



TRIBHUVAN UNIVERSITY
FACULTY OF HUMANITIES AND SOCIAL SCIENCE

Project Proposal

on

“KharchaTrack : Smart Expenses Tracker with Insights”

Submitted to

Department of Computer Application
National College of Computer Studies

In partial fulfilment of the requirements
of

Bachelor's Degree in Computer Application

Submitted By:

Reman Buddhacharya
(NCCSBCA099 / 2081)

Under Supervision of
Yuba Raj Devkota



Tribhuvan University
Faculty of Humanities and Social Sciences
National College of Computer Studies

Supervisor's Recommendations

I hereby recommend that this project prepared under my supervision by **Reman Buddhacharya** entitled “**KharchaTrack : Smart Expenses Tracker with Insights**” in partial fulfillment of the requirements for a degree of Bachelors in Computer Application is recommended for the final evaluation.

SIGNATURE

Yuba Raj Devkota

SUPERVISOR

Faculty Member

Department of Computer Application

National College of Computer



Tribhuvan University
Faculty of Humanities and Social Sciences
National College of Computer Studies

Letter of Approval

This is to certify that this project prepared by **Reman Buddhacharya** entitled “**KharchaTrack : Smart Expenses Tracker with Insights**” in partial fulfillment of the requirements for the degree of Bachelors in Computer Application has been evaluated. In our opinion, it is satisfactory in scope and quality as a project for the required degree.

<p style="text-align: center;">Signature of Supervisor</p> <p style="text-align: center;">Yuba Raj Devkota Faculty Member Department of Computer Application National College of Computer Studies Paknajol, Kathmandu</p>	<p style="text-align: center;">Signature of HOD / Coordinator</p> <p style="text-align: center;">Rajan Poudel Faculty Member Department of Computer Application National College of Computer Studies Paknajol, Kathmandu</p>
<p style="text-align: center;">Signature of Internal Examiner</p>	<p style="text-align: center;">Signature of External Examiner</p>

Abstract

KharchaTrack is a smart, ML-powered expense tracking web application designed to revolutionize personal financial management. Unlike traditional expense trackers, it leverages machine learning algorithms such as Moving Average for forecasting and Isolation Forest for anomaly detection to offer meaningful insights into users' financial habits. Built using Laravel, Livewire, and PHP-ML, KharchaTrack enables real-time monitoring, predictive analytics, and anomaly alerts, all through a user-friendly interface. This system aims to enhance user experience in expense planning, fraud detection, and overall budget management.

Keywords: *KharchaTrack, expense tracking, Laravel, machine learning, Moving Average, Isolation Forest, financial forecasting, anomaly detection.*

Acknowledgement

Completing this study in its present form has been an enriching and rewarding experience. I would like to express my sincere gratitude to all the people who have supported me throughout this project.

First and foremost, I extend my heartfelt thanks to the BCA department for their guidance, supervision, and invaluable support in completing this project. Their assistance and advice have been instrumental in shaping this project.

I would also like to thank all the individuals who contributed directly or indirectly to the success of this project. Their insights, feedback, and encouragement were invaluable.

Finally, I would like to express my gratitude to the readers of this report, who will hopefully benefit from this work. I hope that this report will provide useful insights and inspire further research in this field.

Table of Content

Supervisor's Recommendations	i
Letter of Approval	ii
Abstract	iii
Acknowledgement	iv
Table of Content	v
List of Abbreviation	vii
List of Figures	viii
Chapter 1: Introduction	1
1.1. Introduction	1
1.2. Problem Statement	2
1.3. Objective	2
1.4. Scope and Limitation	3
1.5. Development Methodology	3
1.6. Report Organization	5
1.6.1. Introduction	5
1.6.2. Background Study and Literature Review	5
1.6.3. System Analysis and Design	5
1.6.4. Implementation and Testing	5
1.6.5. Conclusion and Future Recommendations	5
Chapter 2: Background Study and Literature Review	6
2.1. Background Study	6
2.2. Study of Existing Systems	6
2.3. Literature Review	7
Chapter 3: System Analysis and Design	9
3.1. System Analysis	9
3.1.1. Requirement Analysis	9

3.1.2. Feasibility Analysis	10
3.1.2.1. Technical Feasibility	10
3.1.2.2. Operational Feasibility	11
3.1.2.3. Economic Feasibility	11
3.1.3. Object Modelling: Object and Class Diagram	12
3.1.4. Dynamic Modelling : State and Sequence Diagram	16
3.1.5 Process Modelling: Activity Diagram	19
3.2. System Design	21
3.2.1. Refinement of Classes and Objects	21
3.2.2. Component Diagram	25
3.2.3. Deployment diagram	26
3.3. Algorithm Details	27
Chapter 4: Implementation and Testing	29
4.1. Implementation	29
4.1.1. Tools used	29
4.1.2. Implementation details of modules	30
4.2. Testing	36
Chapter 5: Conclusion and Future Recommendations	42
5.1. Lesson Learnt / Outcome	42
5.2. Conclusion	42
5.3. Future Recommendations	42
References	44
Appendix	i

List of Abbreviation

APIs	Application Programming Interface
CSS	Cascading Style Sheets
CSV	Comma-separated values
HTML	HyperText Markup Language
MA	Moving Average
ML	Machine Learning
PDF	Portable Document Format
PHP	Personal Home Page
PWA	Progressive Web App
SMS	Short Message Service
UI	User Interface
YNAB	You Need a Budget

List of Figures

Figure 1.1: Waterfall Model	3
Figure 3.1: Use Case Diagram with Online User	9
Figure 3.2: Object Diagram	12
Figure 3.3: Class Diagram	14
Figure 3.4: State Diagram	16
Figure 3.5: Sequence Diagram	Error! Bookmark not defined.
Figure 3.6: Activity Diagram	Error! Bookmark not defined.
Figure 3.7: Refinement of Classes	Error! Bookmark not defined.
Figure 3.8: Refinement of Object	23
Figure 3.9: Component Diagram	25
Figure 3.10: Deployment Diagram	26

Chapter 1: Introduction

1.1. Introduction

Effective financial management is essential for both individuals and businesses. However, tracking expenses manually or using basic expense trackers often lacks deeper insights, making it difficult to forecast future spending or detect fraudulent transactions. This highlights the need for a smarter, data-driven solution.

KharchaTrack is a smart expense tracker designed to address these challenges. It will provide automated expense tracking, forecasting, and fraud detection using machine learning. Unlike conventional expense trackers, KharchaTrack will analyze spending patterns to predict future expenses and identify unusual transactions that may indicate fraud.

This project will be developed using Laravel, Livewire, and PHP-ML, ensuring a scalable, interactive, and efficient financial management system. The combination of these technologies will allow for real-time data processing, intuitive user interactions, and advanced predictive analytics.

This proposal outlines the development of KharchaTrack, including its objectives, methodology, expected outcomes, and implementation plan. The project aims to create a comprehensive expense management tool that not only records transactions but also helps users make informed financial decisions with ML-powered insights.

1.2. Problem Statement

In today's fast-paced world, managing personal finances has become increasingly challenging. Many individuals struggle with tracking their expenses, identifying spending patterns, and avoiding unnecessary financial strain. Traditional methods, such as manual logs or basic spreadsheets, lack real-time insights and predictive capabilities. Moreover, fraudulent transactions and unusual spending behaviors often go unnoticed, leading to financial losses. Existing expense trackers mostly focus on recording transactions without providing intelligent forecasting or anomaly detection to help users make better financial decisions.

This project aims to develop a Smart Expense Tracker using Laravel, Livewire, and PHP-ML that integrates Moving Average for expense forecasting and Isolation Forest for anomaly detection. This system will not only allow users to log and categorize expenses but also provide future expense predictions and alert users to potentially suspicious transactions.

By leveraging machine learning techniques, this solution will empower users with data-driven financial insights, helping them make informed decisions, detect anomalies, and improve their spending habits.

1.3. Objective

The main objective of KharchaTrack is:

- To help users manage expenses efficiently through data-driven insights, expense prediction (Moving Average), and anomaly detection (Isolation Forest).

1.4. Scope and Limitation

The primary scope of KharchaTrack is to develop a machine learning-powered web application that predicts users' future expenses and flags unusual or potentially fraudulent transactions. The system supports user account management, automatic and manual categorization of expenses, export of data, and real-time dashboard insights. It is designed for general users looking to manage personal finances more efficiently.

Limitations include the need for an internet connection for real-time functionality, reliance on historical data for accurate forecasting, and limited support for multi-user organization-level deployment. Additionally, while the application offers self-hosting capability, it assumes users have the technical expertise to deploy it manually if desired.

1.5. Development Methodology

This project is on a small scale and has well-defined requirements and a linear approach. Under such development circumstances, the simplest development model, the waterfall model, is applicable. The waterfall model produces a set of documents after each stage, as well as a time frame that is enough to implement the project under the methodology.

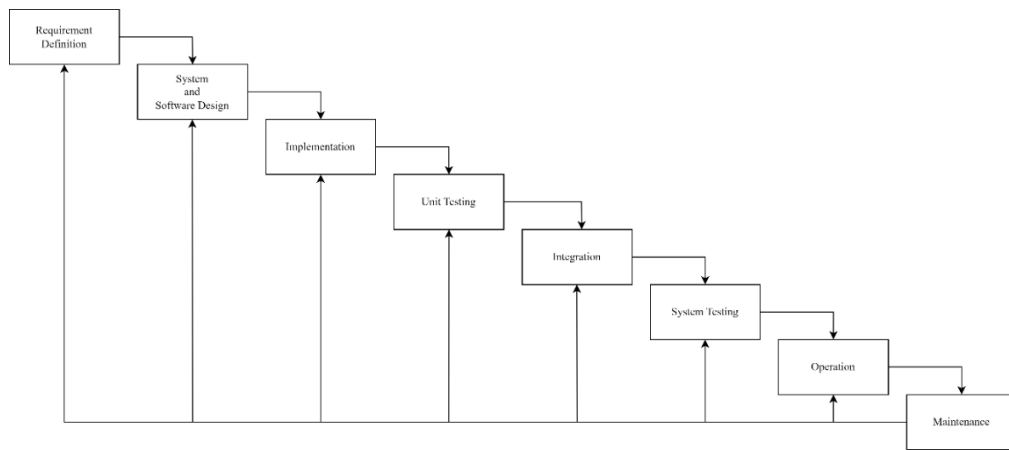


Figure 1.1: Waterfall Model

- i. **Requirement Definition:** The requirements for KharchaTrack were defined in this stage, including the features and functionality that the application should have. This stage involved researching, analyzing, and identifying the key requirements for the application.

- ii. **System and Software Design:** Based on the requirements defined in the previous stage, a detailed plan for the application was created in the system and software design stage. This included defining the application architecture, data structures, and user interface design.
- iii. **Implementation:** The implementation stage was carried out, where the application was coded based on the system and software design created in the previous stage. This stage involved writing the code and testing the application as it was developed.
- iv. **Unit Testing:** Individual components of the application were tested in this stage to ensure that they worked as expected. This stage involved creating test cases and running tests to verify the functionality of the code.
- v. **Integration:** The integration stage involved combining the individual components of the application into a single working application. This stage also included testing the integrated application to ensure that all components worked together seamlessly.
- vi. **System Testing:** The entire application was tested in the system testing stage to ensure that it met the requirements defined in the first stage. This stage involved testing the application in a real-world environment and identifying any issues that may have arisen.
- vii. **Operation:** The operation stage involved deploying the application to a live environment and making it available to users. This stage included monitoring the application to ensure that it functioned properly and addressing any issues that arose.
- viii. **Maintenance:** The maintenance stage will involve making updates and improvements to the application over time. This stage will include fixing bugs, adding new features, and ensuring that the application remains up to date with the latest technologies and standards.

1.6. Report Organization

1.6.1. Introduction

This chapter provides an overview of the project, including the problem it aims to solve, the project objectives, scope and limitations, and development methodology of the project.

1.6.2. Background Study and Literature Review

This chapter focuses on study of existing systems and review of different articles studied.

It also explores the underlying concepts and principles that informed the development of the system.

1.6.3. System Analysis and Design

This chapter covers the project's requirements analysis, feasibility assessment, and design. System Models as Object & Class Diagram, State & Sequence Diagram, and Activity Diagram and System Designs as Refinement of Class and Object, Component Diagram and Deployment Diagram are all included.

1.6.4. Implementation and Testing

This chapter provides details on the project's implementation, the software and tools used, and the types of testing conducted.

1.6.5. Conclusion and Future Recommendations

This chapter discusses the project's potential outcomes and summarizes the main findings. It also provides recommendations for future work.

