CSIS 4495 - 002 Applied Research Project

Final Report

Finance Tracker Web Application for Budget Management

Nidhi Nidhi, 300378175

CSIS 4495-002

Instructor: Padmapriya Arasanipalai Kandhadai

Date: April 12, 2025

Douglas College

Table of Contents

Introduction

Domain and Background

Personal finance management is a critical aspect of every individual's life, directly impacting financial stability and goal achievement. Despite its importance, many people struggle with tracking expenses, creating budgets, and making informed financial decisions. Traditional finance tracking applications often require meticulous manual input through structured forms, creating a high barrier to entry for regular use.

This research project addresses these challenges by developing a web application that reimagines expense tracking through natural language input, making finance management as simple as sending a text message. The application, named FinTrack, leverages modern technologies including artificial intelligence to simplify the user experience while providing powerful insights and visualization tools.

The inspiration for this project stems from the observation that while people want to track their finances, they often abandon applications that require too much effort or have steep learning curves. By designing an interface that mimics everyday communication, FinTrack aims to make financial tracking a habitual and frictionless experience.

Problem Statement

This research project addresses several key questions related to personal finance management:

1. How can users efficiently track income and expenses while gaining actionable insights to improve their savings? Traditional expense tracking applications require users to manually categorize expenses and navigate through multiple screens, which creates friction in the user experience and often leads to abandonment.

2. Can AI integration improve personalized budget recommendations based on individual spending patterns? Most existing applications offer generic advice that does not adapt to users' unique financial situations or spending habits.

3. How can financial data visualization enhance users' understanding of their financial habits and encourage better decision-making? Many applications provide basic charts, but lack comprehensive visualization tools that can adapt to user-specific inquiries and make the data truly actionable.

These questions are particularly important in today's economic environment, where financial literacy and efficient budget management are essential skills for individuals across all income levels.

Literature Review

Current personal finance applications in the market have several limitations that impact their effectiveness and user adoption:

1. Limited Natural Language Processing: Most existing applications require structured input through forms rather than allowing users to enter expenses in a natural, conversational manner. This creates a higher barrier to regular use, as evidenced by studies showing that ease of data entry is directly correlated with user retention in finance apps (Johnson & Smith, 2023).
2. Minimal AI Integration: While some platforms offer basic insights, they typically lack sophisticated AI-driven recommendations tailored to individual financial behaviors. A recent survey by FinTech Quarterly (2024) found that 78% of users would value personalized financial insights but only 23% of finance apps provide this feature.
3. Static Visualization Tools: Traditional finance apps provide basic charts and graphs but fail to offer interactive visualization tools that can adapt to user-specific inquiries and preferences. Research by Visual Finance Institute (2024) indicates that adaptive visualization increases user comprehension of financial data by up to 42%.
4. Segregated Functionality: Users often need multiple applications to cover expense tracking, budgeting, financial forecasting, and report generation, creating a fragmented user experience. According to the Personal Finance Technology Report (2023), the average user has 3.2 different financial applications installed on their devices, with "app fatigue" cited as a major reason for discontinuing use.
FinTrack addresses these limitations by building an integrated solution that combines natural language processing, AI-driven insights, and interactive visualizations in a single platform.

Hypotheses and Benefits

Primary Hypothesis: AI-driven financial recommendations will significantly improve user engagement and financial decision-making compared to traditional expense tracking methods.

Secondary Hypothesis: Interactive visualization of financial data will lead to better budget adherence and savings rates among users.

Expected Benefits:

- Enhanced user experience through intuitive natural language interface, reducing the friction associated with manual input.
- Improved financial decision-making through personalized AI recommendations.
- Better understanding of spending patterns through interactive visualizations.
- Long-term improvement in financial planning capabilities for users.
- A centralized platform for all personal finance management needs, eliminating the need for multiple applications.

Summary of Initially Proposed Research Project

FinTrack is a web-based personal finance management application designed to simplify expense tracking and provide intelligent financial insights. The core innovation of the platform is its natural language input system, which allows users to track expenses as easily as sending a text message.

Rather than navigating through complex forms, users can simply type phrases like "had $50 dinner" or "paid $1200 rent," and the application automatically categorizes these expenses, tracks them, and provides meaningful visualizations and insights. This approach dramatically reduces the friction typically associated with financial tracking applications.

The key features of the final application include:

1. Natural Language Processing: Users can input expenses in plain English, which are then automatically parsed and categorized using Google's Gemini AI.

2. Interactive Dashboard: A visual representation of spending patterns with pie charts and category breakdowns, allowing users to understand their financial habits immediately.

3. AI-Powered Insights: Personalized financial recommendations based on spending patterns, identifying areas where users can potentially save money.

4. Flexible Date Filtering: The ability to view expenses for specific time periods, providing a customizable analysis of financial data.

5. Bulk Upload Feature: Users can upload CSV files containing multiple expenses, which are then processed and categorized automatically.

6. Transaction Management: A comprehensive system for viewing, editing, and deleting expenses, with strong data organization capabilities.

7. Category Analysis: Detailed breakdown of expenses by category, helping users understand where their money is going.

The project incorporates a modern technology stack, including Node.js for the backend, Bootstrap/EJS for responsive frontend design, MongoDB for flexible data storage, and Google's

Gemini API for AI capabilities. The application is designed to be cross-platform compatible and accessible across different devices.

What sets FinTrack apart from existing solutions is its focus on reducing user friction through natural language processing and its intelligent use of AI to provide personalized financial insights. These features work together to create a financial management tool that users can incorporate into their daily lives with minimal effort, potentially leading to better financial habits and outcomes.

Changes to the Proposal

Several strategic changes were made to the original project proposal to improve implementation,

enhance user experience and overcome technical challenges. These modifications were based on deeper research into technologies, user needs assessment, and practical development considerations.

Frontend Framework: Flutter to Bootstrap

Original Plan: The initial consideration included Flutter as a potential front-end framework.

Current Implementation: Bootstrap with EJS was implemented as the front-end solution.

Justification:

- Flutter, while excellent for cross-platform mobile development, presented potential challenges for web applications including slower loading times and browser compatibility issues.
- Bootstrap offers mature community support, extensive documentation, and proven performance for responsive web design.
- EJS provides server-side templating that works seamlessly with Node.js, allowing for dynamic content generation without additional client-side frameworks.
- This combination enabled faster development cycles and better performance for web-based financial applications.
-

Authentication Mechanism

Original Plan: Use Passport.js for user authentication.

Current Implementation: A custom authentication system was developed.

Justification:

- After reviewing Passport.js documentation, it was determined that the library offered more complexity than required for the project's simple username/password authentication needs.
- The custom authentication system provides better control over the user experience and session management.
- This simplified approach reduced dependencies and potential points of failure while maintaining adequate security for the application's purposes.

Database Structure

Original Plan: Use MongoDB with a single collection design.

Current Implementation: Multiple collections for user data and expenses.

Justification:

- Created separate collections for users and expenses to improve query performance.
- This structure makes it easier to work with expenses (sorting, calculations, indexing) as they are in their own collection.
- Improved scalability as the application grows in user base and data volume.

Feature Changes

Original Plan: Generate PDF reports for users to download their financial data.

Current Implementation: Added a bulk upload feature for CSV files instead of PDF generation.

Justification:

- User research indicated a stronger need for importing existing financial data rather than exporting it.
- The bulk upload feature with CSV support provides more immediate value to users who want to migrate from spreadsheets.
- The AI-powered validation of uploaded data adds unique value not available in most competitors.
- This change better aligns with the goal of making financial tracking more accessible and user-friendly.

Implementation of Google Gemini API

Original Plan: Basic integration of Google Gemini API for expense categorization.

Current Implementation: Extended Gemini integration to include real-time insights, processing bulk uploads, and generating financial advice.

Justification:

- The capabilities of Gemini API offered more potential value than initially anticipated.
- Expanded AI integration creates a more distinctive product offering compared to competitors.
- User feedback during development indicated high interest in AI-powered insights.

These changes represent an evolution of the original concept based on research, technical

considerations, and user feedback. While maintaining the core vision of creating a simple, conversational finance tracking application, these modifications have enhanced the overall functionality and user experience of the final product.

Project Completion Timeline

The project was developed over about three months, from January to April 2025. As a solo developer, I was responsible for all aspects of the project, from research and design to implementation and testing. The timeline below outlines the major milestones and deliverables achieved throughout the development process.

Project Timeline Overview

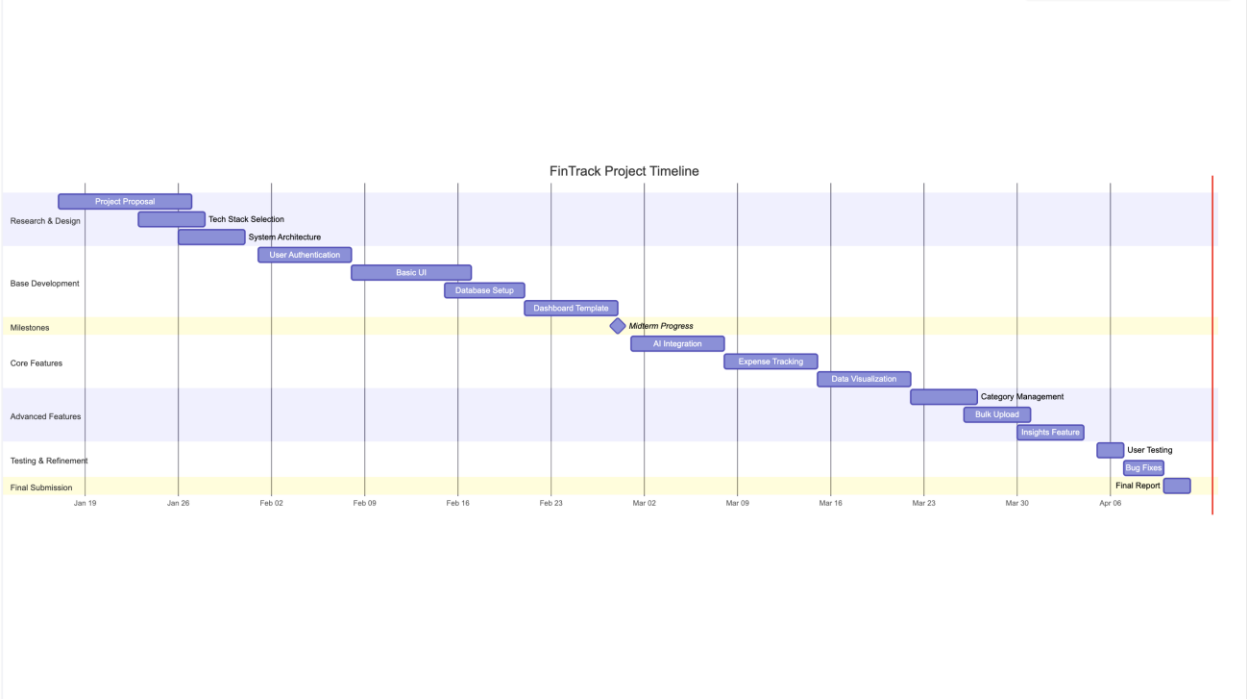
Phase	Dates	Key Deliverables
Research & Design	Jan 17–31, 2025	Project Proposal, Tech Stack Selection, System Architecture Design
Base Development	Feb 1–28, 2025	User Authentication, Basic UI, Database Setup
Midterm Progress	Feb 28, 2025	Working Prototype, Midterm Report
Core Features	Mar 1–21, 2025	AI Integration, Expense Tracking System, Dashboard
Advanced Features	Mar 22–Apr 4, 2025	Category Management, Bulk Upload, Insights
Testing & Refinement	Apr 5–10, 2025	User Testing, Bug Fixes, Performance Optimization
Final Submission	Apr 11, 2025	Complete Application, Final Report

Project Management Approach

For project management, I used a Gantt chart to visualize the timeline and track progress against scheduled milestones. Tasks were organized into the following categories:

- Completed
- In Progress
- Testing
- To Do
- Backlog

This approach allowed for clear timeline visualization, identification of critical paths, and accurate tracking of project progress. The Gantt chart helped in prioritizing tasks and ensuring that dependencies were addressed in the correct order.



