MySQL, and JavaScript. The main objective of the system is to help users manage their daily expenses efficiently while maintaining strong security and usability. The application allows users to record expenses, categorize spending, generate reports, and analyze financial patterns. Unlike many existing expense tracking systems, TrackXpense focuses on integrating strong security mechanisms such as CSRF protection, dual-vector rate limiting, session fixation prevention, and secure authentication. These features ensure that user data remains protected from common web vulnerabilities.

The system also provides useful financial insights without relying on complex machine learning models. Instead, SQL-based analytics are used to detect recurring expenses and identify sudden increases in monthly spending. This approach allows the system to generate meaningful insights even with a small amount of user data. Another important feature of TrackXpense is its multilingual interface, which supports English, Hindi, and Marathi languages. This improves accessibility for users who prefer regional languages.

## III.LITERATURE REVIEW

**Where Existing Systems Start:**

The earliest and most commonly cited implementations in this space — including the systems documented by Kaushik et al. [2], Gehlot et al. [4], and the ResearchGate Expense Tracker Management System [1] — all share the same basic architecture. There's a MySQL database storing expense records, a PHP backend handling CRUD operations, and a frontend rendering some kind of chart. These systems work, and they were genuinely useful as starting points for understanding the problem space. But they all treat categories as a fixed list that everyone shares, and none of them go beyond date-filtered summaries for analytics. More critically, the security model in almost all of them ends at `session_start()` and `password_hash()`.

### The Security Gap:

Going through Papers 1 to 8 carefully, we couldn't find a single one that implements CSRF token validation. That means every form submission — adding an expense, deleting a record, updating a budget — could theoretically be triggered by a malicious page the user happens to have open in another tab. Rate limiting on login forms is similarly absent across the board. Budget Buddy [14] mentions it as a planned feature, but no implementation details are given. These aren't edge-case academic concerns — they're in the OWASP Top Ten and have been for over two decades.

### Machine Learning for Insights — A Double-Edged Sword:

A handful of papers take a more ambitious approach to analytics. Aishwarya and Hemalatha [11] use clustering to auto-categorize expenses, Thakare et al. [12] apply a Naive Bayes classifier, and Lingayat et al. [15] explore SVM and Random Forest for spending predictions. The results in these papers are genuinely interesting, but they come with a significant practical cost. All of them require labeled training data, a model training pipeline, and a minimum amount of user history before they produce any useful output — what's typically called the cold-start problem. SpendWise [13] acknowledges this limitation explicitly. For a personal finance app where a new user might have recorded only ten expenses, ML-based insights are effectively unavailable for the first several weeks of use.

### The Language Problem:

Nineteen of the twenty papers we reviewed are English-only. Money Map [9] lists multilingual support as a future goal. That's the entirety of what the literature has to offer on this front. Given that a large share of potential users in India are significantly more comfortable reading and interacting in regional languages, this is a real accessibility gap. The 2022 study on personal financial management behavior [17] points out explicitly that language barriers are one of the top reasons Indian users don't adopt digital finance tools, yet the published implementations have not responded to that finding.

### Small but Important UX Gaps:

No reviewed paper implements a dark theme, let alone handles the engineering detail of preventing the screen from briefly flashing the wrong theme when the page loads. Export

functionality exists in a few papers [1, 3, 7], but all of them use a direct browser link to a PHP file — meaning if the user's session has expired, the browser downloads a PHP error response as a CSV file instead of the actual data. These are the kinds of small details that separate a functional prototype from something that actually holds up in daily use.

## IV.METHODOLOGY

### Overall Architecture:

TrackXpense follows a straightforward three-tier design. The frontend consists of five HTML pages — authentication, dashboard, add expense, reports, and settings — each with a dedicated JavaScript file handling its logic. The backend is a set of PHP endpoints, each corresponding to a specific operation: fetching summary data, inserting an expense, exporting CSV, and so on. These endpoints communicate via JSON. The database layer is a MySQL schema with five tables: users, expenses, settings, categories, and login\_attempts. The database connection is a single PDO instance shared across all endpoints, configured with exception-mode error handling and prepared statement execution to prevent SQL injection.

One design decision worth noting early is that all frontend-to-backend communication goes through a single JavaScript wrapper function rather than calling `fetch()` directly. That function handles CSRF token injection on POST requests, and it intercepts 401 responses globally and redirects to the login page. This means session expiry is handled consistently across every feature without any individual page needing to worry about it.