Chapter 2: Background Study and Literature Review

2.1. Background Study

With the rise in digital transactions and the complexity of financial management, individuals face challenges in tracking and understanding their expenses. Existing solutions like Mint, YNAB, and Expensify primarily offer basic expense tracking, budgeting, and visualization features. However, they fall short in offering predictive insights or fraud detection.

KharchaTrack was conceived as an answer to these limitations by integrating machine learning into expense management. It builds on established techniques such as Moving Average for financial forecasting and Isolation Forest for anomaly detection. These methods have proven effective in various domains for predictive modeling and outlier detection, and their application in this context offers a significant enhancement over conventional trackers.

2.2. Study of Existing Systems

Several existing expense tracking applications dominate the market, offering a variety of budgeting and reporting tools. However, they each have their limitations:

- **Mint:** Mint is a widely used financial planning tool that provides budgeting, bill tracking, and account aggregation. While it offers auto-categorization of expenses and basic insights, it lacks intelligent forecasting and anomaly detection features essential for modern financial analysis [1].
- **YNAB (You Need A Budget):** This application promotes a strict budgeting methodology and helps users plan every dollar they earn. Although effective for discipline-driven budgeting, YNAB requires manual inputs and does not utilize predictive analytics or anomaly detection through machine learning [2].
- **Expensify:** Geared primarily toward business expense reporting, Expensify includes receipt scanning and reimbursement management. However, it lacks personal budgeting features and fails to provide predictive insights based on historical patterns [3].

These tools provide helpful functionality, but their scope is limited to static data representation and manual financial tracking. KharchaTrack introduces intelligent features by integrating machine learning algorithms such as Moving Average for expense forecasting and Isolation Forest for identifying abnormal expenses. This ML-

driven approach enables users to gain real-time insights, recognize spending trends, and proactively respond to financial anomalies.

2.3. Literature Review

Expense tracking systems have significantly evolved over the past decades, transitioning from manual record-keeping to intelligent, automated platforms. Traditional methods like pen-and-paper logs and spreadsheets were often prone to human error and lacked analytical capabilities [4]. The rise of early digital solutions such as Quicken enabled users to categorize expenses and visualize budgets but still required manual effort [5].

Modern expense tracking applications like Mint, YNAB, and Expensify have integrated automation and cloud synchronization to streamline financial management. Mint automatically links bank accounts, categorizes transactions, and provides budget alerts [1]. YNAB (You Need a Budget) emphasizes proactive, rule-based budgeting where users assign every dollar a “job” before spending, encouraging disciplined financial habits [2]. Expensify, targeted more toward business users, offers smart receipt scanning using OCR and real-time anomaly detection for policy compliance [3].

While these tools have improved convenience, they are largely reactive. Recent academic and industry research has explored the integration of machine learning (ML) to make expense tracking predictive and insightful. Moving Average (MA) algorithms are widely used in financial applications to forecast future spending trends by smoothing out short-term fluctuations [6]. In the context of personal finance, this technique can help users plan budgets based on past behavior.

For anomaly detection, Isolation Forest has gained attention for its effectiveness in identifying unusual expense patterns. This unsupervised learning algorithm isolates data points by randomly selecting features and splitting values, making outliers easier to detect as they require fewer partitions [7]. The algorithm has been used in both academic projects and commercial systems for fraud detection and behavioral analysis.

Recent studies support the effectiveness of integrating ML into personal finance applications. Khan et al. [8] developed a system combining ML-based forecasting and anomaly detection, demonstrating improved budgeting accuracy and fraud detection. Similarly, Aishwarya and Hemalatha [9] reviewed multiple algorithms such as decision trees, linear regression, and neural networks for smart expense tracking and concluded that ML techniques significantly enhance usability and insight.

In summary, existing systems provide helpful tools for budget management, but their predictive and anomaly detection capabilities are limited. The integration of algorithms like Moving Average and Isolation Forest into applications such as KharchaTrack presents a valuable innovation. These ML-powered features can enable users not only to track and categorize their expenses but also to receive forward-looking insights and alerts on unusual activity, thereby making personal finance management more intelligent and secure.

Chapter 3: System Analysis and Design

3.1. System Analysis

3.1.1. Requirement Analysis

Requirements were collected through personal evaluation of different existing systems, along with suggestions from mentors, classmates, and supervisors.

i. Functional Requirements



Figure 3.1: Use Case Diagram with Online User

The use case diagram in previous page specifies the basic operations a user can perform in the proposed system.

- Users can create an account and log in with unique credentials.
- Password recovery is available if users forget their credentials.
- Users can add, view, update, and delete expense entries, with timestamps for each transaction.
- Expenses can be categorized automatically or manually.
- The system predicts future expenses using the Moving Average algorithm.
- Isolation Forest detects and flags unusual or potentially fraudulent transactions.

- Users can view and revert to previous versions of their expense entries.
- A personalized dashboard displays expense summaries, forecasts, and anomalies.
- Users are notified about forecasted expenses and flagged transactions.
- Users can export expense data to CSV or PDF formats.
- The system is compatible with modern web browsers.

ii. Non-Functional Requirements

- The application should provide real-time updates for expense tracking and forecasting with minimal latency.
- The system should be able to handle a growing number of users and increasing expense data without performance issues.
- The platform should have an intuitive and user-friendly interface to make navigation and expense management easy.
- The dashboard should display clear insights and be simple to use.
- The system should be available 24/7 with minimal downtime for maintenance or updates.
- Regular backups of user data should be taken to ensure data integrity.
- The system should be compatible with all modern web browsers (e.g., Chrome, Firefox, Edge).
- It should be responsive and function well on both desktop and mobile devices.
- The application should have clean, modular code that is easy to update and maintain over time.

3.1.2. Feasibility Analysis

The system is evaluated for future development with a set of constraints. The feasibility study is done regarding the available technologies, time constraints, area of application, cost of deployment and upkeep, and future possibilities of the project.

3.1.2.1. Technical Feasibility

KharchaTrack will be developed using stable, open-source technologies. The frontend will use HTML, CSS, and JavaScript, with Livewire for real-time updates. The backend will be powered by Laravel, ensuring a secure and efficient architecture. PostgreSQL will store user data securely. PHP-ML will handle machine learning algorithms like

Moving Average and Isolation Forest for forecasting and anomaly detection. The system will be web-based and compatible with modern browsers and mobile devices, ensuring accessibility from anywhere. Development will be done using free software like Visual Studio Code and GitHub.

3.1.2.2. Operational Feasibility

The user interface of KharchaTrack will be designed to be intuitive and user-friendly, ensuring ease of use for individuals of all technical backgrounds. Similar to existing expense tracking applications, the interface will feature simple navigation and interactive elements for a smooth user experience. The platform will be responsive, ensuring compatibility across various devices, including desktops, tablets, and mobile phones. By following basic design principles, the application will be easy to understand, enabling users to quickly adopt and effectively manage their expenses.

3.1.2.3. Economic Feasibility

Most of the software used for developing this project will be open source and free. Along with it, suitable cloud hosting will be used as 000webhostapp or infinityfree. All of the technology used will be free of cost for developing this project.

3.1.3. Object Modelling: Object and Class Diagram

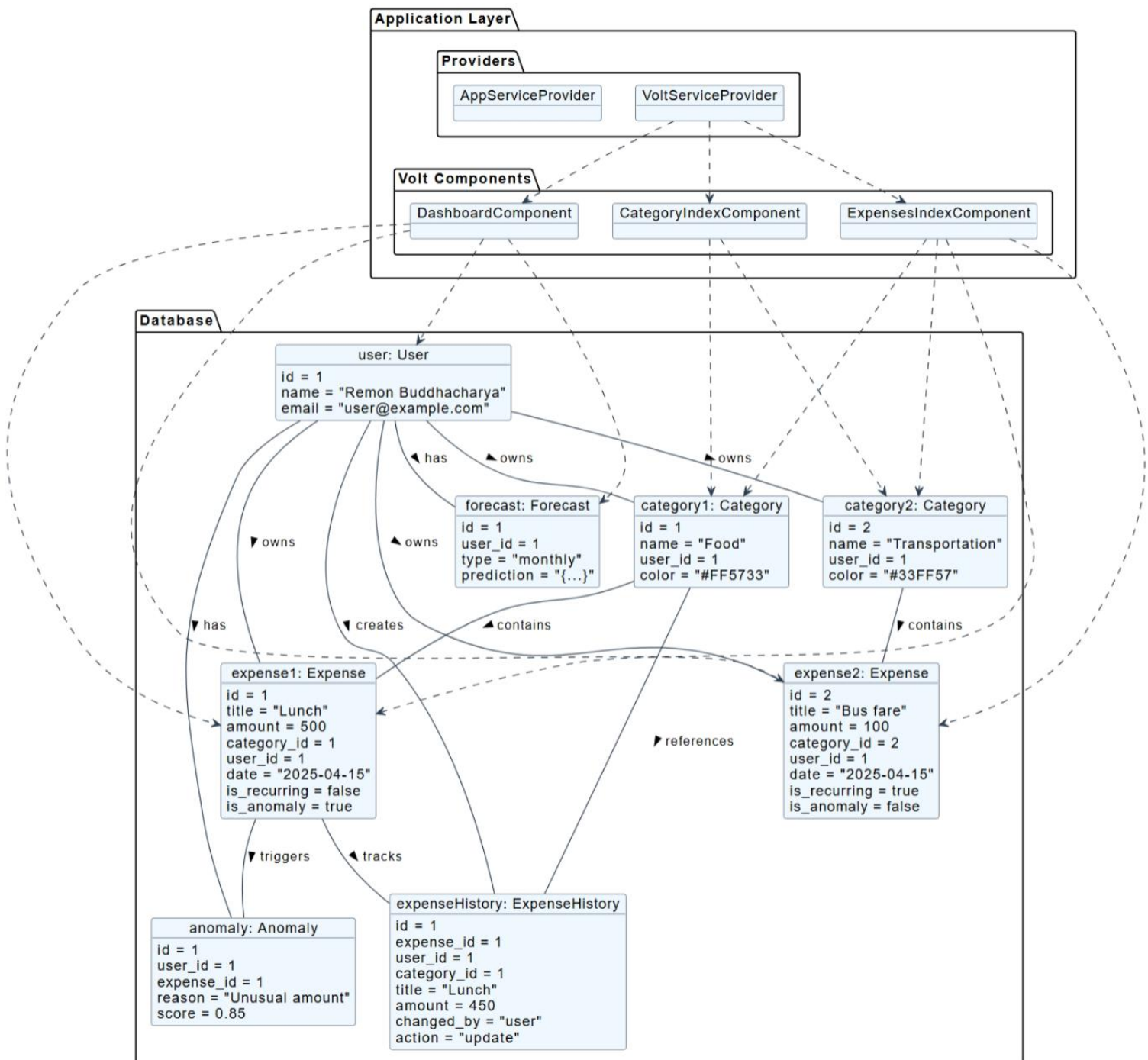


Figure 2.2: Object Diagram

The object diagram provides a concrete example of how actual data instances relate to each other in the system:

- It shows a user named "Remon Buddhacharya" with sample data including two categories ("Food" and "Transportation") and two expenses ("Lunch" for RS500 and "Bus fare" for RS100).
- The ExpenseHistory object shows a previous state of the "Lunch" expense when it cost RS450, demonstrating how expense modifications are tracked.
- Sample Forecast and Anomaly objects are shown, with the Anomaly flagging the "Lunch" expense for an "Unusual amount" with a confidence score of 0.85.

This diagram helps visualize how real data flows through the application, making the relationships more concrete and understandable.