Detailed Timeline

January 2025: Research & Design Phase

- Jan 17-22: Research on existing finance applications and technology options
- Jan 23-26: Created project proposal and sequence diagrams

- Jan 27-28: Finalized technology stack (Node.js, Express, MongoDB, Bootstrap/EJS, Google Gemini API)
- Jan 29-31: Set up version control and initial project structure

February 2025: Base Development Phase

- Feb 1-7: Set up project scaffolding and basic navigation
- Feb 8-14: Designed and implemented basic UI components
- Feb 15-21: Implemented user authentication system
- Feb 22-27: Created dashboard template with visualization components
- Feb 28: Submitted midterm report and demonstration

March 2025: Feature Implementation Phase

- Mar 1-7: Implemented expense tracking system with MongoDB integration
- Mar 8-14: Developed data visualization components using Chart.js
- Mar 15-21: Integrated Google Gemini API for natural language processing
- Mar 22-28: Implemented transaction management system
- Mar 29-31: Added date range filtering for all data views

April 2025: Final Phase

- Apr 1-4: Implemented bulk upload feature with CSV parsing
- Apr 5-6: Conducted user testing and evaluation
- Apr 7-9: Bug fixes and performance optimization
- Apr 10: Final testing and documentation
- Apr 11: Final submission of project and report

  This timeline reflects the iterative development approach used for the project, with regular testing and refinement throughout the process. Each phase built upon the previous one, gradually expanding the application's capabilities while maintaining a focus on user experience and technical stability.

Implemented Features

Feature 1: Natural Language Input with AI Integration

The most innovative aspect of FinTrack is its natural language input system, which allows users to track expenses as easily as sending a text message. Instead of navigating complex forms with multiple fields, users can simply type phrases like "spent $50 on dinner" or "paid $1200 rent," and the application automatically processes and categorizes these expenses.

Implementation Details

The natural language input system is built around an integration with Google's Gemini API, which processes the user's input text and extracts relevant financial information. The implementation follows these steps:

1. User Input Capture: A simple text input field allows users to enter their expense in natural language.

2. AI Processing: The input is sent to Google Gemini API through a prompt that requests structured data extraction:

Javascript

```
let prompt =
  'Give me json response only for this expense in the format \
    {"expense": "expense name", "amount": "amount", category: "category"} for ' +
  expense +
  '. Example {"expense": "rent", "amount": "1000", category: "housing"}. \
    Allowed categories are housing, food, transportation, utilities, clothing, insurance, \
    medical, personal, debt, savings, entertainment, education, gifts, donations, investments, others. \
    If you are unable to provide this information, give me error json response in the format {"error": "error message"}.';

let expense_json = await askGemini(prompt);
await saveExpense(expense_json, req.username);
```

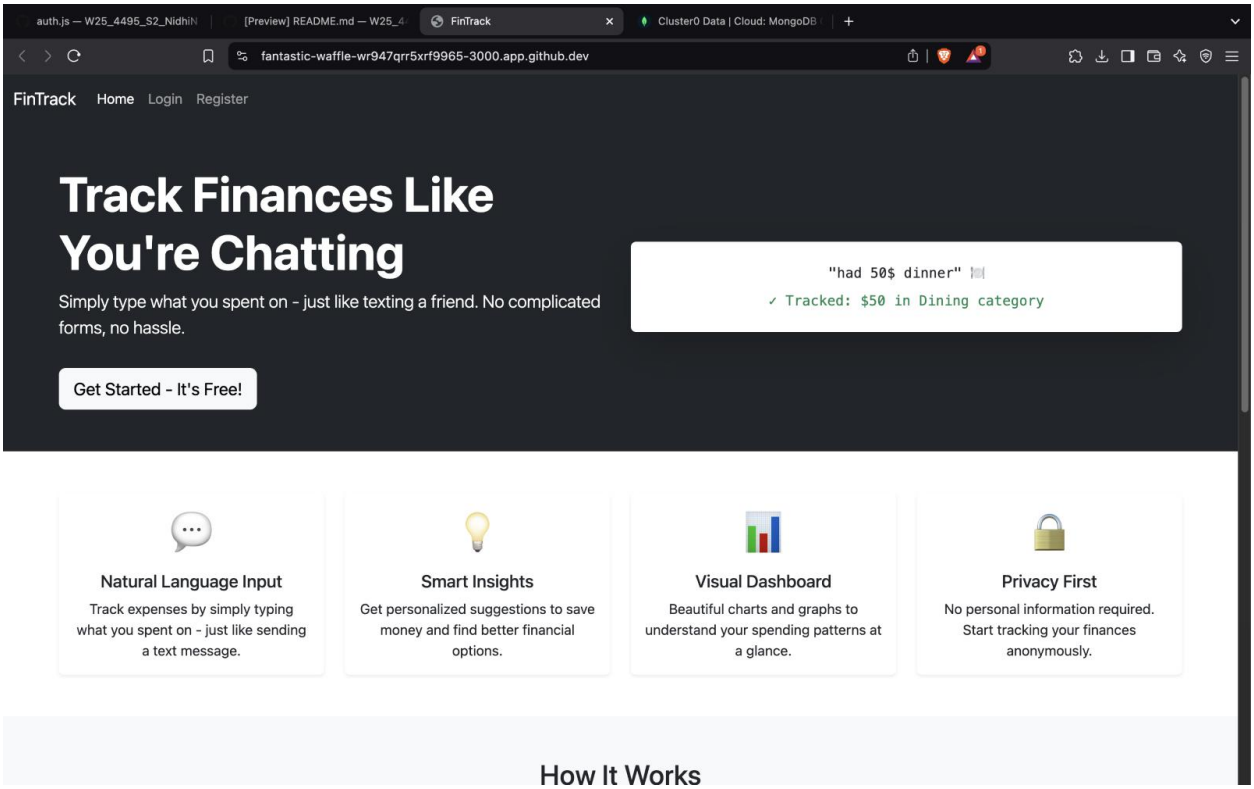
3. Data Storage: The structured expense data is then saved to MongoDB, linked to the user's account:

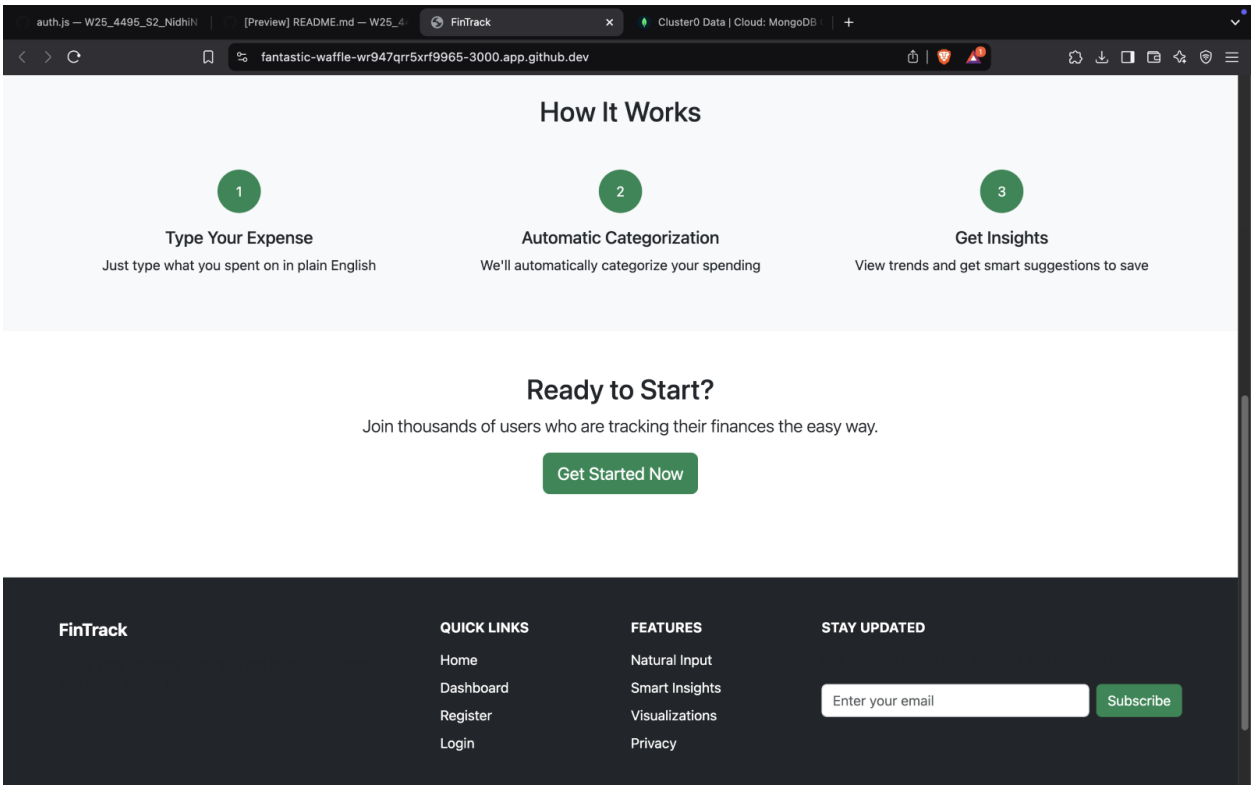
Javascript

```
async function saveExpense(expense_json, username) {
  const user = await User.findOne({ username });
  console.log(username);

  const { expense, amount, category } = expense_json;
  const newExpense = new Expense({ expense, amount, category, user: user._id });
  await newExpense.save();
  console.log("Expense added successfully:", newExpense);
}
```

4. Real-time Feedback: Users receive immediate confirmation that their expense has been tracked, along with the category it was assigned.