### Security Design:

The security architecture was designed so that each layer independently addresses a different attack surface, and so that the layers interlock rather than just stacking. CSRF tokens are generated server-side using cryptographically secure random bytes, stored in the session, and validated on every POST using a constant-time comparison function that doesn't leak information through response timing. Rate limiting uses a database-backed sliding window, but it tracks the email identifier and the IP address independently — so a credential stuffing attack that rotates email addresses is caught by the IP counter, one the moment login succeeds. Every delete operation includes the authenticated user's ID in the WHERE clause, so even a correctly authenticated user cannot affect another user's records.

### Analytics Without Machine Learning:

We wanted to give users genuinely useful spending insights without asking them to wait for a model to warm up. The solution we arrived at was to express both insight features as SQL queries. The recurring expense detector groups the past six months of transactions by product name and category, then surfaces combinations that appear three or more times — along with how often they've occurred, the average amount, and when they last appeared. This gives users a reasonably accurate picture of upcoming recurring costs from the very

first week of use. The dual threshold is important — without the absolute amount floor, a category where the user spent twenty rupees last month and forty this month would trigger a fifty-percent surge alert, which isn't meaningful.

### Internationalization Design:

The translation system stores a full dictionary for all three languages directly in the JavaScript file that loads on every page. When the user switches language, the engine walks the DOM looking for elements with a data-i18n attribute and swaps in the right string — no page reload, no server request. Placeholder text is handled separately through a data-i18n-placeholder attribute. The selected language is saved in localStorage, so returning users see their preferred language immediately. We also wired the PDF export function to check the active language before proceeding, because jsPDF can't render Devanagari script — the characters come out as empty boxes. If Hindi or Marathi is active, the user sees a warning and can choose to switch to CSV instead.

## V.IMPLEMENTATION DETAILS

### Database and Connection Layer:

The schema is set up with ON DELETE CASCADE on all foreign keys pointing to the users table, so removing a user account automatically cleans up all associated expenses, settings, and categories. The expenses table has a composite index on (user\_id, date) since virtually every query filters on both of those columns. The login\_attempts table is indexed on (identifier, attempted\_at) to keep the window-check queries fast even as the table grows. The PHP connection wrapper was deliberately not given a global JSON content-type header, because the CSV export endpoint needs to set its own content-type — a small thing, but it caused a real bug in an earlier version.

### Authentication Flow:

Registration validates the email format, checks that the username is within fifty characters, and enforces a password policy requiring eight to fifteen characters with at least one uppercase letter, lowercase letter, digit, and special character. All of this is checked on the server regardless of what the frontend sends. On success, the new user's row is inserted along with a default settings row and seven starting categories — Food, Transport, Bills, Shopping, Entertainment, Health, and Other — so they can start recording expenses

immediately without configuring anything. The login flow checks rate limits for both the email and IP before touching the users table at all. If either limit is exceeded, the request is rejected before any database password query runs. On successful login, the session ID is regenerated immediately.

### Dashboard and Expense Logic:

The dashboard summary endpoint runs three queries: today's total, the current month's total, and the user's settings row. From those it computes the remaining budget and whether the user is overspent. It returns the current server date and a boolean indicating whether today is the last day of the month. The frontend uses these server-provided flags for all budget alert logic and never uses the client's JavaScript Date object for anything time-sensitive. When a user is overspent, the remaining budget widget gets a visual indicator showing exactly how much over budget they are — not just a neutral zero.

### Reports, Charts, and Export:

The reports page shares a single getFilters() function across the pie chart, bar chart, table, CSV export, and PDF export. Whatever filter combination the user has selected is what all of those outputs reflect. CSV export works by fetching the data through the authenticated wrapper, creating a Blob in memory, and triggering a download — no direct browser navigation to the PHP file. This means session expiry during export results in a redirect to login rather than a corrupt download. The PDF export uses jsPDF's autoTable plugin on the currently-rendered HTML table, which means it reflects exactly what the user sees on screen.

### Theme and Language System:

The theme-lang.js file is the only script on every page that loads without the defer attribute. It runs an immediately invoked function at the top that reads the saved theme from localStorage and applies it to the document root before the browser renders any visible content. Without this, loading the page with a saved dark theme would briefly show a white page before the JavaScript runs and corrects it. The same file defines every shared utility the app uses — the fetch wrapper, the logout modal, the reusable confirmation dialog, the chart color helper, and the hamburger menu behavior. Centralizing all of this means that if any of it needs to change, there's exactly one place to change it.