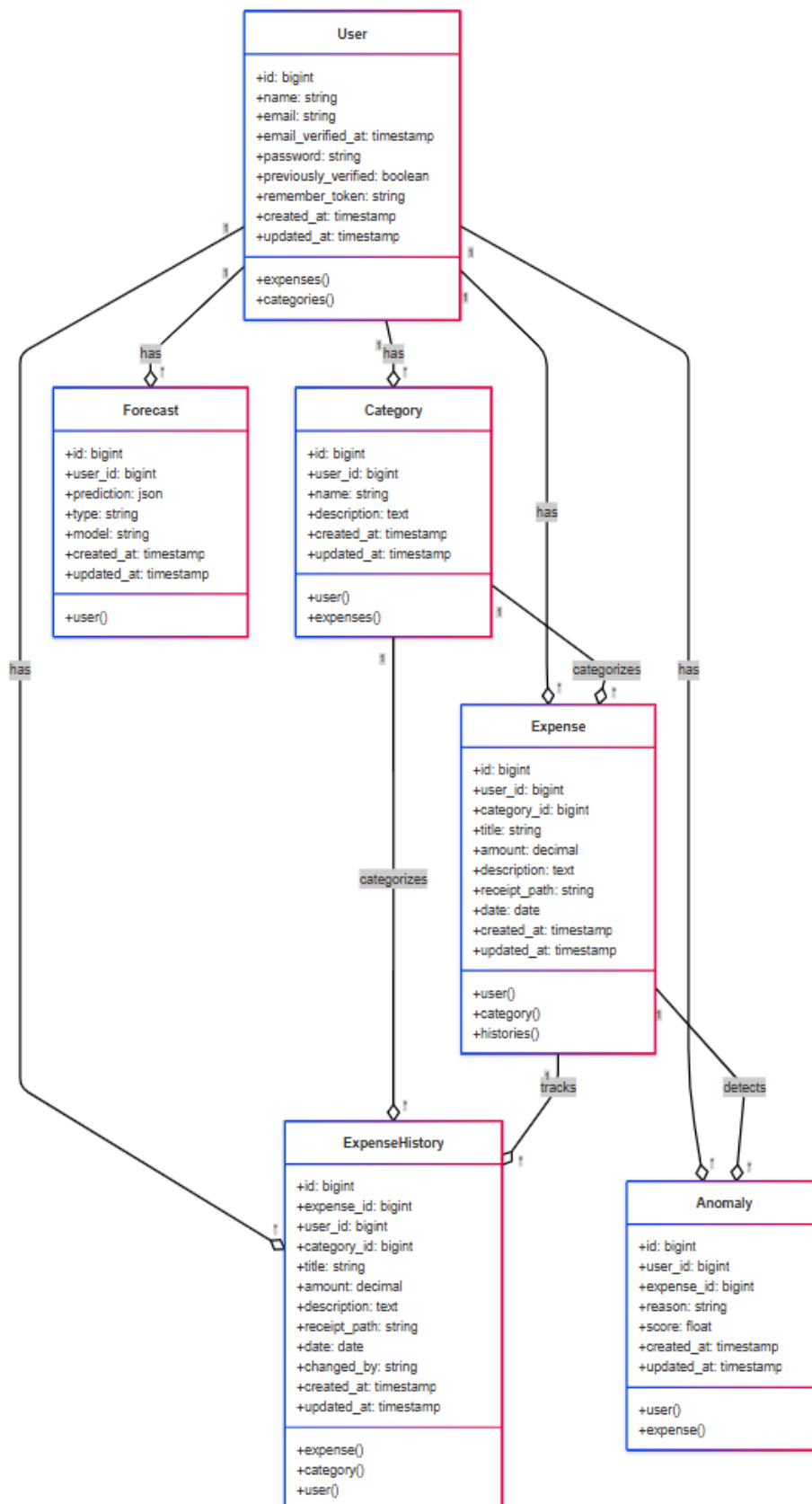


Figure 3.3: Class Diagram

This diagram defines the model and data structure of your expense tracking system.

Component Interactions:

- **User-Expense Relationship:** One user can have many expenses (1 to *), establishing ownership of financial data.
- **User-Category Relationship:** One user (admin) can create many categories (1 to *), allowing personalized expense organization.
- **Category-Expense Relationship:** One category can contain many expenses (1 to *), enabling classification of spending.
- **Expense-ExpenseHistory Relationship:** One expense tracks many history records (1 to *), implementing an audit trail feature.
- **Expense-Anomaly Relationship:** Expenses can be flagged by the anomaly detection system (1 to *), marking unusual spending patterns.

This diagram forms the foundation of the data model, showing how entities relate to each other and defining the core business objects. The diamond notation (--o) indicates composition relationships, where parent entities "own" child entities.

3.1.4. Dynamic Modelling : State and Sequence Diagram

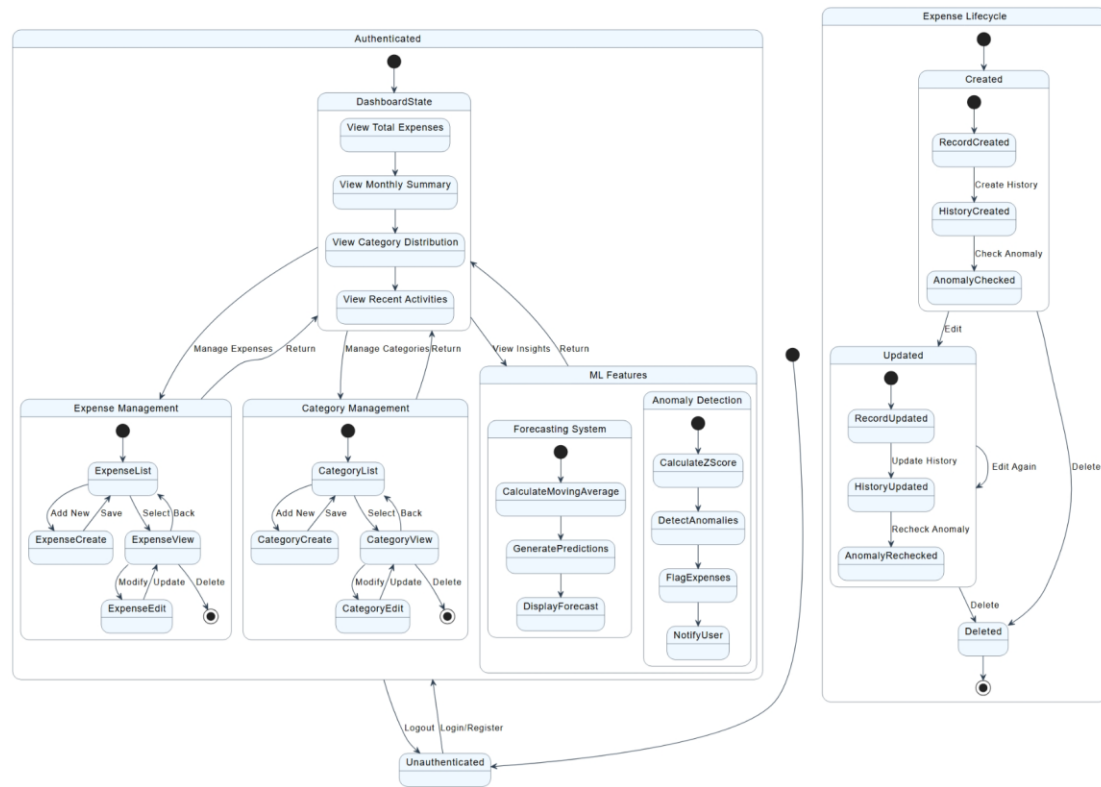


Figure 3.4: State Diagram

The state diagram depicts both user authentication states and expense lifecycle states:

- **Authenticated User States:** Within the Authenticated composite state, it shows navigation between Dashboard, ExpenseManagement, CategoryManagement, and other features.
- **Expense Lifecycle States:** The Expense state machine shows how an expense transitions from Created → Updated → Deleted, with History records created for each state change.

This diagram helps understand both how users navigate through the system and how data entities change throughout their lifecycle.

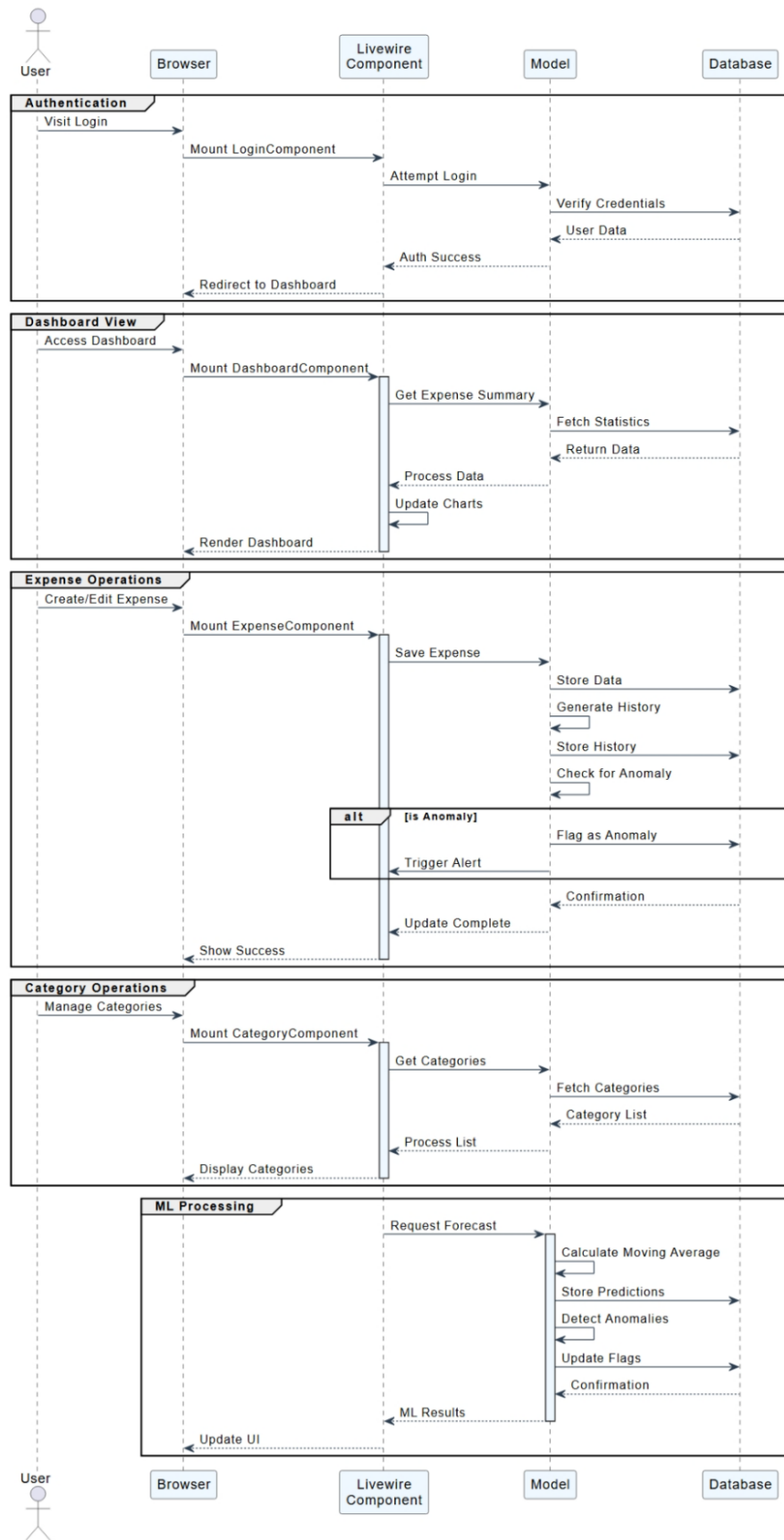


Figure 3.5: Sequence Diagram

The sequence diagram for KharchaTrack illustrates the temporal interactions and message flow across five key modules:

1. **Authentication:** Shows the interaction from the user's login request to credential verification by the model and database. Upon successful authentication, the user is redirected to the dashboard.
2. **Dashboard View:** Displays how the dashboard component retrieves and processes expense summary data by interacting with backend models and the database to render analytical charts.
3. **Expense Operations:** Details the process for creating or editing an expense. It includes saving data, generating and storing history, checking for anomalies, and, if detected, flagging them and triggering an alert.
4. **Category Operations:** Explains how category data is fetched and processed for display. The system retrieves a list of categories from the database and updates the user interface accordingly.
5. **ML Processing:** Demonstrates how the application handles machine learning-based forecasting. It shows the sequence of calculating moving averages, storing predictions, detecting anomalies, updating anomaly flags, and displaying results back to the user interface.

This diagram provides a comprehensive overview of component-level communication and timing in the KharchaTrack system, emphasizing its modular, event-driven architecture and intelligent financial processing.