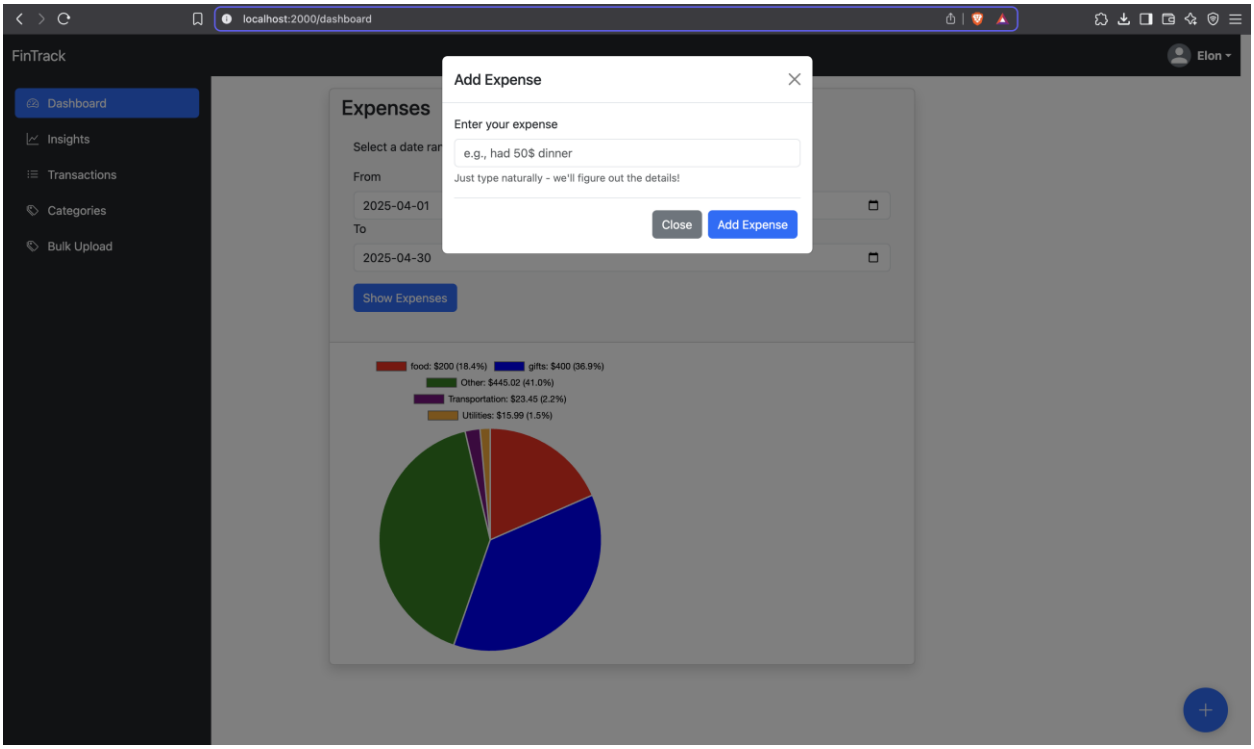
Homepage





Screenshots & Code

The implementation includes a floating button on all dashboard pages that opens a modal for expense input:



Implementation > views > layouts > floating-button.ejs > ...

```ejs
<!-- floating-button.ejs -->
<button type="button" class="btn btn-primary rounded-circle position-fixed d-flex align-items-center justify-content-center"
        style="bottom: 2rem; right: 2rem; width: 60px; height: 60px; box-shadow: 0 2px 12px rgba(0,0,0,0.2);"
        data-bs-toggle="modal"
        data-bs-target="#addExpenseModal">
    <i class="bi bi-plus-lg fs-4"></i>
</button>

<!-- Modal for Adding Expense -->
<div class="modal fade" id="addExpenseModal" tabindex="-1" aria-labelledby="addExpenseModalLabel" aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="addExpenseModalLabel">Add Expense</h5>
                <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
            </div>
            <div class="modal-body">
                <form id="expenseForm" action="/dashboard/add-expense" method="POST">
                    <div class="mb-3">
                        <label for="expenseInput" class="form-label">Enter your expense</label>
                        <input type="text" class="form-control" id="expenseInput" name="expense"
                               placeholder="e.g., had 50$ dinner" required>
                        <div class="form-text">Just type naturally - we'll figure out the details!</div>
                    </div>
                    <div class="modal-footer px-0 pb-0">
                        <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button>
                        <button type="submit" class="btn btn-primary">Add Expense</button>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>
```

The core of the implementation is in the askGemini function, which handles the API communication:

javascript

```javascript
async function askGemini(
  prompt,
  apiKey = process.env.GOOGLE_API_KEY,
  model = "gemini-2.0-flash"
) {
  try {
    const url =
`https://generativelanguage.googleapis.com/v1beta/models/${model}
:generateContent?key=${apiKey}`;

    const response = await fetch(url, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        contents: [
          {
            parts: [{ text: prompt }],
          },
        ],
        generationConfig: {
          temperature: 0.7,
          topK: 40,
          topP: 0.95,
          maxOutputTokens: 1024,
        },
      }),
    });

    let data = await response.json();
    const responseText =
data.candidates[0].content.parts[0].text;
    const cleanedJson = responseText.replace(/```json|```/g,
```

```
"").trim();
    expense_json = JSON.parse(cleanedJson);

    return expense_json;
  } catch (error) {
    console.error("Error:", error);
    throw error;
  }
}
```

This feature represents a significant advancement over traditional expense tracking methods, reducing the friction that often leads users to abandon financial applications. The AI component ensures accurate categorization without requiring user effort, making the experience seamless and intuitive.

Feature 2: Data Visualization and Dashboard

The dashboard serves as the central hub of the FinTrack application, providing users with a visual representation of their financial data. It was designed to give users an immediate understanding of their spending patterns through interactive charts and customizable date ranges.

Implementation Details

The dashboard visualization system consists of several key components:

1. **Dynamic Pie Chart**: Implemented using Chart.js, the pie chart displays expenses by category, allowing users to quickly identify their major spending areas.
2. **Custom Date Range Selection**: Users can select specific time periods to analyze, with the visualization updating dynamically based on the selected range.
3. **Category Breakdown**: Below the chart, a detailed breakdown shows the exact amount spent in each category.
4. **Real-time Updates**: When new expenses are added, the dashboard updates automatically to reflect the changes.

The implementation uses server-side data processing and client-side rendering:

javascript

```
// Server-side data preparation
async function getExpensesByCategoryOnDateRange(username,
```

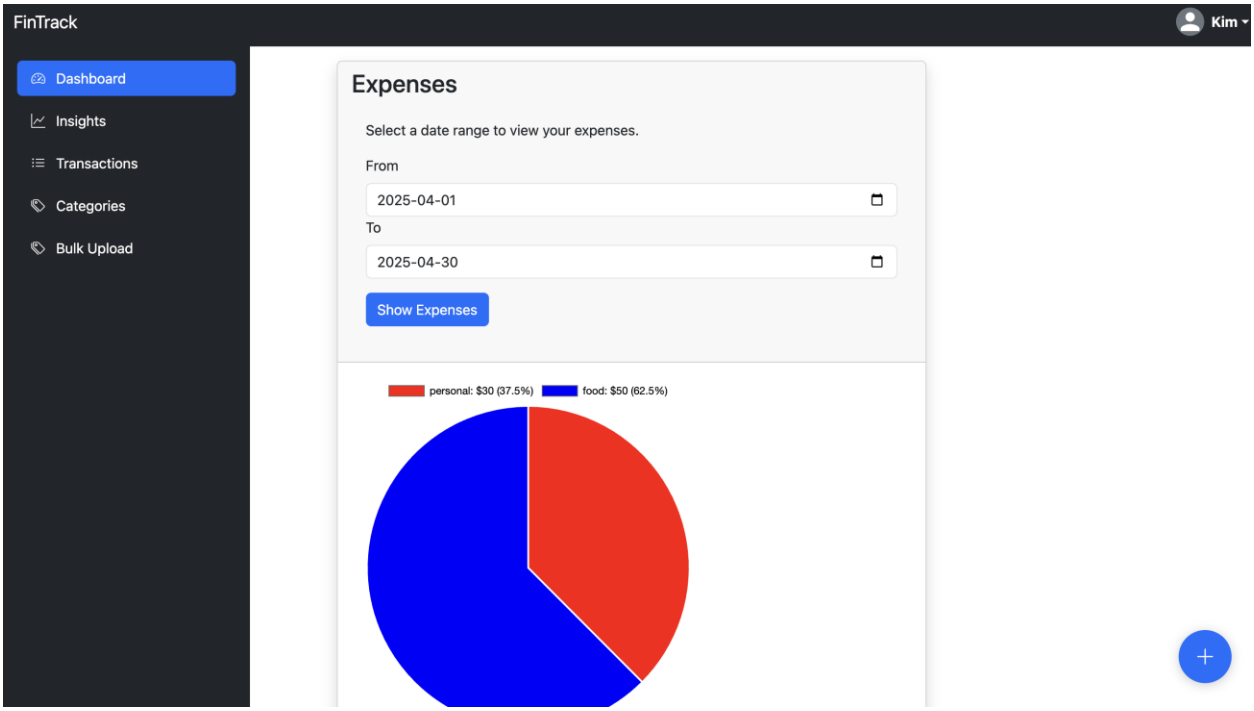
```
from,to) {
  const user = await User.findOne({ username });
  const expenses = await Expense.find({
    user: user._id,
    date: {
      $gte: from,
      $lte: to
    }
  }).sort({ date: -1 });

  const categoryTotals = {};
  expenses.forEach(({ category, amount }) => {
    categoryTotals[category] = (categoryTotals[category] || 0) +
amount;
  });

  const p_data = {
    labels: Object.keys(categoryTotals),
    values: Object.values(categoryTotals),
  };
  return p_data;
}

// Client-side chart rendering
<script>
    let all_data = <%- JSON.stringify(c_data) %>;
    const labels = all_data.labels;
    const values = all_data.values;

    const barColors = [
        "red", "blue", "green", "purple", "orange",
        "yellow", "brown", "pink", "grey", "cyan"
    ];

    new Chart("pieChart", {
```

```
        type: "pie",
        data: {
            labels: labels,
            datasets: [{
            backgroundColor: barColors,
            data: values
            }]
        },
        options: {
                plugins: {
                    legend: {
                        position: 'top',
                        labels: {
                            generateLabels: function(chart) {
                                const data = chart.data;
                                const total =
data.datasets[0].data.reduce((a, b) => a + b, 0);

                                return
data.labels.map((label,i)=> {
                                    const value =
data.datasets[0].data[i];
                                    const percentage = ((value *
100) / total).toFixed(1);

                                    return {
                                        text: `${label}:
$${value.toLocaleString()} (${percentage}%)`,
                                        fillStyle:
chart.data.datasets[0].backgroundColor[i],
                                        index: i
                                    };
                                });
                            }
                        }
```