**Table 1: Feature Comparison — TrackXpense vs. Reviewed Systems**

| Feature / Capability                        | Basic | ML-Based [11,12,15] | PFM Apps [6–10] | TrackXpense (Ours) |
|---------------------------------------------|-------|---------------------|-----------------|--------------------|
| CSRF Protection with Timing-Safe Validation | ✗     | ✗                   | ✗               | ✓                  |
| Dual-Vector Rate Limiting (Email + IP)      | ✗     | ✗                   | ✗               | ✓                  |
| Session Fixation Prevention on Login        | ✗     | ✗                   | ✗               | ✓                  |

|                                              |   |   |   |   |
|----------------------------------------------|---|---|---|---|
| Ownership Check in DELETE Operations         | X | X | X | ✓ |
| XSS-Safe DOM Rendering                       | X | X | X | ✓ |
| Multilingual Support (EN / HI / MR)          | X | X | X | ✓ |
| Dark + Light Theme with FOUC Prevention      | X | X | X | ✓ |
| Smart Insights — No ML Infrastructure Needed | X | ✓ | X | ✓ |
| Month-over-Month Surge Detection via SQL     | X | X | X | ✓ |
| Server-Side Budget Date Logic                | X | X | X | ✓ |
| Per-User Custom Category Management          | X | X | X | ✓ |
| Session-Safe CSV Export via Blob             | X | X | X | ✓ |
| PDF Warning for Devanagari Unicode           | X | X | X | ✓ |
| Styled Confirmation Modal                    | X | X | X | ✓ |
| Zero Framework / Zero Build Step             | ✓ | X | X | ✓ |

✓ = Implemented    X = Not Present or Not Documented in the Paper

## FUTURE SCOPE

### Smarter Predictions with Machine Learning:

The SQL-based analytics in the current version work well for pattern detection, but they can't really project forward in a probabilistic way. A sensible next step would be adding a lightweight ML layer — something like a simple regression model trained on a user's own expense history — that could estimate likely spending for the coming month by category. This would need to be done carefully to avoid the cold-start problem that affects the approaches in Papers 11, 12, and 15 — probably by starting with the SQL-based estimates and gradually shifting toward the model as more data accumulates.

### UPI and Bank Statement Import:

Manual entry is the single biggest friction point in any expense tracker. Connecting to India's UPI ecosystem through the NPCI API would allow automatic import of transaction records, removing the need for users to enter anything at all for most purchases. A fallback option for users who prefer not to connect their accounts would be a PDF statement parser that can extract transaction rows from the statements most Indian banks offer for download — this could be implemented as a server-side PDF processing step using existing PHP libraries.

### Native Mobile App:

The current UI is mobile-responsive and usable in a phone browser, but it can't do things a native app can — no push notifications when a budget threshold is crossed, no quick-entry widget on the home screen, no offline caching for areas with spotty connectivity. A Flutter or React Native app backed by the same PHP API, with SQLite used for local caching, would cover all of these. The API layer is already clean enough that building a mobile client on top of it wouldn't require significant backend changes.

### More Regional Languages:

Adding Marathi was a deliberate choice given our target audience, but India has twenty-two official languages and the translation architecture is set up to accept new language objects with minimal changes. Tamil, Telugu, Kannada, Bengali, and Gujarati together would cover the majority of the remaining regional user base. The main work involved is translation and testing — the underlying engine doesn't need to change at all.

### Shared Household Accounts:

Right now each account belongs to one person, and data is strictly isolated. For couples or families managing shared finances, it would be useful to have a shared pool where multiple users can contribute records while keeping some categories private. This would require adding an account

groups table, a roles layer distinguishing owners from contributors, and changes to the query layer to aggregate across multiple user IDs within the same group.

### Richer Visualizations:

The current dashboard shows a line chart and a financial summary. There's a lot of room to expand — a spending heatmap calendar, a month-end projection based on the current daily rate, year-on-year category comparisons, and a savings runway indicator showing how long the current savings balance would last given the current monthly spend rate. All of these can be built on top of the existing data model without schema changes.

## I.RESULTS AND DISCUSSION

### Security Testing:

We tested each security mechanism directly rather than assuming it worked. For CSRF, we sent POST requests to every state-changing endpoint without the token header and confirmed that all of them returned a 403 and made no database changes. We then sent the same requests with a plausible but incorrect token and confirmed the same result. For rate limiting, we submitted five failed login attempts and confirmed lockout, then tried a different email from the same IP and confirmed the IP counter was also tracking independently. We tested session fixation by recording the PHP session ID before and after login and confirming they differed. For cross-user data access, we logged in as User A, retrieved the ID of an expense belonging to User B, and sent a delete request

— confirming it was blocked. For XSS, we created an expense with a product name containing a script tag and confirmed it displayed as literal text in the reports table.

conditions were met simultaneously. Response times for both queries were in line with other dashboard queries, even on datasets with several hundred rows.