3.1.5 Process Modelling: Activity Diagram

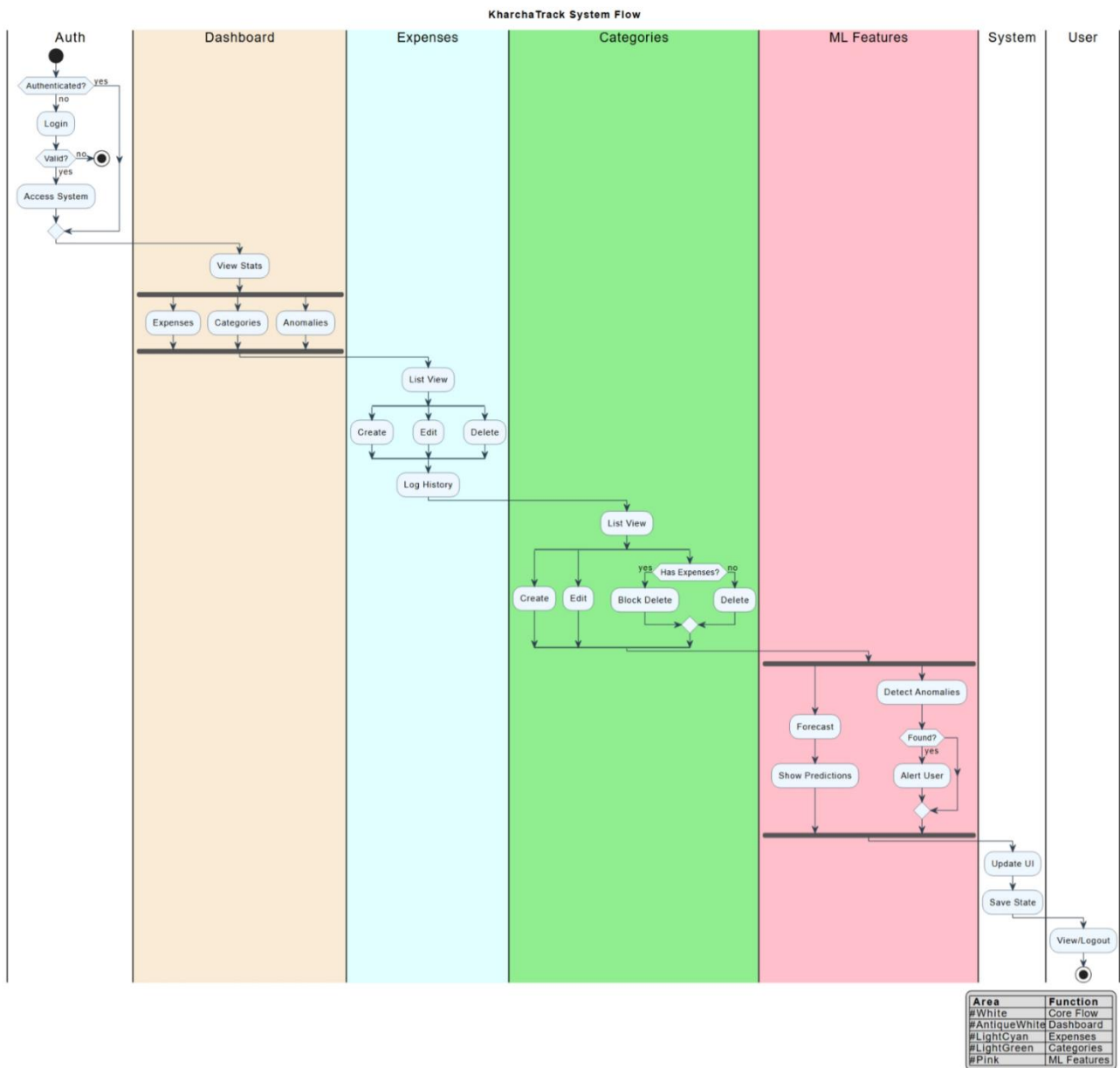


Figure 3.6: Activity Diagram

The activity diagram displays the system flow of the project, organized into color-coded vertical swim lanes:

- **Auth (White):** Begins with authentication flow including "Authenticated?" decision point, Login action, and "Valid?" verification before granting system access.
- **Dashboard (Light Beige):** After authentication, users can "View Stats" and navigate to three parallel options: Expenses, Categories, or Anomalies.

- **Expenses (Light Cyan):** Shows flow for expense management with "List View" followed by three options (Create, Edit, Delete), all concluding with "Log History."
- **Categories (Light Green):** Similar to expenses section with "List View" but includes a decision point "Has Expenses?" that determines whether categories can be deleted. If yes, there's a "Block Delete" action; if no, normal "Delete" is permitted. "Create" and "Edit" actions are also available.
- **ML Features (Pink):** Shows machine learning functionality including "Detect Anomalies," "Forecast," a decision point "Found?" with "Show Predictions" and "Alert User" actions.
- **System (White):** Contains system functions like "Update UI" and "Save State."
- **User (White):** Shows a "View/Logout" option concluding the flow.

The diagram uses consistent visual symbols with rounded rectangles for actions, diamond shapes for decision points, and arrows showing flow direction. A legend at the bottom maps each colored area to its function.

3.2. System Design

3.2.1. Refinement of Classes and Objects

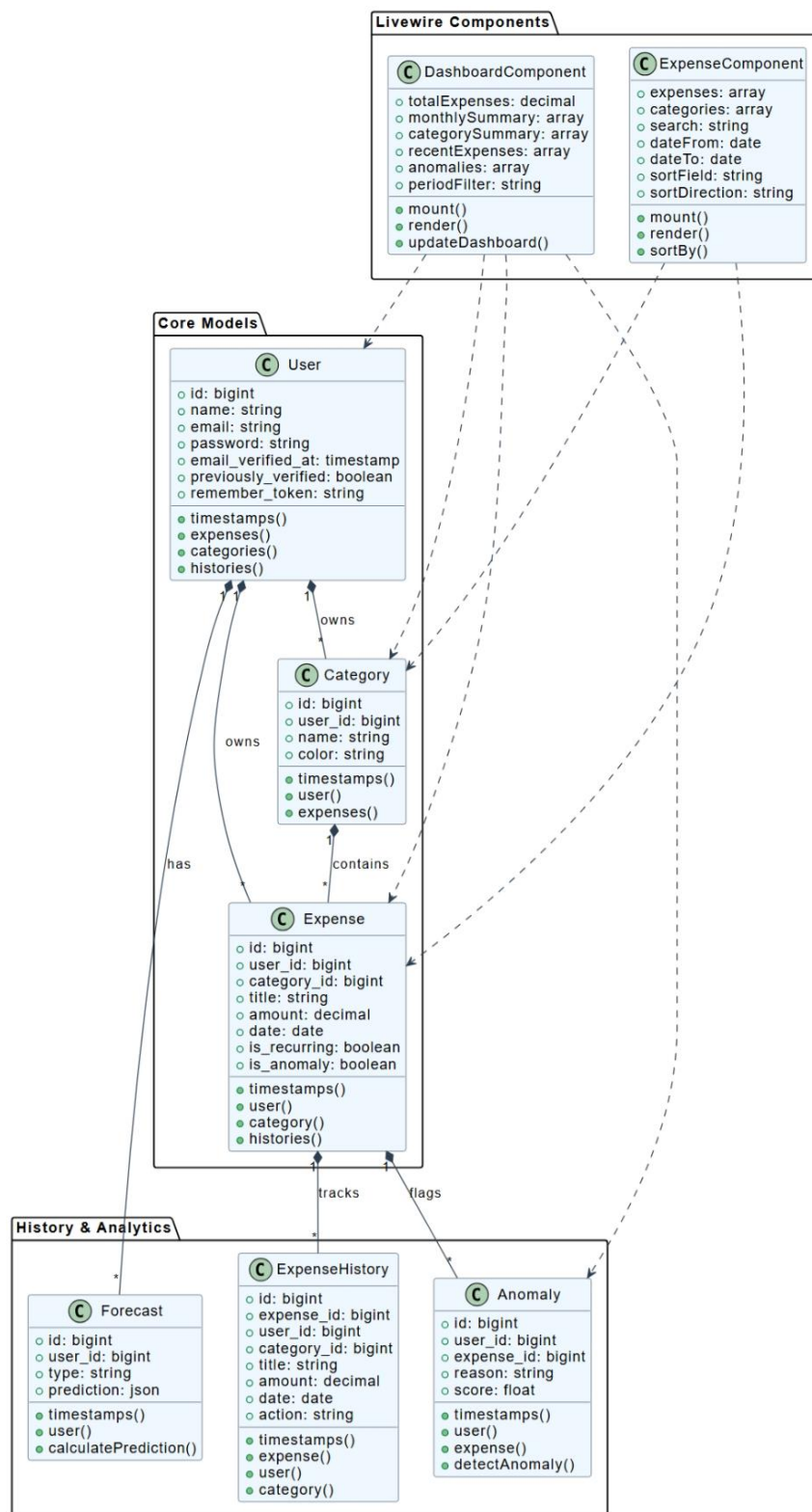


Figure 3.7: Refinement of Classes

The diagram shows a class diagram for a financial tracking application with the following structure:

- **Livewire Components:** Located at the top, showing UI components:
 - **DashboardComponent:** Contains properties for financial data display (totalExpenses, monthlySummary, categorySummary, recentExpenses, anomalies, periodFilter) and methods (mount(), render(), updateDashboard()).
 - **ExpenseComponent:** Includes properties for handling expense data (expenses, categories, search, dateFrom, dateTo, sortField, sortDirection) and methods (mount(), render(), sortBy()).
- **Core Models:** Central section containing foundational data structures:
 - **User:** Stores user information (id, name, email, password, email_verified_at, previously_verified, remember_token) with methods (timestamps(), expenses(), categories(), histories()).
 - **Category:** Manages expense categories (id, user_id, name, color) with methods (timestamps(), user(), expenses()).
 - **Expense:** Records financial transactions (id, user_id, category_id, title, amount, date, is_recurring, is_anomaly) with methods (timestamps(), user(), category(), histories()).
- **History & Analytics:** Bottom section for tracking changes and predictions:
 - **ExpenseHistory:** Logs expense modifications (id, expense_id, user_id, category_id, title, amount, date, action) with methods (timestamps(), expense(), user(), category()).
 - **Forecast:** Generates financial predictions (id, user_id, type, prediction) with methods (timestamps(), user(), calculatePrediction()).
 - **Anomaly:** Flags unusual transactions (id, user_id, expense_id, reason, score) with methods (timestamps(), user(), expense(), detectAnomaly()).

The diagram shows relationships between classes with solid and dashed lines indicating ownership, composition, and dependencies.

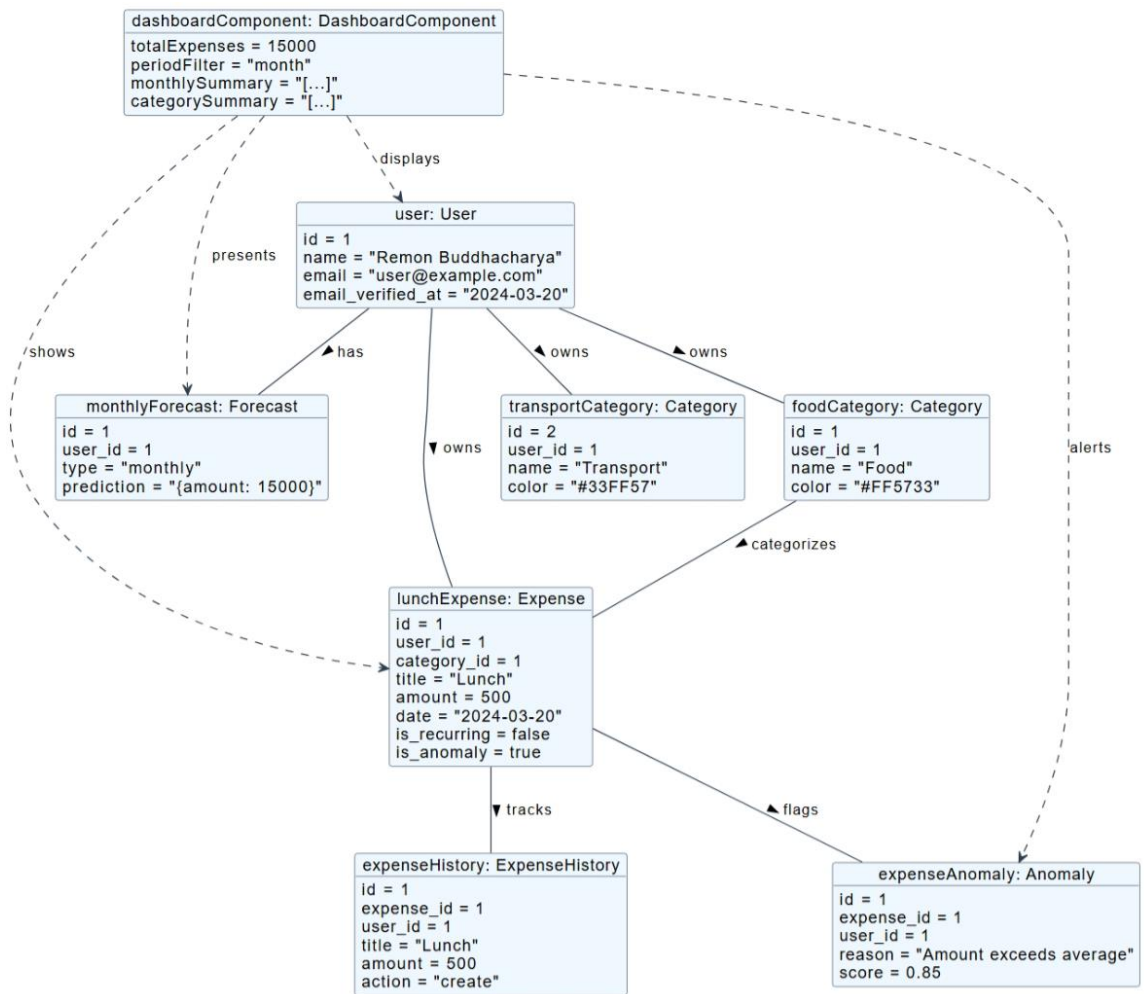


Figure 3.8: Refinement of Object

The diagram shows an object diagram for a financial tracking application that illustrates specific instances and their relationships:

- **DashboardComponent**: Contains actual data values (totalExpenses = 15000, periodFilter = "month") that displays financial information.
- **User**: Shows a specific user instance (id = 1, name = "Remon Buddhacharya", email = "user@example.com") who owns categories and has forecast data.
- **Category**: Two specific categories are shown:
 - **transportCategory**: A transport expense category (id = 2, name = "Transport", color = "#33FF57")
 - **foodCategory**: A food expense category (id = 1, name = "Food", color = "#FF5733")
- **Expense**: Contains a specific expense instance (lunchExpense) with concrete values (id = 1, title = "Lunch", amount = 500, date = "2024-03-20", is_recurring = false, is_anomaly = true).