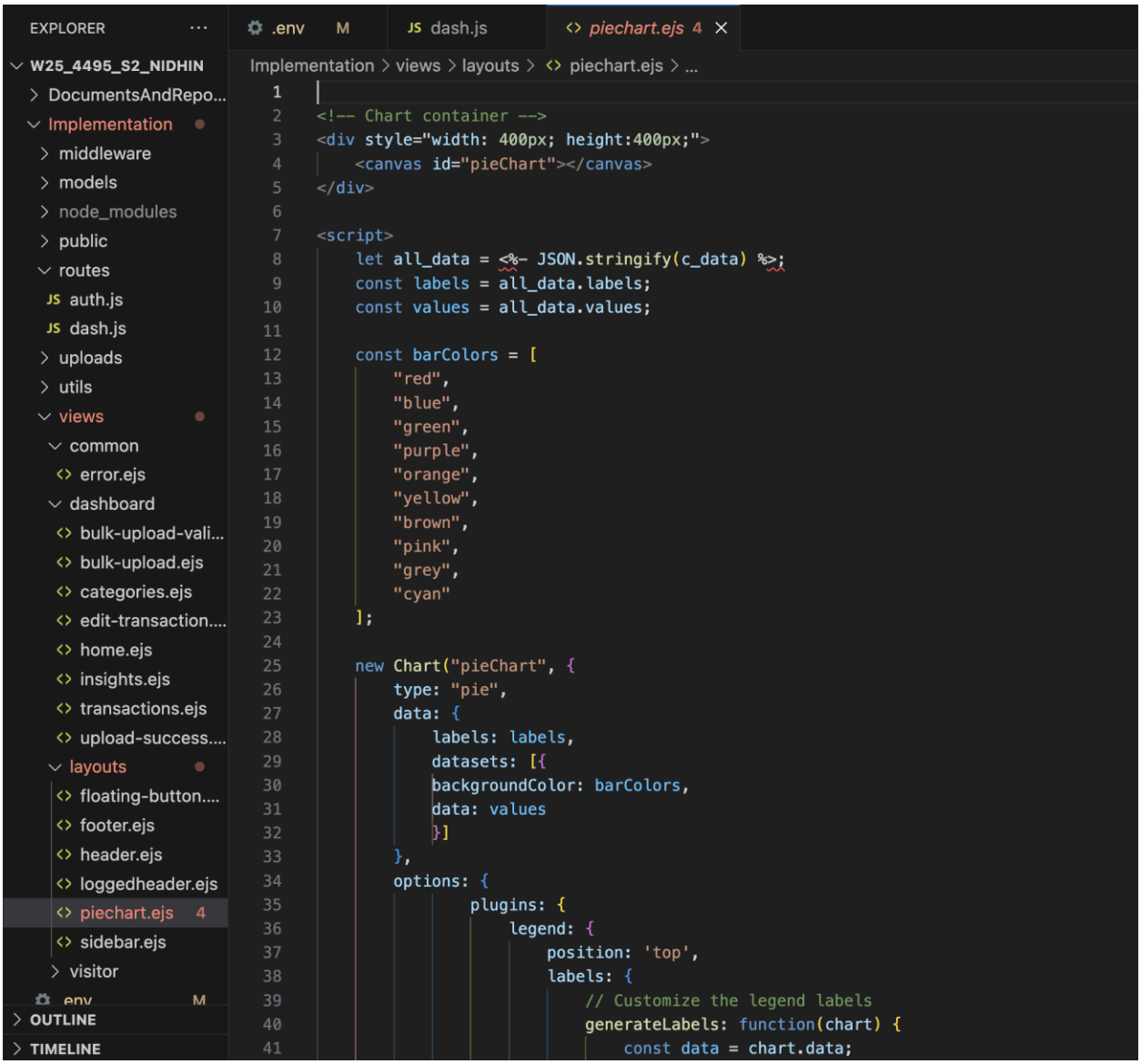
```
                    },
                    tooltip: {
                        callbacks: {
                            label: function(context) {
                                const value = context.parsed;
                                const total =
context.dataset.data.reduce((a, b) => a + b, 0);
                                const percentage = ((value * 100)
/ total).toFixed(1);
                                return `${context.label}:
$${value.toLocaleString()} (${percentage}%)`;
                            }
                        }
                    }
                }
            }

    });
</script>
```

Screenshots & Code

The dashboard interface shows a clean, visual representation of financial data:

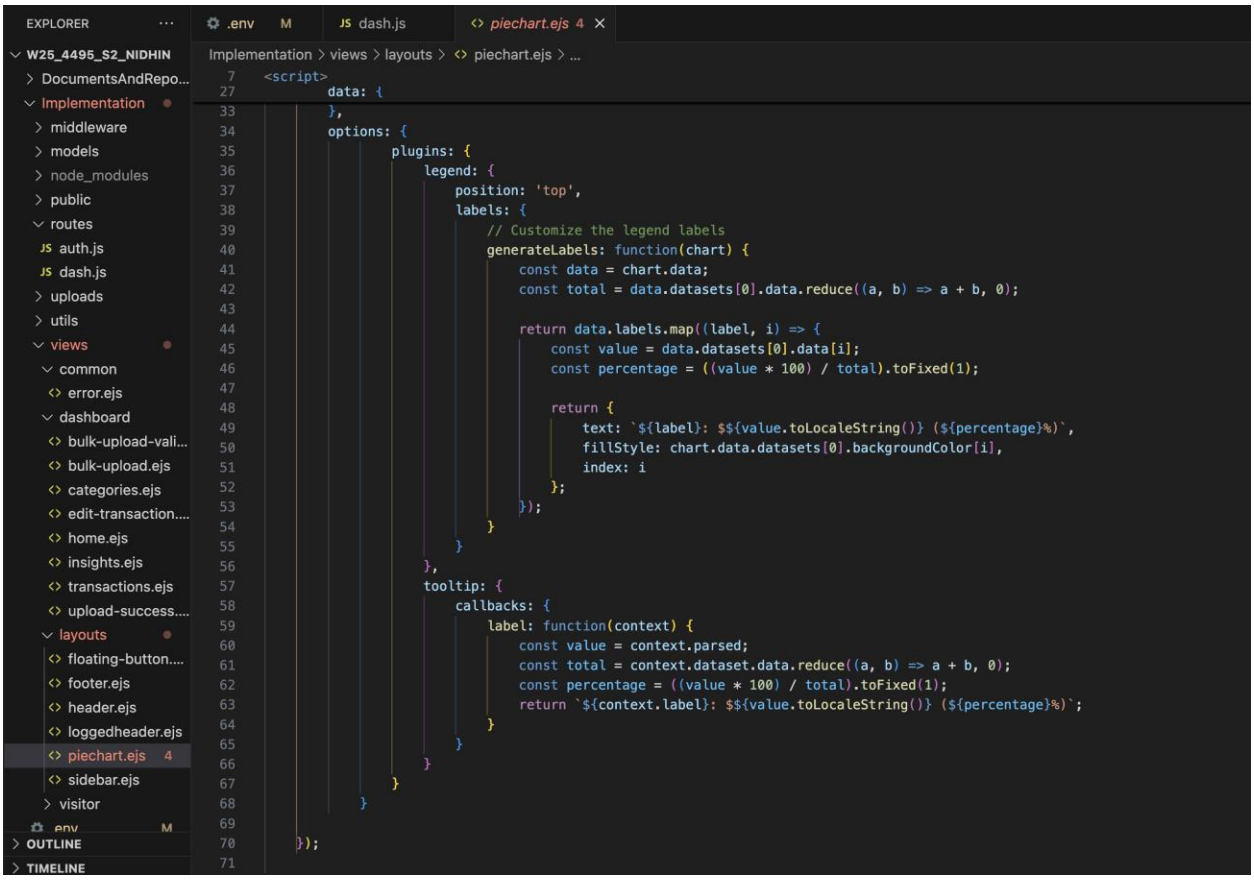


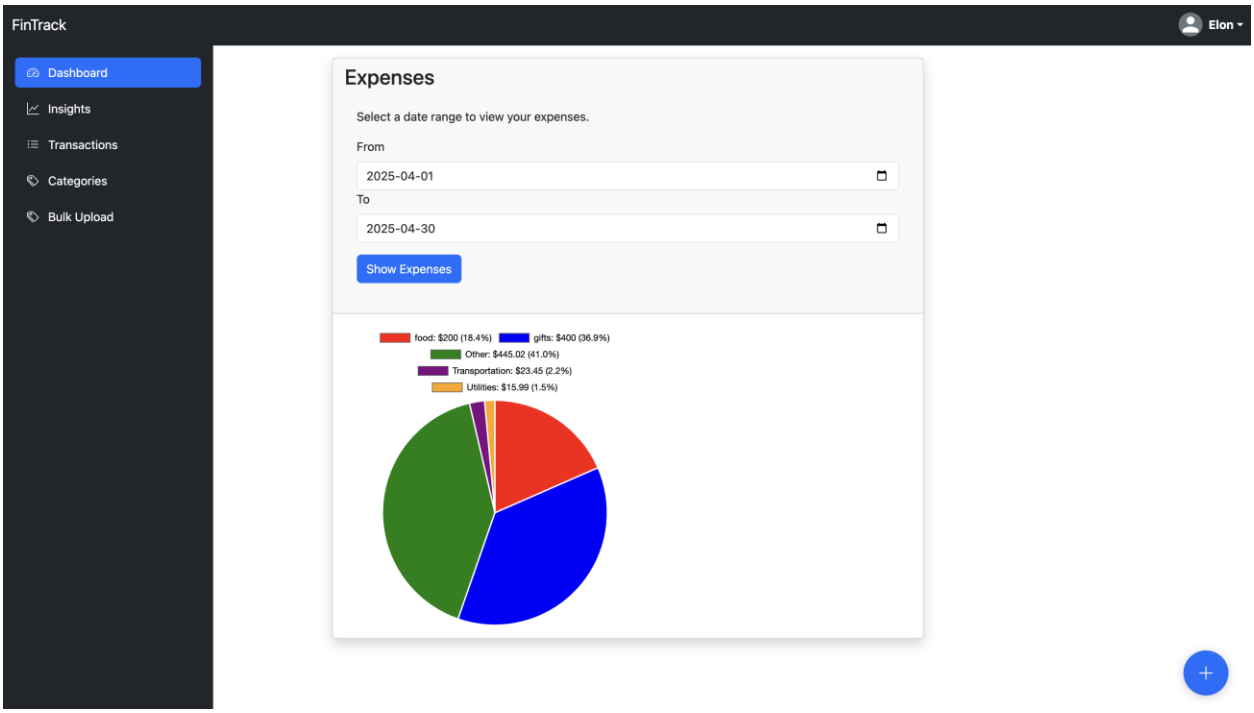
Piechart.ejs


EXPLORER
.env M
dash.js
piechart.ejs 4
Implementation > views > layouts > piechart.ejs > ...
W25_4495_S2_NIDHIN
DocumentsAndRepo...
Implementation
middleware
models
node_modules
public
routes
auth.js
dash.js
uploads
utils
views
common
error.ejs
dashboard
bulk-upload-vali...
bulk-upload.ejs
categories.ejs
edit-transaction....
home.ejs
insights.ejs
transactions.ejs
upload-success....
layouts
floating-button....
footer.ejs
header.ejs
loggedheader.ejs
piechart.ejs 4
sidebar.ejs
visitor
env M
OUTLINE
TIMELINE
<!-- Chart container -->