### Theme and Language:

We tested FOUC prevention by throttling CPU speed in browser developer tools to slow down JavaScript execution and confirmed that the dark theme was applied before any content was visible on screen. Language switching was tested by toggling between all three languages and verifying that every text element, placeholder, and button label updated correctly without a page reload. We also confirmed that language preference survived a full browser restart. The PDF warning was confirmed to appear when Hindi or Marathi was set, and to be skipped when English was active.

### Export and Data Integrity:

We verified export integrity by applying a specific set of filters, confirming the table showed a known set of records, and then exporting the CSV and cross-checking every row. All records matched exactly. We tested session expiry during export by manually clearing the session cookie after page load and then triggering the CSV download — the app redirected to the login page cleanly instead of downloading a PHP error response. PDF export was tested with an empty result set, a small set, and a large set of about three hundred records, all of which rendered correctly in the generated file.

**Table 2: Summary of Functional Test Results**

| Test Aspect                      | Result / Metric                  | What We Observed                                                                     |
|----------------------------------|----------------------------------|--------------------------------------------------------------------------------------|
| User Registration and Login      | <b>Fully functional</b>          | CSRF validation, rate limiting, and session regeneration all worked as expected      |
| CSRF Token Check                 | <b>403 on forged requests</b>    | The constant-time comparison blocked tampered tokens without leaking timing info     |
| Dual-Vector Rate Limiting        | <b>Locked out at 5 attempts</b>  | Email and IP counters worked independently — bypassing one did not bypass the other  |
| Budget Overspent Alert           | <b>Server-side trigger only</b>  | Alert fired correctly even when the client device clock was manually changed         |
| Recurring Expense Detection      | <b>Active from 3 entries</b>     | No warm-up needed; the SQL query returned results immediately on small datasets      |
| Spending Surge Detection         | <b>Month-on-month comparison</b> | Dual threshold ( $>₹500$ and $>25\%$ ) prevented false flags on low-spend categories |
| CSV Export after Session Expiry  | <b>Redirect to login</b>         | No JSON error file was downloaded — the Blob pipeline caught the 401 cleanly         |
| PDF Export with Hindi/Marathi on | <b>Warning modal appeared</b>    | Users were informed before export rather than receiving a garbled output file        |
| Theme Toggle                     | <b>No visible flash</b>          | Synchronous IIFE confirmed to run before the browser's first paint                   |

| Test Aspect                | Result / Metric               | What We Observed                                                              |
|----------------------------|-------------------------------|-------------------------------------------------------------------------------|
| Language Switch (EN/HI/MR) | <b>All 60+ keys updated</b>   | Switch was instant and preference persisted across pages and browser restarts |
| Cross-User Delete Attempt  | <b>Blocked by query check</b> | WHERE id=? AND user_id=? prevented deletion of another user's records         |
| XSS via Product Name Field | <b>Tags rendered as text</b>  | Script injected into product name displayed safely without executing          |

## VI.CONCLUSIONS

Looking back at the twenty papers we reviewed when starting this project, the most striking thing isn't what they got wrong — it's what they consistently skipped over. Security, multilingual support, dark mode, and intelligent analytics that work before you have months of data to train on are not exotic features. They are the kind of things a real application needs to function well for real users. The fact that most published implementations don't have them isn't a knowledge gap; it seems more like a priority gap.

TrackXpense was our attempt to build a version of this application that took all of those areas seriously at the same time. The security architecture — CSRF validation with timing-safe comparison, dual-vector rate limiting, session fixation prevention, and ownership enforcement in every delete operation — came directly from working through what the attack surface of a personal finance app actually looks like. The SQL-based analytics came from wanting to give users useful feedback from day one without the overhead of an ML pipeline. The Marathi language support came from thinking honestly about who would actually use an app like this in Maharashtra.

## REFERENCES

1. "Money Map: The Personal Finance Tracker," IEEE Xplore. Available: <https://ieeexplore.ieee.org/document/11035699>
2. "Personal Finance Management and Prediction using ML Algorithms," IEEE Xplore, 2024. DOI: 10.1109/10816949
3. "Personal Financial Management Behavior using Digital Platforms," Journal of Financial Management, Markets and Institutions, Vol. 10, No. 2, pp. 1–26, 2022.
4. "Mobile Bookkeeper: Personal Financial Management Application with Receipt Scanner," IEEE Xplore, 2021. DOI: 10.1109/OT4ME53559.2021.9638794
5. "Personal Financial Planner using Forward Chaining," IEEE Xplore, 2015. DOI: 10.1109/7010211