- **ExpenseHistory:** Tracks a specific history record (id = 1) for the lunch expense with action = "create".
- **Anomaly:** Shows an anomaly instance (expenseAnomaly) flagging the lunch expense with reason = "Amount exceeds average" and score = 0.85.

The diagram uses solid and dashed lines with directional labels ("owns", "categorizes", "tracks", "flags", etc.) to show how these specific objects relate to each other during runtime.

3.2.2. Component Diagram

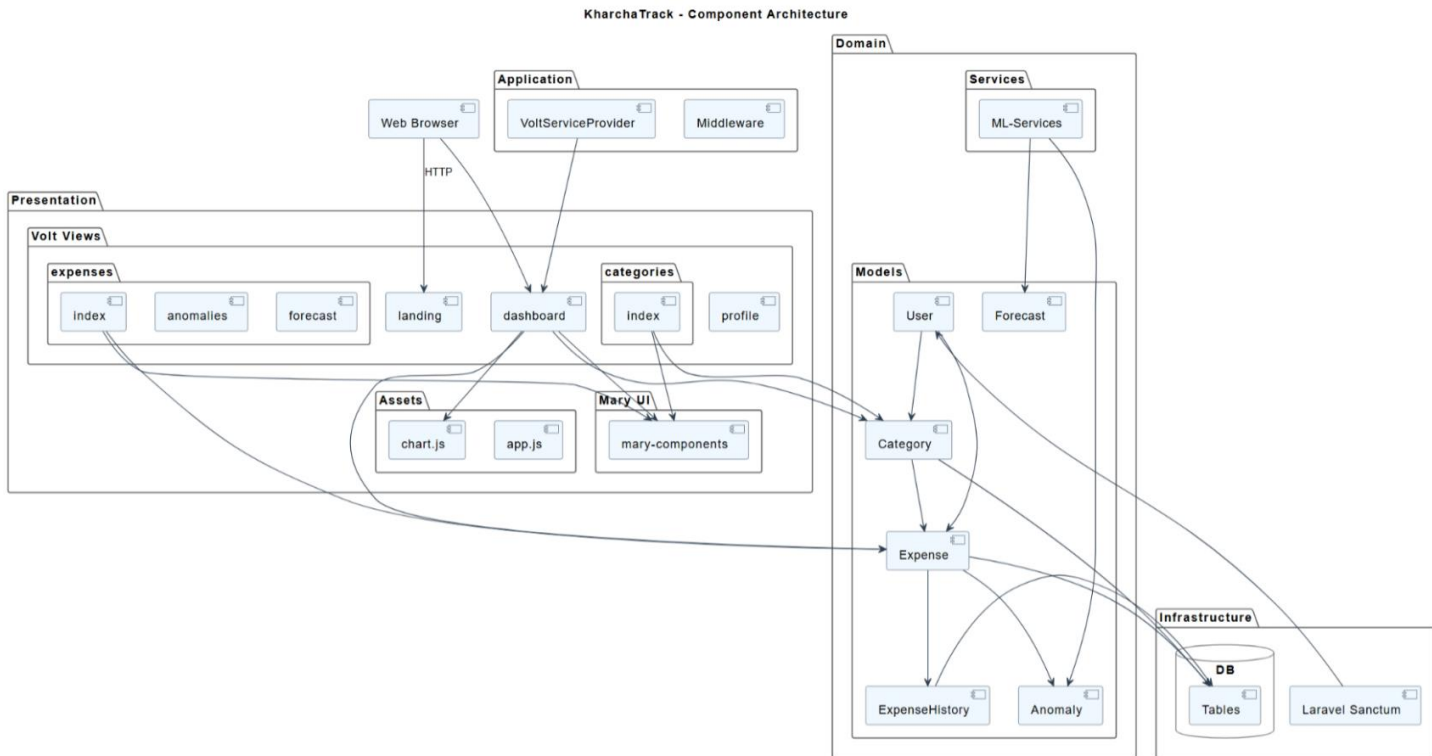


Figure 3.9: Component Diagram

The component diagram describes the system architecture across four layers:

- **Presentation Layer:** Shows how Blade Templates, Layouts, Livewire/Volt Components (Dashboard, Expenses, Categories), Mary UI, and Frontend Assets interact to create the user interface. Dependencies show that Livewire renders through Blade, which uses MaryUI for styling.
- **Application Layer:** Illustrates how Routes direct requests to Controllers or mount Livewire components, with Middleware filtering requests and Service Providers bootstrapping components.
- **Domain Layer:** Shows the core data models (User, Expense, Category, ExpenseHistory) and their relationship with Events for tracking changes.
- **Infrastructure Layer:** Depicts the foundational services including Database (with Migrations, Seeders, Factories), Authentication (Sanctum), Authorization (Spatie Permissions), and File Storage.

The diagram shows clear separation of concerns with well-defined dependencies between components. For example, Models dispatch Events which create ExpenseHistory records, and Controllers use Models to interact with the Database.

3.2.3. Deployment diagram

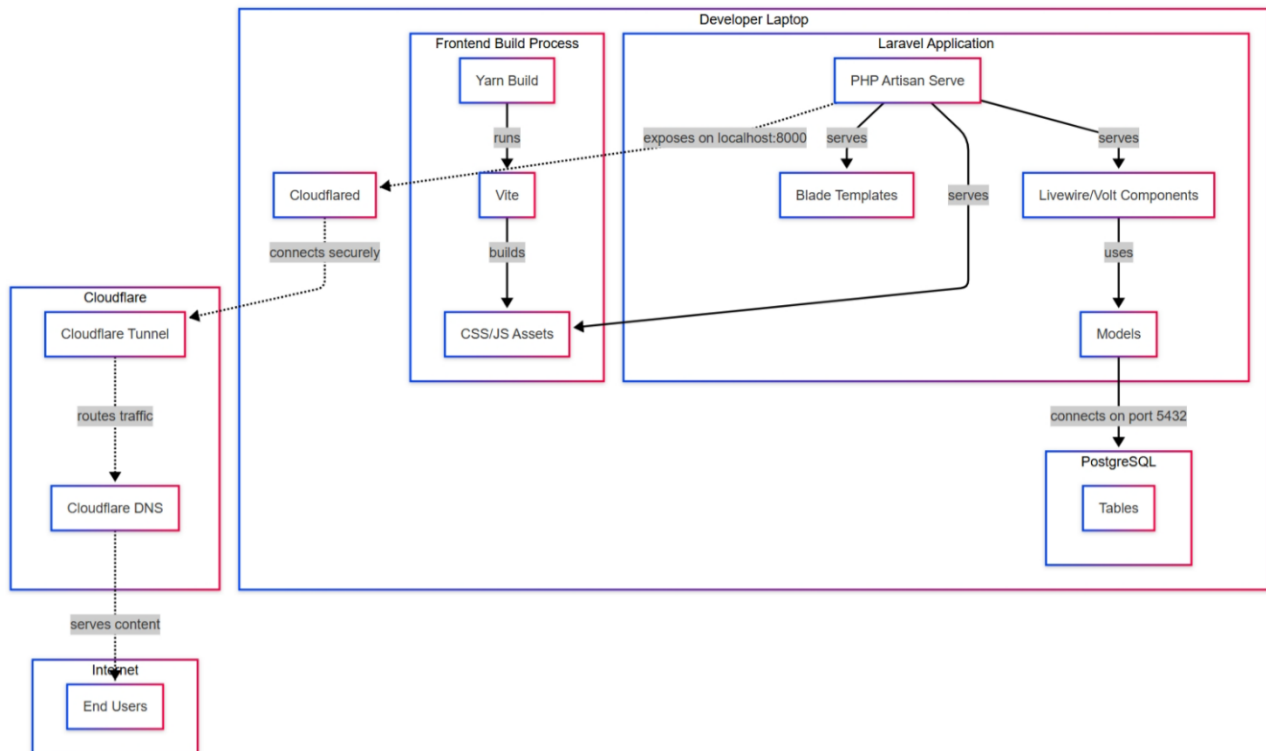


Figure 3.10: Deployment Diagram

The deployment diagram shows the development environment setup:

- **Developer Laptop:** Contains the Laravel Application running via PHP Artisan Serve on port 8000, Frontend Build Process using Yarn Dev and Vite, PostgreSQL database on port 5432, and Cloudflared for tunneling.
- **Cloudflare:** Shows how Cloudflare Tunnel and DNS services route internet traffic securely to the local development environment.
- **Internet:** Represents end users accessing the application through their browsers.

Connection arrows show how components interact: PHP Artisan serves Blade templates and Livewire components, Models connect to PostgreSQL, Yarn Dev builds frontend assets, and Cloudflared creates a secure tunnel to expose the local development environment to the internet.

This setup enables development and testing without deploying to a production server, while still making the application accessible online through the Cloudflare tunnel.

3.3. Algorithm Details

The system for KharchaTrack employs two key algorithms for its functionalities: Moving Average for expense forecasting and Isolation Forest for anomaly detection.

1. **Moving Average Algorithm (for Expense Forecasting):**

The **Moving Average** (MA) algorithm is used to predict future expenses based on past spending behavior. The MA method smooths out fluctuations in the data to provide a trend line that represents average expenditure over a set period. This helps users forecast their future expenses, enabling better budget management.

Steps:

- **Data Collection:** The algorithm collects the user's expense data over a specified time period.
- **Calculation of Average:** It calculates the average of expenses over the given period (e.g., weekly, monthly).
- **Prediction:** The algorithm then uses the average to forecast future expenses, making the assumption that spending patterns will remain relatively consistent.

2. **Isolation Forest Algorithm (for Anomaly Detection):**

The **Isolation Forest** algorithm is used to detect outliers or unusual expense patterns, which could indicate fraudulent activity or errors in data entry. This algorithm isolates anomalies by recursively partitioning the data into smaller subsets, ensuring that anomalies are more likely to be isolated in fewer partitions compared to normal data.

Steps:

- **Data Splitting:** The algorithm divides the expense data into partitions, progressively isolating subsets of data.
- **Scoring:** It assigns a score to each transaction based on how isolated it is from the rest of the data. Transactions that are isolated early on are flagged as anomalies.
- **Detection:** Anomalous or unusual transactions are identified and flagged for review. These transactions might indicate unexpected spending or potential fraud.

In summary, The **Moving Average** algorithm helps forecast future expenses by calculating the average of past spending patterns, providing users with useful insights for budgeting. Meanwhile, the **Isolation Forest** algorithm detects outliers or anomalies, ensuring that any unexpected or potentially fraudulent expenses are flagged for further investigation. These algorithms work together to provide accurate forecasting and enhance the security of the expense tracking process in **KharchaTrack**.

Chapter 4: Implementation and Testing

4.1. Implementation

4.1.1. Tools used

For the development of **KharchaTrack**, several tools and technologies were utilized to ensure a smooth development process and efficient system performance.