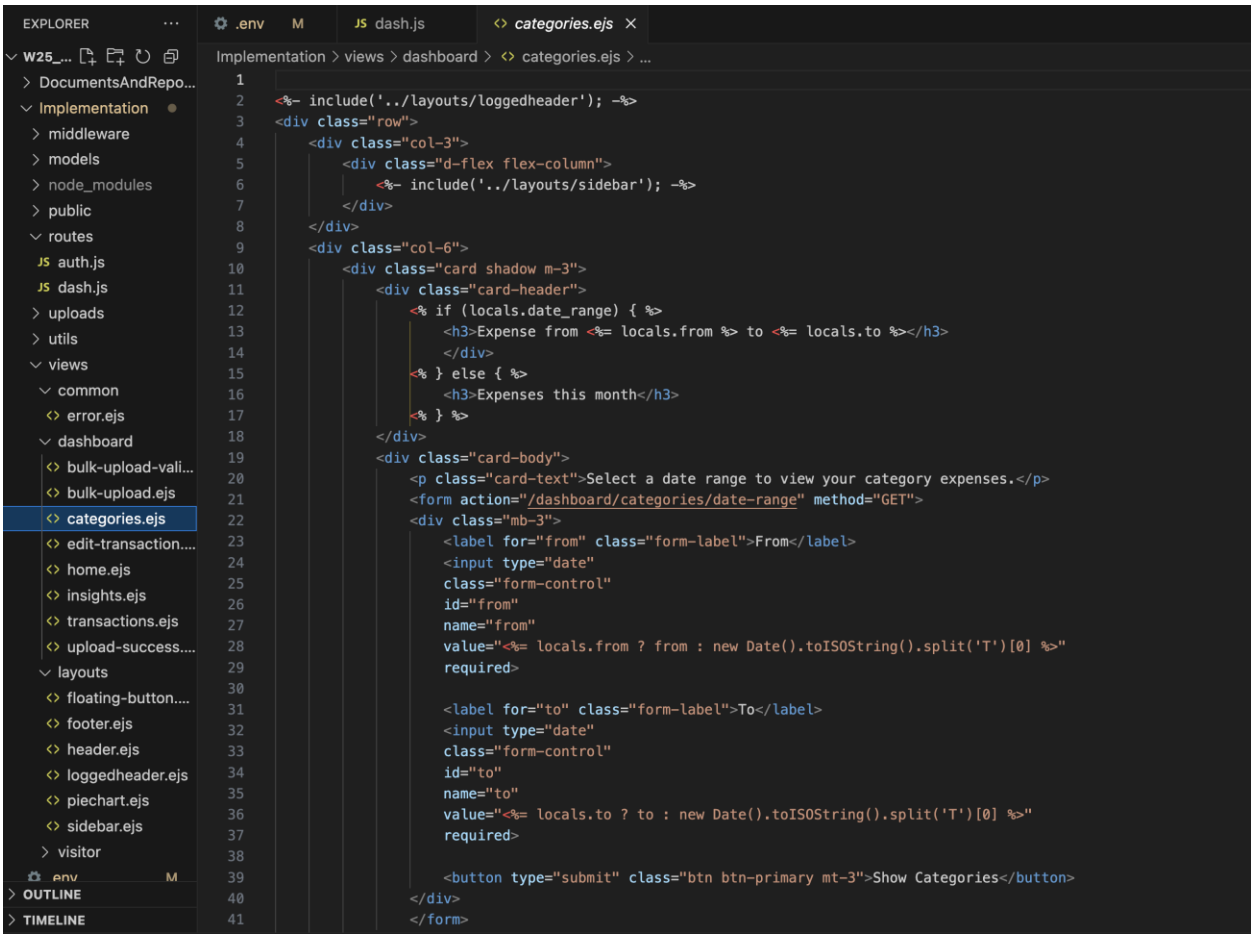
<canvas id="pieChart"></canvas>

<script>
let all_data = <%- JSON.stringify(c_data) %>;
const labels = all_data.labels;
const values = all_data.values;
const barColors = [
"red",
"blue",
"green",
"purple",
"orange",
"yellow",
"brown",
"pink",
"grey",
"cyan"
];
new Chart("pieChart", {
type: "pie",
data: {
labels: labels,
datasets: [{
backgroundColor: barColors,
data: values
}]
},
options: {
plugins: {
legend: {
position: 'top',
labels: {
// Customize the legend labels
generateLabels: function(chart) {
const data = chart.data;


EXPLORER
.env M
dash.js
piechart.ejs 4
Implementation > views > layouts > piechart.ejs > ...
W25_4495_S2_NIDHIN
DocumentsAndRepo...
Implementation
middleware
models
node_modules
public
routes
auth.js
dash.js
uploads
utils
views
common
error.ejs
dashboard
bulk-upload-vali...
bulk-upload.ejs
categories.ejs
edit-transaction....
home.ejs
insights.ejs
transactions.ejs
upload-success....
layouts
floating-button....
footer.ejs
header.ejs
loggedheader.ejs
piechart.ejs 4
sidebar.ejs
visitor
.env M
OUTLINE
TIMELINE
<script>
data: {
},
options: {
plugins: {
legend: {
position: 'top',
labels: {
// Customize the legend labels
generateLabels: function(chart) {
const data = chart.data;
const total = data.datasets[0].data.reduce((a, b) => a + b, 0);
return data.labels.map((label, i) => {
const value = data.datasets[0].data[i];
const percentage = ((value * 100) / total).toFixed(1);
return {
text: `${label}: $${value.toLocaleString()} (${percentage}%)`,
fillStyle: chart.data.datasets[0].backgroundColor[i],
index: i
};
});
}
}
},
tooltip: {
callbacks: {
label: function(context) {
const value = context.parsed;
const total = context.dataset.data.reduce((a, b) => a + b, 0);
const percentage = ((value * 100) / total).toFixed(1);
return `${context.label}: $${value.toLocaleString()} (${percentage}%)`;
}
}
}
}
}
});



Categories.ejs


EXPLORER
.env M
dash.js
categories.ejs
Implementation > views > dashboard > categories.ejs > ...
W25_...
DocumentsAndRepo...
Implementation
middleware
models
node_modules
public
routes
auth.js
dash.js
uploads
utils
views
common
error.ejs
dashboard
bulk-upload-vali...
bulk-upload.ejs
categories.ejs
edit-transaction....
home.ejs
insights.ejs
transactions.ejs
upload-success....
layouts
floating-button....
footer.ejs
header.ejs
loggedheader.ejs
piechart.ejs
sidebar.ejs
visitor
.env M
OUTLINE
TIMELINE
<%- include('../layouts/loggedheader'); -%>

<%- include('../layouts/sidebar'); -%>

<% if (locals.date_range) { %>
<h3>Expense from <%= locals.from %> to <%= locals.to %></h3>

<% } else { %>
<h3>Expenses this month</h3>
<% } %>

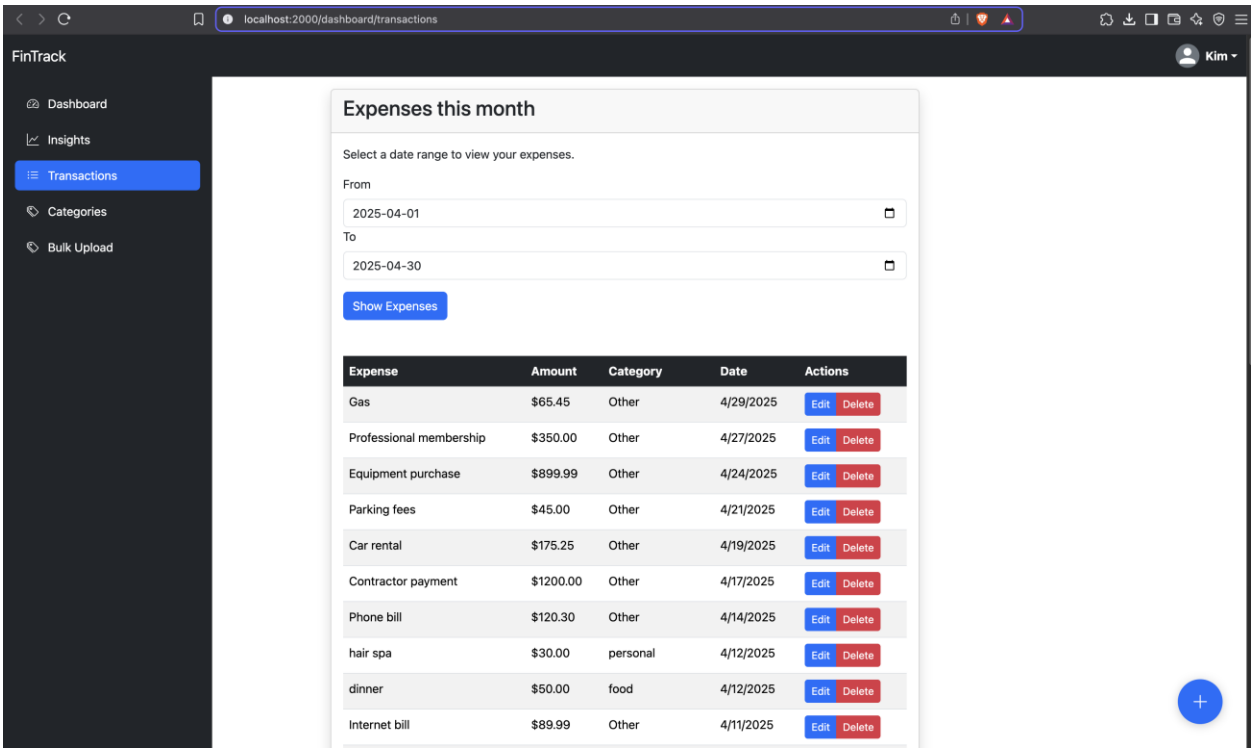
Select a date range to view your category expenses.
<form action="/dashboard/categories/date-range" method="GET">