- **Visual Studio Code:** Used as the primary code editor for development. It provided debugging tools, terminal access, and plugin support which significantly enhanced productivity.
- **Laravel Framework:** A PHP-based web application framework used for back-end development, offering features like routing, Eloquent ORM, and Blade templating.
- **Livewire:** Integrated with Laravel to build dynamic interfaces using server-side rendering, allowing for real-time interaction without writing JavaScript.
- **PHP-ML:** A PHP machine learning library used to implement the Moving Average algorithm for expense forecasting and Isolation Forest for anomaly detection.
- **PostgreSQL:** Served as the relational database management system to store and manage user and expense data securely and efficiently.
- **Git and GitHub:** Used for version control and collaborative code management. GitHub also served as a backup repository.

4.1.2. Implementation details of modules

1. Core Modules

1.1 Authentication Module

The system uses Laravel's built-in authentication with some customizations:

```
// app/Models/User.php
class User extends Authenticatable {
    use HasApiTokens, HasFactory, Notifiable;

    protected $fillable = [
        'name', 'email', 'password', 'previously_verified'
    ];

    // Relationships
    public function expenses() {
        return $this->hasMany(Expense::class);
    }

    public function categories() {
        return $this->hasMany(Category::class);
    }
}
```

1.2 Expense Management Module

The core expense tracking functionality is implemented through:

```
// app/Models/Expense.php
class Expense extends Model {
    protected $fillable = [
        'user_id', 'category_id', 'title', 'amount',
        'description', 'date', 'is_recurring', 'is_anomaly'
    ];

    protected $casts = [
        'amount' => 'decimal:2',
        'date' => 'date',
    ];
}
```

```

        'is_recurring' => 'boolean',
        'is_anomaly' => 'boolean'
    ];

    // History tracking
    protected static function booted() {
        static::created(function ($expense) {
            ExpenseHistory::create([
                'expense_id' => $expense->id,
                'user_id' => $expense->user_id,
                'category_id' => $expense->category_id,
                'title' => $expense->title,
                'amount' => $expense->amount,
                'date' => $expense->date,
                'action' => 'create'
            ]);
        });
    }
}

```

1.3 Category Management

```

// app/Models/Category.php
class Category extends Model {
    protected $fillable = [
        'user_id', 'name', 'color', 'description'
    ];

    public function expenses() {
        return $this->hasMany(Expense::class);
    }
}

```


2. Machine Learning Features

2.1 Expense Forecasting Algorithm

The system uses Moving Average algorithm for expense prediction:

```
// app/Services/ForecastingService.php
```

```
class ForecastingService {  
    public function calculateMovingAverage($expenses, $period = 3) {  
        $amounts = $expenses->pluck('amount')->toArray();  
        $movingAverages = [];  
  
        for ($i = $period - 1; $i < count($amounts); $i++) {  
            $sum = 0;  
            for ($j = 0; $j < $period; $j++) {  
                $sum += $amounts[$i - $j];  
            }  
            $movingAverages[] = $sum / $period;  
        }  
  
        return $movingAverages;  
    }  
  
    public function generateForecast($userId) {  
        $expenses = Expense::where('user_id', $userId)  
            ->orderBy('date')  
            ->get();  
  
        $forecast = $this->calculateMovingAverage($expenses);  
  
        return Forecast::create([  
            'user_id' => $userId,  
            'prediction' => json_encode($forecast),  
            'type' => 'monthly'  
        ]);  
    }  
}
```

2.2 Anomaly Detection Algorithm

The system uses Z-score statistics for detecting unusual expenses:

```
// app/Services/AnomalyDetectionService.php
```

```
class AnomalyDetectionService {  
    private const THRESHOLD = 2.0;  
  
    public function detectAnomalies($expenses) {  
        $amounts = $expenses->pluck('amount')->toArray();  
        $mean = array_sum($amounts) / count($amounts);  
        $stdDev = $this->calculateStdDev($amounts, $mean);  
  
        foreach ($expenses as $expense) {  
            $zScore = ($expense->amount - $mean) / $stdDev;  
  
            if (abs($zScore) > self::THRESHOLD) {  
                Anomaly::create([  
                    'user_id' => $expense->user_id,  
                    'expense_id' => $expense->id,  
                    'score' => $zScore,  
                    'reason' => 'Amount significantly deviates from average'  
                ]);  
            }  
        }  
    }  
  
    private function calculateStdDev($amounts, $mean) {  
        $variance = array_reduce($amounts, function($carry, $amount) use ($mean) {  
            return $carry + pow($amount - $mean, 2);  
        }, 0) / count($amounts);  
  
        return sqrt($variance);  
    }  
}
```

3. Frontend Implementation

3.1 Dashboard Component

```
// resources/views/livewire/dashboard.blade.php
```

```
class Dashboard extends Component {  
    public $totalExpenses;  
    public $monthlySummary;  
    public $categorySummary;  
  
    public function mount() {  
        $this->loadDashboardData();  
    }  
  
    public function loadDashboardData() {  
        $this->totalExpenses = auth()->user()->expenses()  
            ->whereMonth('date', now()->month)  
            ->sum('amount');  
  
        $this->monthlySummary = $this->calculateMonthlySummary();  
        $this->categorySummary = $this->calculateCategorySummary();  
    }  
}
```

3.2 Expense Management Component

```
// resources/views/livewire/expenses/index.blade.php
```

```
class ExpensesIndex extends Component {  
    public $expenses;  
    public $categories;  
    public $search = "";  
    public $dateFrom;  
    public $dateTo;  
  
    protected $queryString = [  
        'search' => ['except' => ""],  
        'dateFrom' => ['except' => ""],  
    ]
```

```

        'dateTo' => ['except' => "']
    ];

    public function mount() {
        $this->loadExpenses();
    }

    public function loadExpenses() {
        $this->expenses = auth()->user()
            ->expenses()
            ->when($this->search, function($query) {
                $query->where('title', 'like', "% {$this->search} %");
            })
            ->when($this->dateFrom, function($query) {
                $query->where('date', '>=', $this->dateFrom);
            })
            ->when($this->dateTo, function($query) {
                $query->where('date', '<=', $this->dateTo);
            })
            ->orderBy('date', 'desc')
            ->get();
    }
}

```

4. Algorithm Implementation Details

4.1 Moving Average Algorithm

- **Purpose:** Predict future expenses based on historical patterns
- **Implementation Steps:**
 - i. Collect historical expense data for specified period
 - ii. Calculate simple moving average using sliding window
 - iii. Generate predictions for next period
 - iv. Store predictions in forecast table

4.2 Z-Score Anomaly Detection

- **Purpose:** Identify unusual spending patterns
- **Implementation Steps:**

- i. Calculate mean and standard deviation of expenses
- ii. Compute z-score for each expense
- iii. Flag expenses with z-score beyond threshold
- iv. Store anomaly records with explanation

This implementation guide provides a solid foundation for building the KharchaTrack system. Each module is designed to be maintainable and scalable, with clear separation of concerns and proper use of Laravel's features.

4.2. Testing

Below is a comprehensive set of test cases for the Kharcha Track Web application, organized by feature area in tabular form:

1. Authentication Tests					
Test ID	Test Case Description	Test Steps	Expected Result	Priority	Test Type
AUTH-001	User registration with valid data	1. Navigate to register page 2. Fill valid user data 3. Submit form	User is registered and redirected to dashboard	High	Functional
AUTH-002	Login with valid credentials	1. Navigate to login page 2. Enter valid credentials 3. Click login	User is authenticated and redirected to dashboard	High	Functional
AUTH-003	Forgot password functionality	1. Click "Forgot Password" 2. Enter registered email 3. Submit form	Reset password email is sent	Medium	Functional
AUTH-004	Email verification process	1. Register new account 2. Open verification email 3. Click verification link	Email is marked as verified and user is redirected	High	Functional
AUTH-005	Login with unverified email	1. Create user with unverified email 2. Attempt to login	User is redirected to verification notice page	Medium	Functional

2. Expense Management Tests

Test ID	Test Case Description	Test Steps	Expected Result	Priority	Test Type
EXP-001	Create new expense	1. Navigate to expenses/create 2. Fill expense details 3. Submit form	Expense is created and appears in expense list	High	Functional
EXP-002	Edit existing expense	1. Navigate to expense edit page 2. Modify expense details 3. Save changes	Expense is updated with new details	High	Functional
EXP-003	Historical tracking of expense updates	1. Create expense 2. Edit expense 3. Check expense history	History record is created for both actions	Medium	Functional
EXP-004	Filter expenses by date range	1. Go to expenses list 2. Select date range 3. Apply filter	Only expenses within range are shown	Medium	Functional
EXP-005	Filter expenses by category	1. Go to expenses list 2. Select a category 3. Apply filter	Only expenses in selected category are shown	Medium	Functional

3. Category Management Tests

Test ID	Test Case Description	Test Steps	Expected Result	Priority	Test Type
CAT-001	Create new category	1. Navigate to categories page 2. Click add category 3. Fill category details and save	New category is created and appears in list	High	Functional
CAT-002	Edit existing category	1. Go to categories page 2. Select edit for a category 3. Update details and save	Category is updated with new details	Medium	Functional
CAT-003	Delete category	1. Go to categories page 2. Select delete for a category 3. Confirm deletion	Category is removed from system	Medium	Functional

Test ID	Test Case Description	Test Steps	Expected Result	Priority	Test Type
CAT-004	Assign expense to category	1. Create/edit expense 2. Select category from dropdown 3. Save expense	Expense is associated with selected category	High	Functional
CAT-005	View expenses by category	1. Navigate to category detail view 2. Check list of expenses	All expenses for category are displayed	Medium	Functional

4. Forecasting Tests

Test ID	Test Case Description	Test Steps	Expected Result	Priority	Test Type
FOR-001	View expense forecast page	1. Navigate to forecast page 2. Verify forecast components load	Forecast page displays with expected components	High	Functional
FOR-002	Generate overall expense forecast	1. Go to forecast page 2. Have sufficient historical data 3. Check forecast results	Next 3 months forecast is displayed	High	Functional
FOR-003	Generate category-specific forecasts	1. Go to forecast page 2. Have expenses with categories 3. Check category forecasts	Category-specific forecasts are displayed	Medium	Functional
FOR-004	Toggle past forecasts visibility	1. Go to forecast page 2. Click toggle for past forecasts 3. Check if past forecasts appear	Past forecasts visibility toggles correctly	Low	Functional
FOR-005	Change forecast months range	1. Go to forecast page 2. Change forecast months setting 3. Check if forecast updates	Forecast updates to show selected number of months	Medium	Functional

5. Anomaly Detection Tests

Test ID	Test Case Description	Test Steps	Expected Result	Priority	Test Type
ANM-001	View anomaly detection page	1. Navigate to anomaly page 2. Check page loads correctly	Anomaly page shows expected components	High	Functional
ANM-002	Detect anomalies in spending	1. Create normal expenses 2. Create one unusually high expense 3. Run anomaly detection	Unusually high expense is flagged as anomaly	High	Functional
ANM-003	Review and confirm anomaly	1. View detected anomaly 2. Click to review 3. Confirm as anomaly	Anomaly is marked as reviewed and confirmed	Medium	Functional
ANM-004	Review and reject anomaly	1. View detected anomaly 2. Click to review 3. Mark as not an anomaly	Anomaly is marked as reviewed but not confirmed	Medium	Functional
ANM-005	Anomaly visualization	1. Go to anomaly page 2. View scatter plot visualization 3. Check anomalies highlighted	Anomalies are visually highlighted in chart	Medium	UI

6. User Profile Tests

Test ID	Test Case Description	Test Steps	Expected Result	Priority	Test Type
USR-001	View user profile	1. Login 2. Navigate to profile page 3. Verify profile information	Profile page shows correct user information	Medium	Functional
USR-002	Update profile information	1. Go to profile page 2. Edit profile details 3. Save changes	Profile is updated with new information	Medium	Functional

Test ID	Test Case Description	Test Steps	Expected Result	Priority	Test Type
USR-003	Change password	1. Go to profile page 2. Input current and new password 3. Save changes	Password is updated successfully	High	Functional
USR-004	Update email address	1. Go to profile page 2. Change email address 3. Submit form	Verification email is sent to new address	Medium	Functional

7. Admin Tests

Test ID	Test Case Description	Test Steps	Expected Result	Priority	Test Type
ADM-001	Access admin user management	1. Login as admin 2. Navigate to admin/users 3. Check page loads	User management page displays correctly	High	Functional
ADM-002	Manage user roles	1. Go to admin/roles 2. Attempt to add/edit roles 3. Save changes	Roles are updated successfully	High	Functional
ADM-003	Manage permissions	1. Go to admin/permissions 2. Attempt to modify permissions 3. Save changes	Permissions are updated successfully	High	Functional
ADM-004	Admin access restriction	1. Login as regular user 2. Attempt to access admin URLs	User is denied access to admin pages	High	Security

8. Integration Tests

Test ID	Test Case Description	Test Steps	Expected Result	Priority	Test Type
INT-001	Expense-Category relationship	1. Create category 2. Create expense with category 3. View category details	Expense appears in category's expense list	High	Integration
INT-002	Forecast based on expense history	1. Create expenses for 3+ months	Forecast updates based on new expense data	High	Integration

Test ID	Test Case Description	Test Steps	Expected Result	Priority	Test Type
		2. View forecasts 3. Add more expenses and refresh			
INT-003	User expense isolation	1. Create two users 2. Add expenses for each user 3. Login as first user	Only first user's expenses are visible	High	Integration
INT-004	End-to-end expense workflow	1. Create category 2. Add expense to category 3. Edit expense 4. View forecast 5. Check anomaly detection	All components interact correctly	High	E2E

Chapter 5: Conclusion and Future Recommendations

5.1. Lesson Learnt / Outcome

From this project, I learned about project management, how the coding structure is maintained and how programming is done on the projects. I also learned to build the diagrams that are needed for the system. I learned many things which can lead us to do any work related to projects further.

5.2. Conclusion

The development of **KharchaTrack** incorporates predictive analytics and anomaly detection through machine learning algorithms. This enables users not only to log and categorize expenses but also to forecast future spending trends and flag suspicious or irregular transactions.

The integration of Laravel, Livewire, and PHP-ML facilitated the development of a dynamic and scalable platform. The system's personalized dashboard, real-time processing capabilities, and user-friendly interface contributed to a rich user experience.

Overall, KharchaTrack fulfills its objective of helping users make smarter, data-driven financial decisions while maintaining an intuitive and accessible platform. The project demonstrates how machine learning can be effectively used in real-life applications, especially in financial domains.

5.3. Future Recommendations

To further enhance KharchaTrack, the following features and improvements are recommended:

- Integration with Financial APIs: Linking with bank APIs to auto-import and sync transactions can significantly reduce manual data entry and improve real-time tracking.
- Progressive Web App (PWA) Conversion: Making the system installable as a PWA for improved mobile accessibility and offline capabilities.
- SMS/Email Alerts: Implementing notification systems to alert users about upcoming predicted expenses or flagged anomalies.
- Multi-user/Household Budgeting: Supporting collaborative budgeting for families or teams.

- Integration with Digital Wallets: Allowing syncing with eSewa, Khalti, or international wallets for automatic transaction logging.
- Enhanced Security Features: Including two-factor for improved user data protection.

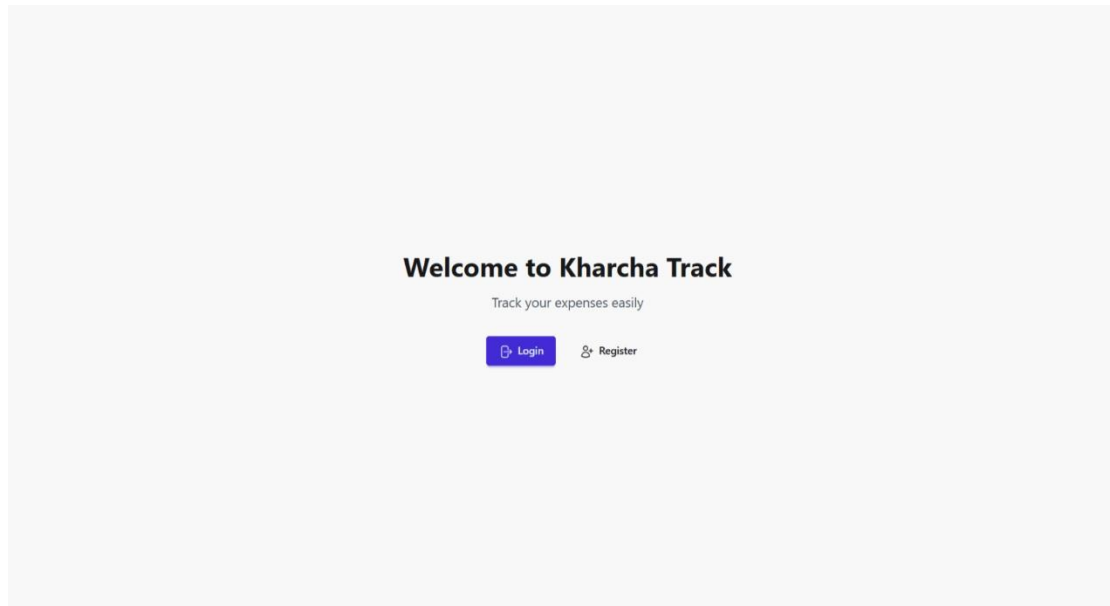
These enhancements would make KharchaTrack a more comprehensive and competitive personal finance management tool.

References

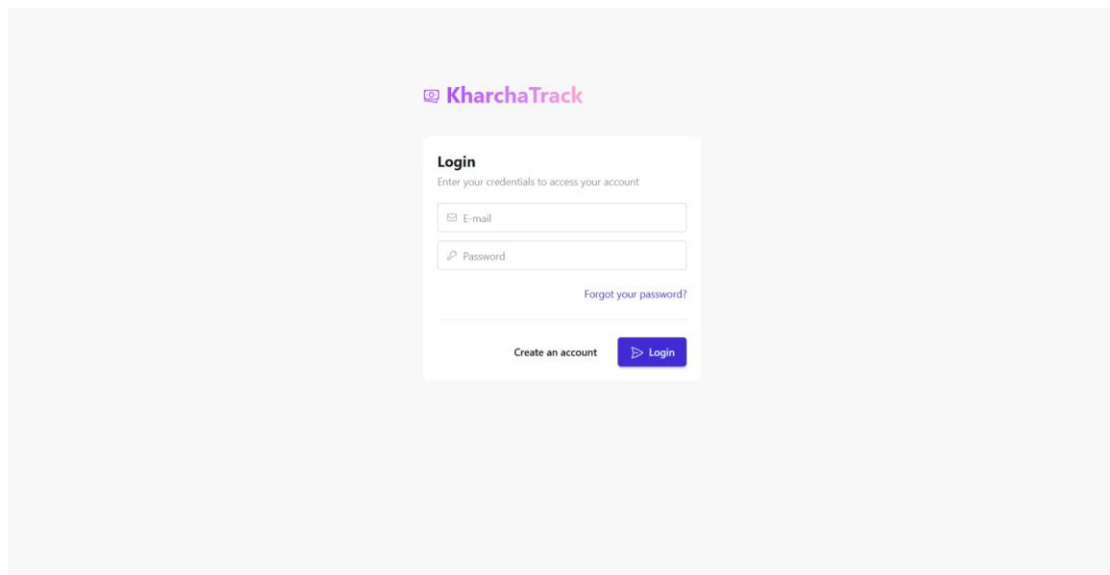
- [1] I. Intuit, “Mint: Budget, Bills, and Finance Tracker,” [Online]. Available: <https://mint.intuit.com/>. [Accessed: Apr. 18, 2025].
- [2] You Need A Budget, “YNAB – You Need A Budget,” [Online]. Available: <https://www.youneedabudget.com/>. [Accessed: Apr. 18, 2025].
- [3] Expensify, “Expensify - Expense Reports That Don’t Suck!,” [Online]. Available: <https://www.expensify.com/>. [Accessed: Apr. 18, 2025].
- [4] A. Parker, “From Paper to Digital: The Evolution of Budgeting and Expense Tracking,” Medium, Aug. 2, 2023. [Online]. Available: <https://medium.com/@a.parker/evolution-of-budgeting-tools>. [Accessed: May 14, 2025].
- [5] Quicken, “Budgets That Work,” Quicken.com. [Online]. Available: <https://www.quicken.com/>. [Accessed: May 14, 2025].
- [6] Investopedia, “Moving Average (MA): Purpose, Uses, Formula, and Examples,” updated Aug. 6, 2024. [Online]. Available: <https://www.investopedia.com/terms/m/movingaverage.asp>. [Accessed: May 14, 2025].
- [7] Wikipedia, “Isolation Forest,” [Online]. Available: https://en.wikipedia.org/wiki/Isolation_forest. [Accessed: May 14, 2025].
- [8] S. Khan, “Leveraging Machine Learning for Predictive Budgeting and Anomaly Detection in Personal Finance,” SSRN, Apr. 2025. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4477331. [Accessed: May 14, 2025].
- [9] S. Aishwarya and S. Hemalatha, “Smart Expense Tracking System Using Machine Learning,” in Proc. 1st Int. Conf. AIIoT 2023 (AI for IoT), 2023, pp. 634–640.

Appendix


Screenshot of the website



Landing Page



Login Page



Register

Create a new account to get started

Name

E-mail

Password


Confirm Password

Already registered?

>

Register

Register Page



Dashboard

Expenses


Categories

Profile

Regular User


user@example.com

Dashboard




3,064,100.58

Your Total Expenses (NPR)



1

Your Categories

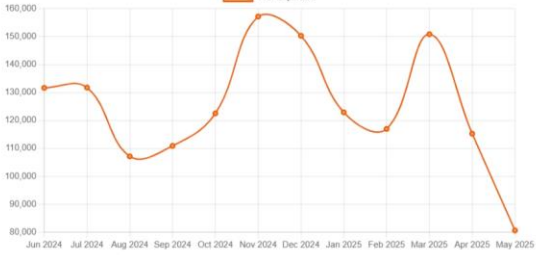


127

Your Anomalies

Your Expense Trend (Last 12 Months)