<label for="from" class="form-label">From</label>
<input type="date"
class="form-control"
id="from"
name="from"
value="<%= locals.from ? from : new Date().toISOString().split('T')[0] %>"
required>
<label for="to" class="form-label">To</label>
<input type="date"
class="form-control"
id="to"
name="to"
value="<%= locals.to ? to : new Date().toISOString().split('T')[0] %>"
required>
<button type="submit" class="btn btn-primary mt-3">Show Categories</button>

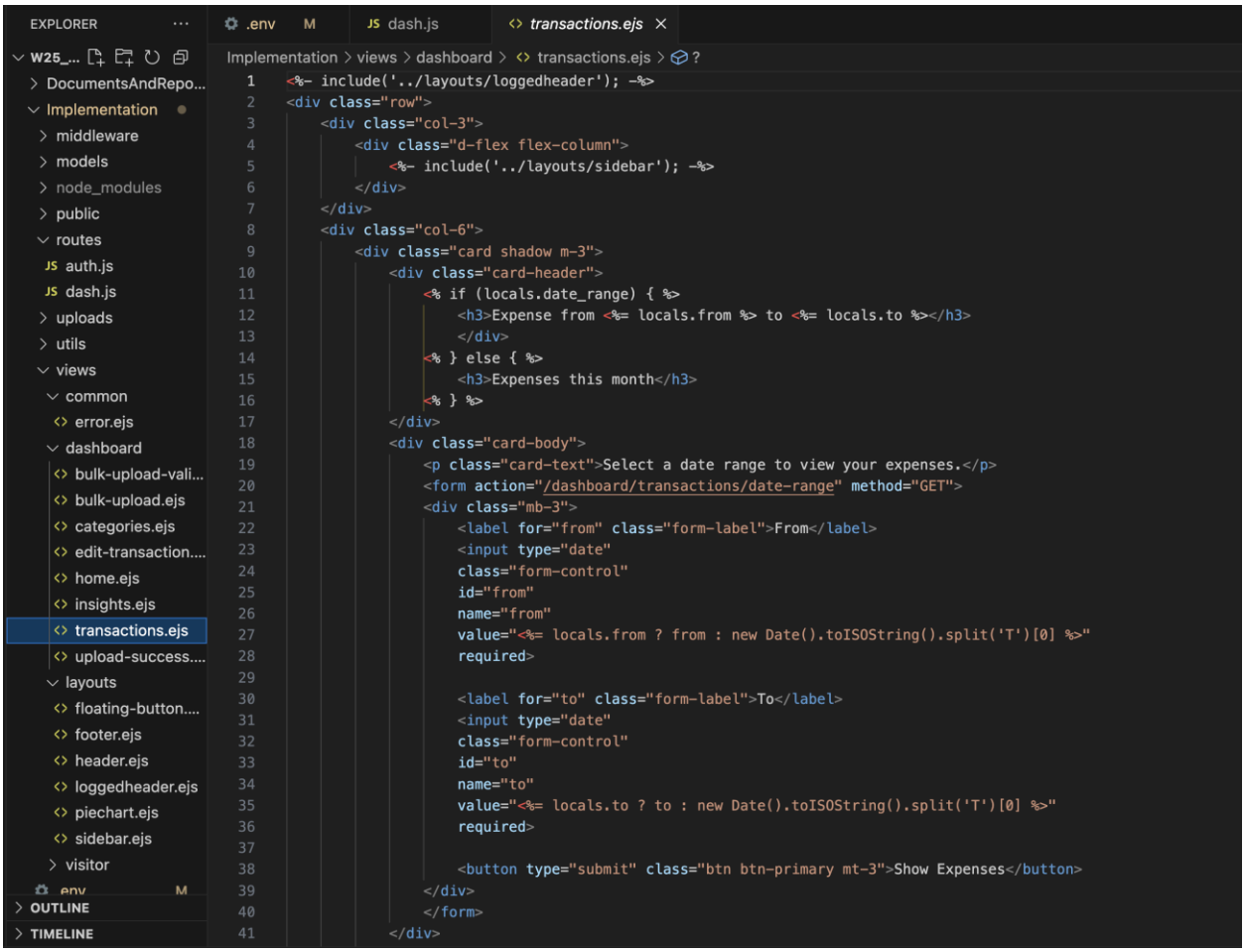
</form>

```
            <div class="card-body">
                <div class="table-responsive">
                    <% if(typeof success_msg !== 'undefined' && success_msg != ''){ %>
                        <div class="alert alert-success alert-dismissible fade show" role="alert">
                            <%= success_msg %>
                            <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
                        </div>
                    <% } %>
                    <table class="table table-striped">
                        <thead class="table-dark">
                        <tr>
                            <th scope="col">Category</th>
                            <th scope="col">Amount</th>
                        </tr>
                        </thead>
                    </table>
                </div>
                <% if (expenses.length === 0) { %>
                <div class="alert alert-info">
                    No expenses found. Add your first expense to get started.
                </div>
                <% } %>

                <div class="table-responsive">
                    <table class="table table-striped">
                        <tbody>
                        <% for (var i = 0; i < expenses.labels.length; i++) { %>
                            <tr>
                            <td><%= expenses.labels[i]%></td>
                            <td>$<%= expenses.values[i].toFixed(2) %></td>
                            </tr>
                        <% } %>
                        </tbody>
                    </table>
                </div>
            </div>
        </div>
    </div>
</div>
```

```
                        <thead class="table-dark">
                        <tr>
                            <th scope="col">Category</th>
                            <th scope="col">Amount</th>
                        </tr>
                        </thead>
                    </table>
                </div>
                <% if (expenses.length === 0) { %>
                <div class="alert alert-info">
                    No expenses found. Add your first expense to get started.
                </div>
                <% } %>

                <div class="table-responsive">
                    <table class="table table-striped">
                        <tbody>
                        <% for (var i = 0; i < expenses.labels.length; i++) { %>
                            <tr>
                            <td><%= expenses.labels[i]%></td>
                            <td>$<%= expenses.values[i].toFixed(2) %></td>
                            </tr>
                        <% } %>
                        </tbody>
                    </table>
                </div>
            </div>
        </div>
    </div>
</div>
<%- include('../layouts/floating-button') %>
<%- include('../layouts/footer'); -%>
```



Transaction.ejs


EXPLORER
.env M
dash.js
transactions.ejs
Implementation > views > dashboard > transactions.ejs > ?
W25_...
DocumentsAndRepo...
Implementation
middleware
models
node_modules
public
routes
auth.js
dash.js
uploads
utils
views
common
error.ejs
dashboard
bulk-upload-vali...
bulk-upload.ejs
categories.ejs
edit-transaction....
home.ejs
insights.ejs
transactions.ejs
upload-success....
layouts
floating-button....
footer.ejs
header.ejs
loggedheader.ejs
piechart.ejs
sidebar.ejs
visitor
env M
OUTLINE
TIMELINE
<%- include('../layouts/loggedheader'); -%>

<%- include('../layouts/sidebar'); -%>

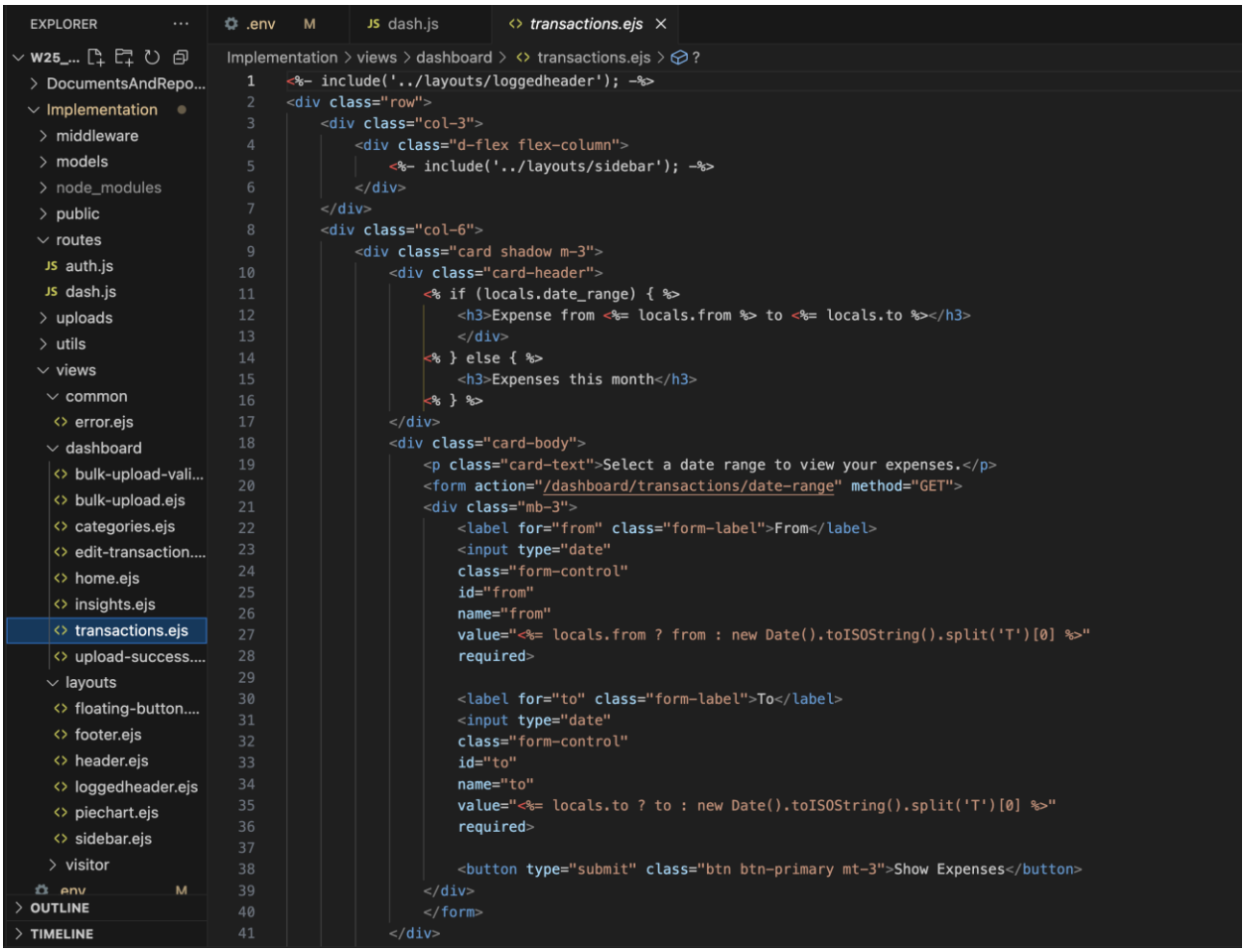
<% if (locals.date_range) { %>
<h3>Expense from <%= locals.from %> to <%= locals.to %></h3>

<% } else { %>
<h3>Expenses this month</h3>
<% } %>

Select a date range to view your expenses.
<form action="/dashboard/transactions/date-range" method="GET">

<label for="from" class="form-label">From</label>
<input type="date"
class="form-control"
id="from"
name="from"
value="<%= locals.from ? from : new Date().toISOString().split('T')[0] %>"
required>
<label for="to" class="form-label">To</label>
<input type="date"
class="form-control"
id="to"
name="to"
value="<%= locals.to ? to : new Date().toISOString().split('T')[0] %>"
required>
<button type="submit" class="btn btn-primary mt-3">Show Expenses</button>

</form>


EXPLORER
.env M
dash.js
transactions.ejs
Implementation > views > dashboard > transactions.ejs > ?
W25_...
DocumentsAndRepo...
Implementation
middleware
models
node_modules
public
routes
auth.js
dash.js
uploads
utils
views
common
error.ejs
dashboard
bulk-upload-vali...
bulk-upload.ejs
categories.ejs
edit-transaction....
home.ejs
insights.ejs
transactions.ejs
upload-success....
layouts
floating-button....
footer.ejs
header.ejs
loggedheader.ejs
piechart.ejs
sidebar.ejs
visitor
env M
OUTLINE
TIMELINE
<%- include('../layouts/loggedheader'); -%>

<%- include('../layouts/sidebar'); -%>

<% if (locals.date_range) { %>
<h3>Expense from <%= locals.from %> to <%= locals.to %></h3>

<% } else { %>
<h3>Expenses this month</h3>
<% } %>

Select a date range to view your expenses.
<form action="/dashboard/transactions/date-range" method="GET">

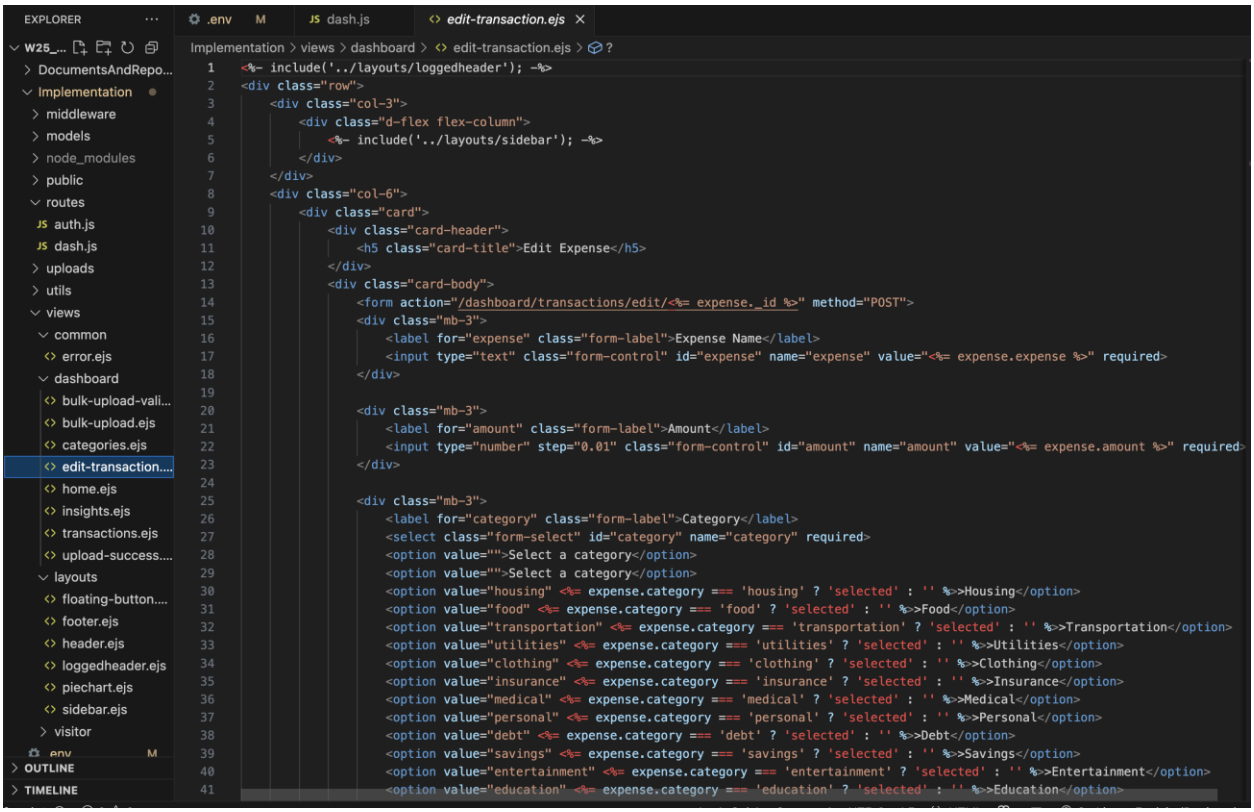
<label for="from" class="form-label">From</label>
<input type="date"
class="form-control"
id="from"
name="from"
value="<%= locals.from ? from : new Date().toISOString().split('T')[0] %>"
required>
<label for="to" class="form-label">To</label>
<input type="date"
class="form-control"
id="to"
name="to"
value="<%= locals.to ? to : new Date().toISOString().split('T')[0] %>"
required>
<button type="submit" class="btn btn-primary mt-3">Show Expenses</button>

</form>

```
            </div>
            <div class="card-body">
                <div class="table-responsive">
                    <% if(typeof success_msg !== 'undefined' && success_msg != ''){ %>
                        <div class="alert alert-success alert-dismissible fade show" role="alert">
                            <%= success_msg %>
                            <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
                        </div>
                    <% } %>
                    <table class="table table-striped">
                        <thead class="table-dark">
                        <tr>
                            <th scope="col">Expense</th>
                            <th scope="col">Amount</th>
                            <th scope="col">Category</th>
                            <th scope="col">Date</th>
                            <th scope="col">Actions</th>
                        </tr>
                        </thead>
                        <tbody>
                        <% expenses.forEach(expense => { %>
                            <tr>
                            <td><%= expense.expense %></td>
                            <td>$<%= expense.amount.toFixed(2) %></td>
                            <td><%= expense.category %></td>
                            <td><%= new Date(expense.date).toLocaleDateString() %></td>
                            <td>
                                <div class="btn-group" role="group">
                                <a href="/dashboard/transactions/edit/<%= expense._id %>" class="btn btn-sm btn-primary">Edit</a>
                                <a href="/dashboard/transactions/delete/<%= expense._id %>" class="btn btn-sm btn-danger">Delete</a>
                                </div>
                            </td>
                            </tr>
                        <% }); %>
                        </tbody>
                    </table>
                </div>
                <% if (expenses.length === 0) { %>
```

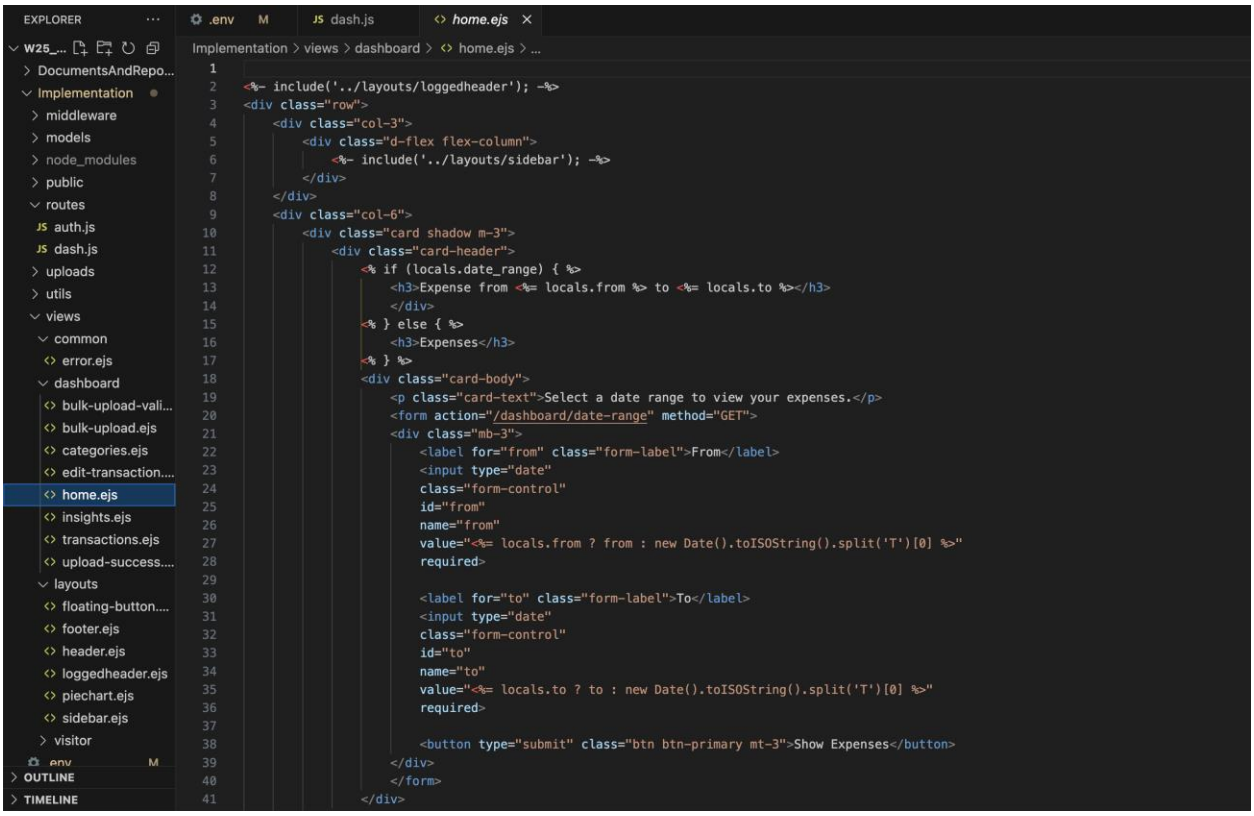
```
                                <th scope="col">Category</th>
                                <th scope="col">Date</th>
                                <th scope="col">Actions</th>
                            </tr>
                            </thead>
                            <tbody>
                            <% expenses.forEach(expense => { %>
                                <tr>
                                <td><%= expense.expense %></td>
                                <td>$<%= expense.amount.toFixed(2) %></td>
                                <td><%= expense.category %></td>
                                <td><%= new Date(expense.date).toLocaleDateString() %></td>
                                <td>
                                    <div class="btn-group" role="group">
                                    <a href="/dashboard/transactions/edit/<%= expense._id %>" class="btn btn-sm btn-primary">Edit</a>
                                    <a href="/dashboard/transactions/delete/<%= expense._id %>" class="btn btn-sm btn-danger">Delete</a>
                                    </div>
                                </td>
                                </tr>
                            <% }); %>
                            </tbody>
                        </table>
                    </div>
                    <% if (expenses.length === 0) { %>
                    <div class="alert alert-info">
                        No expenses found. Add your first expense to get started.
                    </div>
                    <% } %>
                </div>
            </div>
        </div>
    </div>
    <%- include('../layouts/floating-button') %>
    <%- include('../layouts/footer'); -%>
```