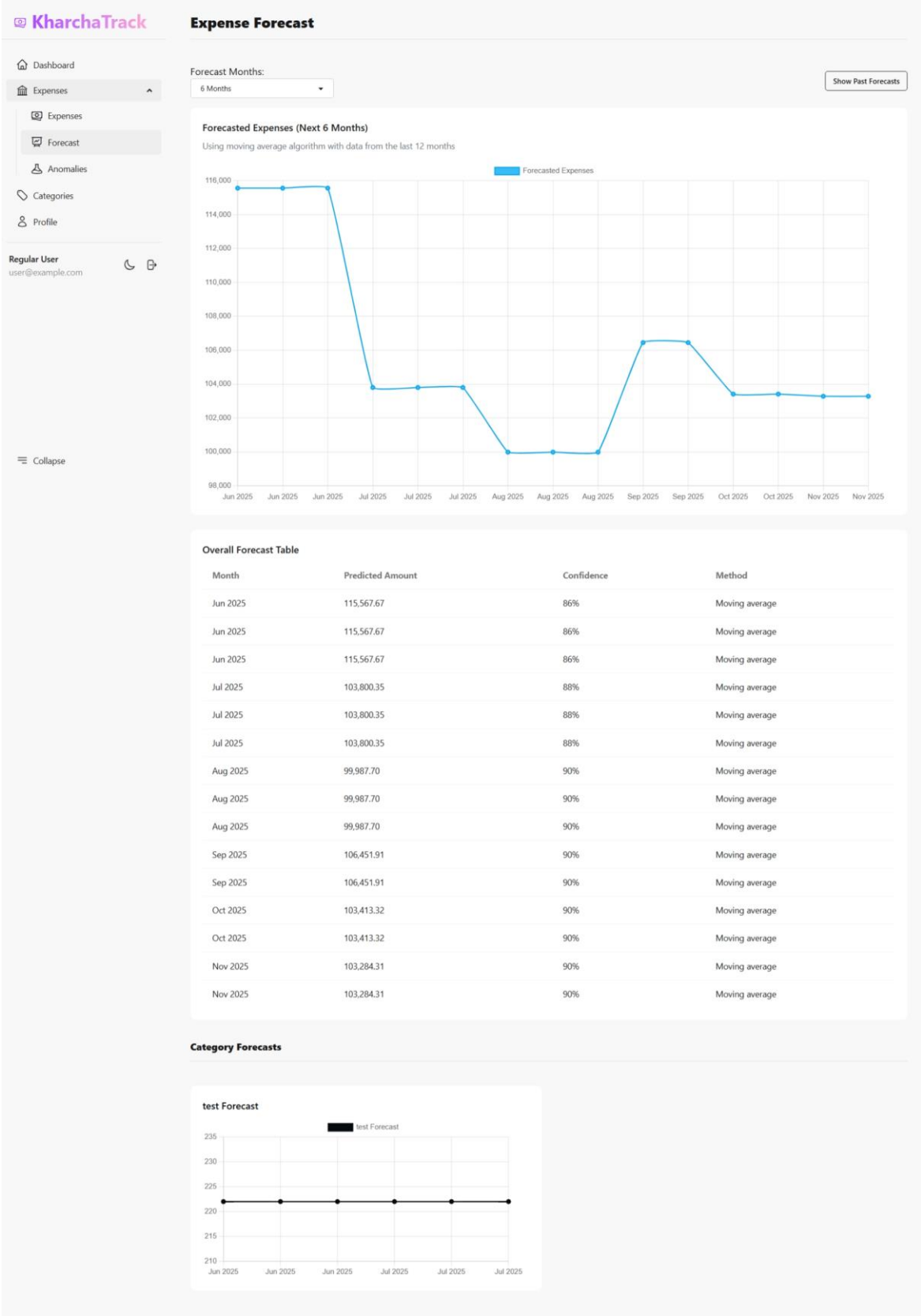
Your Expenses



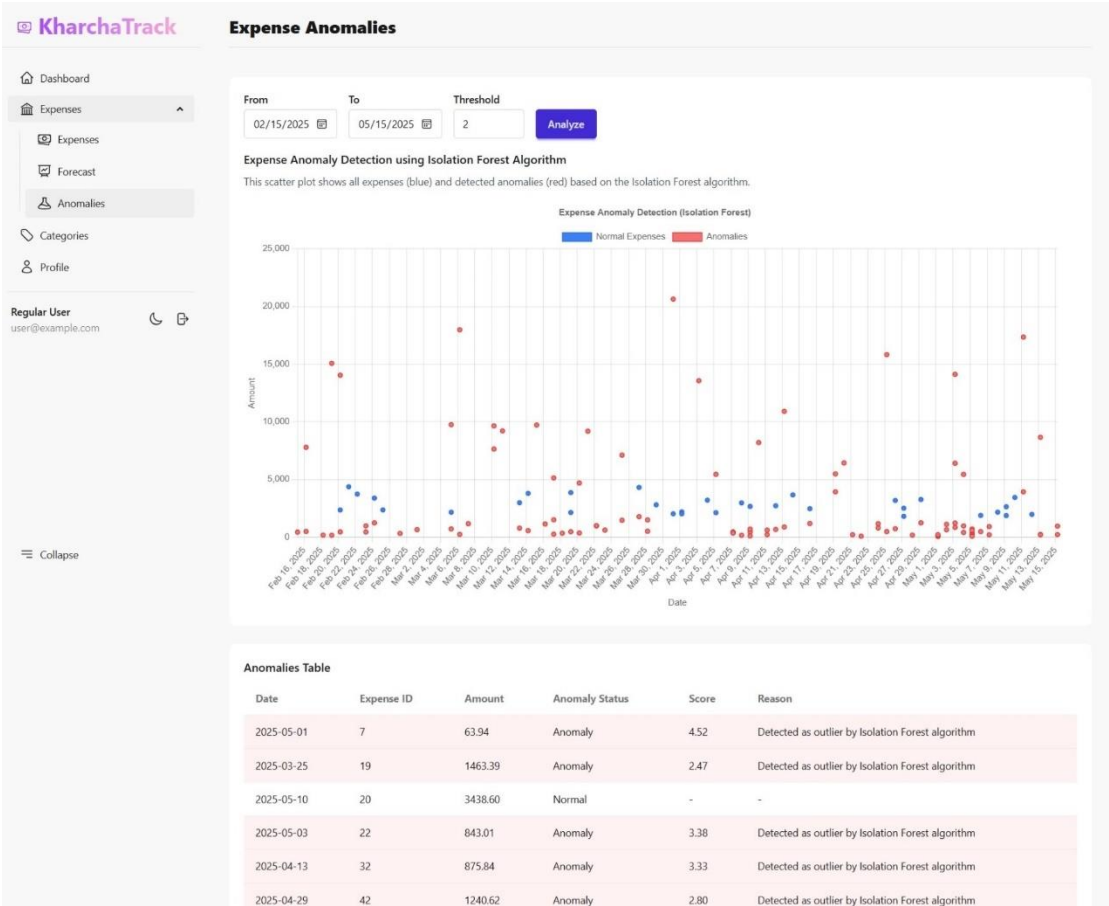
Collapse

User Dashboard

ii



Expense Forecast Page



KharchaTrack

Dashboard

Expenses

Categories

Profile

Regular User

user@example.com

🌙

🔒

Collapse

Profile Information

Update your account's profile information and email address.

Name *

Regular User

Email *

user@example.com

Verified

Current Password

Leave empty if you don't want to change your password

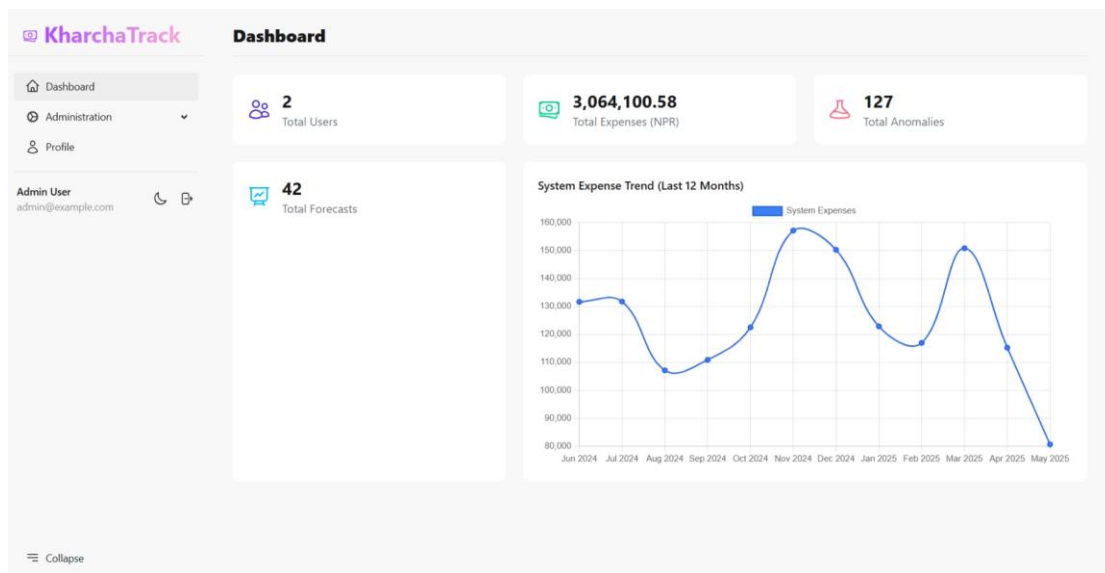
New Password

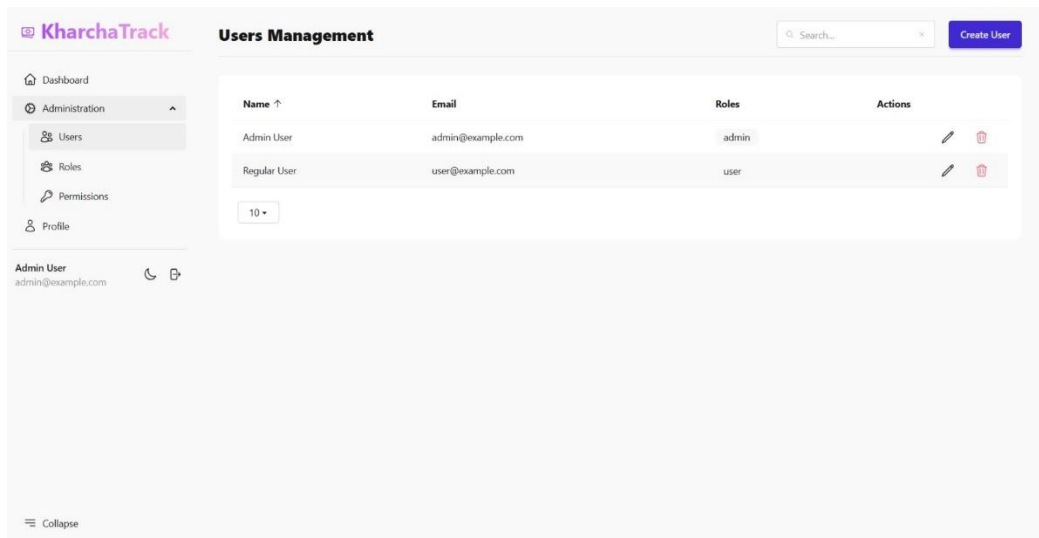
Minimum 8 characters

Confirm New Password

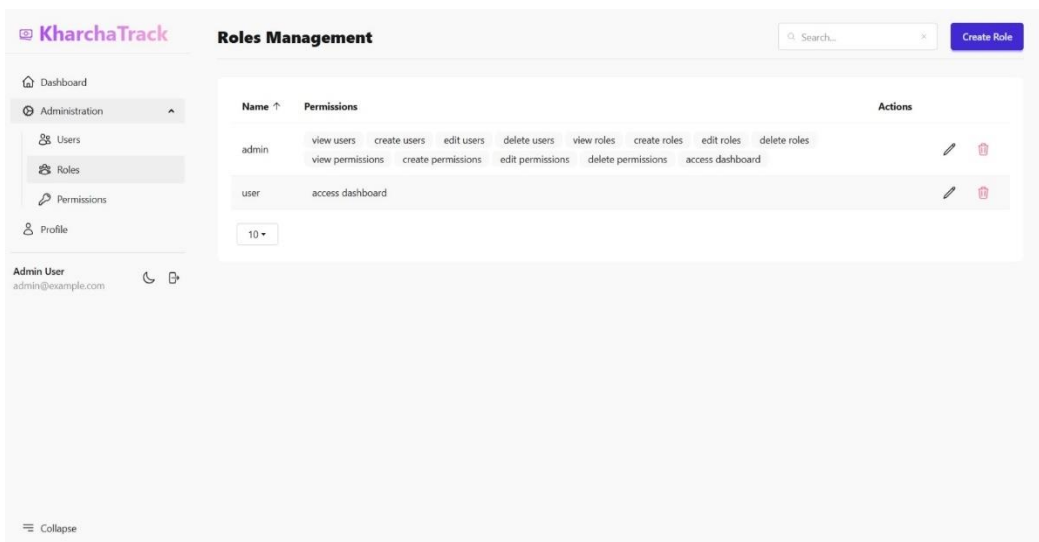
Save Changes

User Profile Page

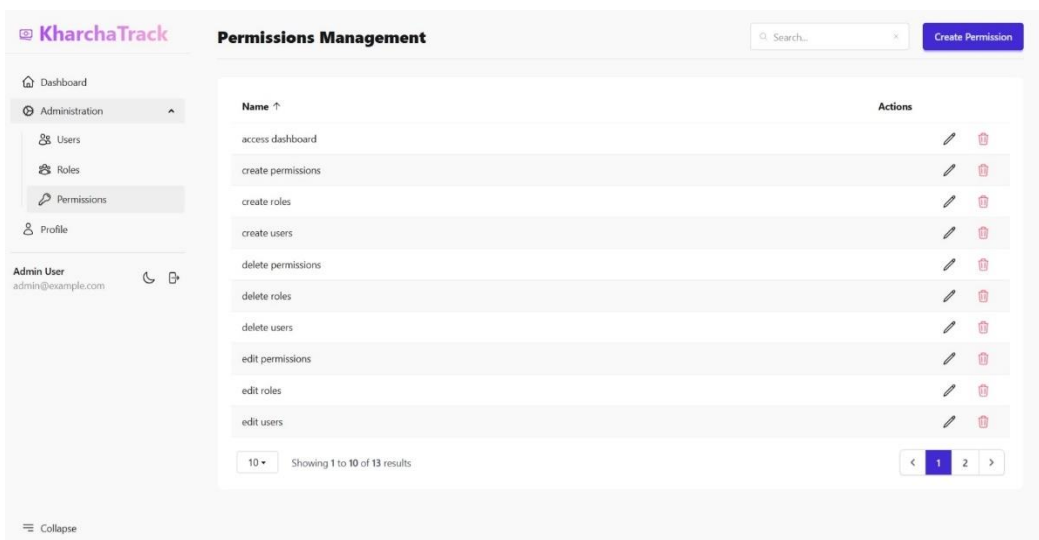




Admin's User Management Page



Admin's Roles Management Page



Admin's Permission Management Page