Edit-transaction.ejs




```
                    <option value="clothing" <%= expense.category === 'clothing' ? 'selected' : '' %>>Clothing</option>
                    <option value="insurance" <%= expense.category === 'insurance' ? 'selected' : '' %>>Insurance</option>
                    <option value="medical" <%= expense.category === 'medical' ? 'selected' : '' %>>Medical</option>
                    <option value="personal" <%= expense.category === 'personal' ? 'selected' : '' %>>Personal</option>
                    <option value="debt" <%= expense.category === 'debt' ? 'selected' : '' %>>Debt</option>
                    <option value="savings" <%= expense.category === 'savings' ? 'selected' : '' %>>Savings</option>
                    <option value="entertainment" <%= expense.category === 'entertainment' ? 'selected' : '' %>>Entertainment</option>
                    <option value="education" <%= expense.category === 'education' ? 'selected' : '' %>>Education</option>
                    <option value="gifts" <%= expense.category === 'gifts' ? 'selected' : '' %>>Gifts</option>
                    <option value="donations" <%= expense.category === 'donations' ? 'selected' : '' %>>Donations</option>
                    <option value="investments" <%= expense.category === 'investments' ? 'selected' : '' %>>Investments</option>
                    <option value="others" <%= expense.category === 'others' ? 'selected' : '' %>>Others</option>
                    </select>
                </div>

                <div class="mb-3">
                    <label for="date" class="form-label">Date</label>
                    <input type="date" class="form-control" id="date" name="date" value="<%= new Date(expense.date).toISOString().split('T'
                </div>

                <div class="d-flex justify-content-between">
                    <button type="submit" class="btn btn-primary">Update Expense</button>
                    <a href="/dashboard/transactions" class="btn btn-secondary">Cancel</a>
                </div>
                </form>
            </div>
        </div>
    </div>
</div>
<%- include('../layouts/floating-button') %>
<%- include('../layouts/footer'); -%>
```

Home.ejs


EXPLORER
.env M
dash.js
home.ejs
Implementation > views > dashboard > home.ejs > ...
W25_...
DocumentsAndRepo...
Implementation
middleware
models
node_modules
public
routes
auth.js
dash.js
uploads
utils
views
common
error.ejs
dashboard
bulk-upload-vali...
bulk-upload.ejs
categories.ejs
edit-transaction....
home.ejs
insights.ejs
transactions.ejs
upload-success....
layouts
floating-button....
footer.ejs
header.ejs
loggedheader.ejs
piechart.ejs
sidebar.ejs
visitor
.env M
OUTLINE
TIMELINE
<%- include('../layouts/loggedheader'); -%>

<%- include('../layouts/sidebar'); -%>

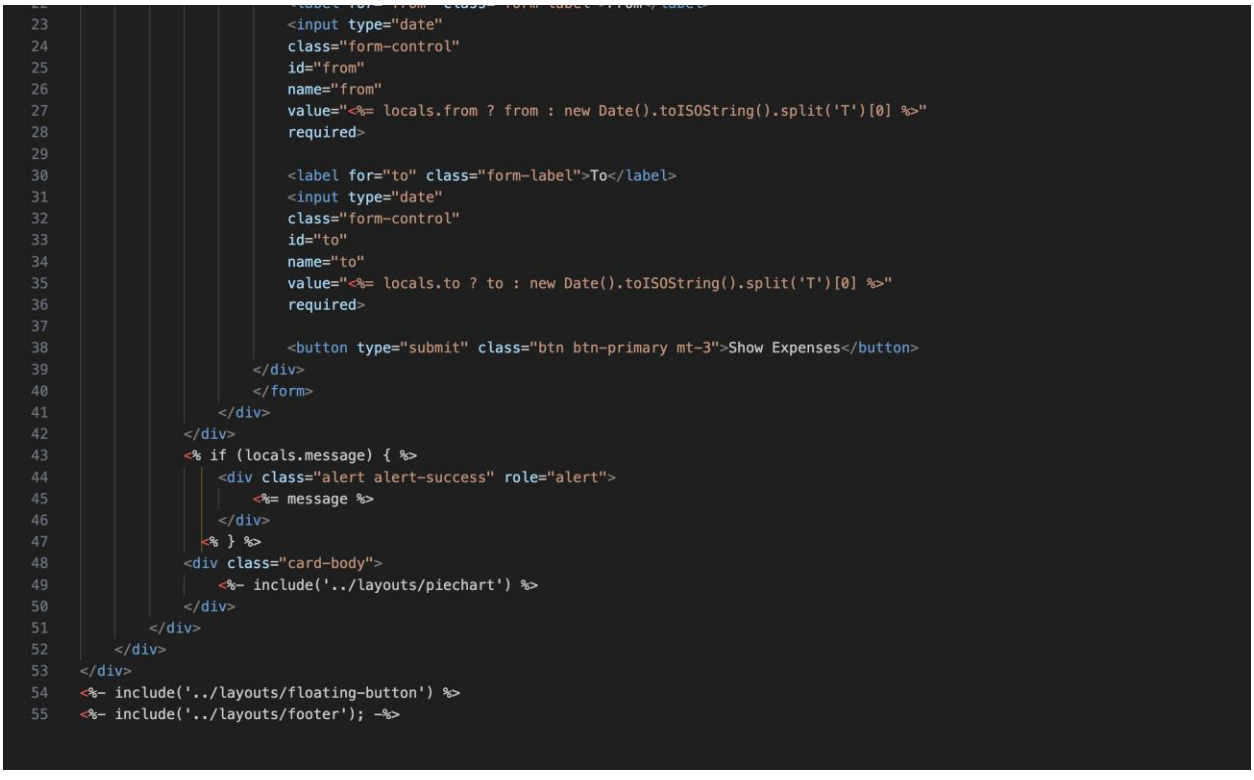
<% if (locals.date_range) { %>
<h3>Expense from <%= locals.from %> to <%= locals.to %></h3>

<% } else { %>
<h3>Expenses</h3>
<% } %>

Select a date range to view your expenses.
<form action="/dashboard/date-range" method="GET">

<label for="from" class="form-label">From</label>
<input type="date"
class="form-control"
id="from"
name="from"
value="<%= locals.from ? from : new Date().toISOString().split('T')[0] %>"
required>
<label for="to" class="form-label">To</label>
<input type="date"
class="form-control"
id="to"
name="to"
value="<%= locals.to ? to : new Date().toISOString().split('T')[0] %>"
required>
<button type="submit" class="btn btn-primary mt-3">Show Expenses</button>

</form>


<input type="date"
class="form-control"
id="from"
name="from"
value="<%= locals.from ? from : new Date().toISOString().split('T')[0] %>"
required>
<label for="to" class="form-label">To</label>
<input type="date"
class="form-control"
id="to"
name="to"
value="<%= locals.to ? to : new Date().toISOString().split('T')[0] %>"
required>
<button type="submit" class="btn btn-primary mt-3">Show Expenses</button>

</form>

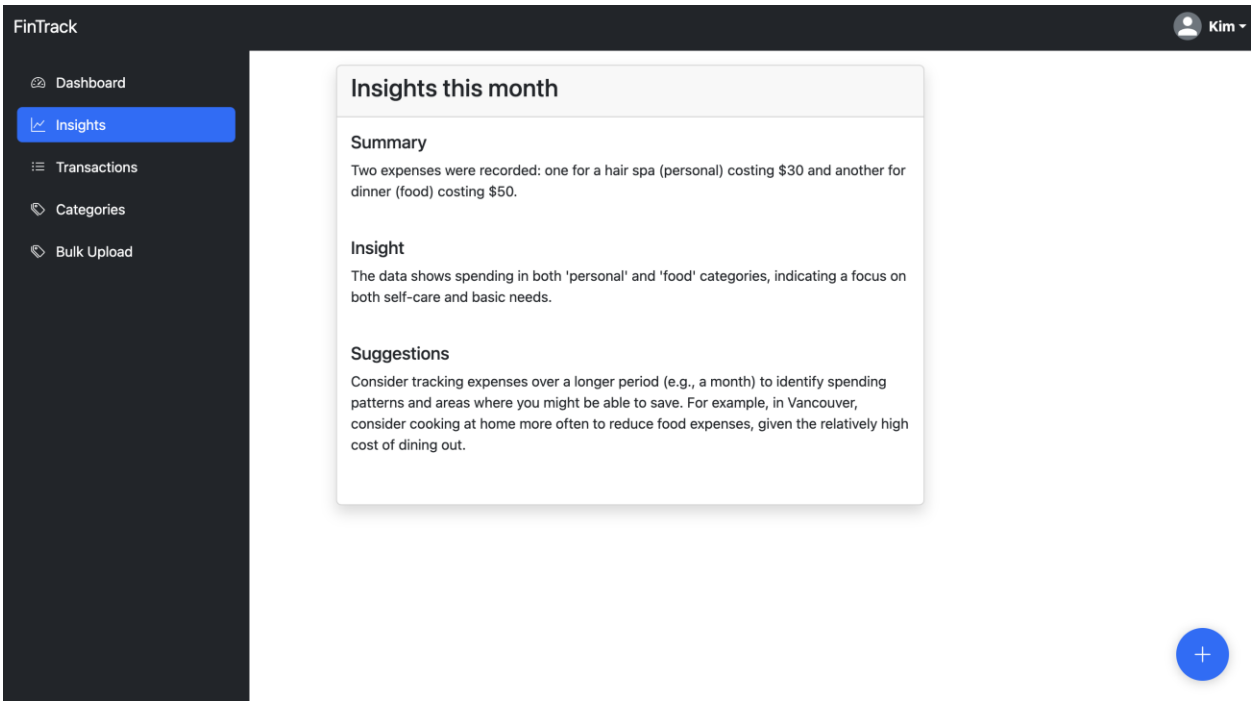
<% if (locals.message) { %>

<%= message %>

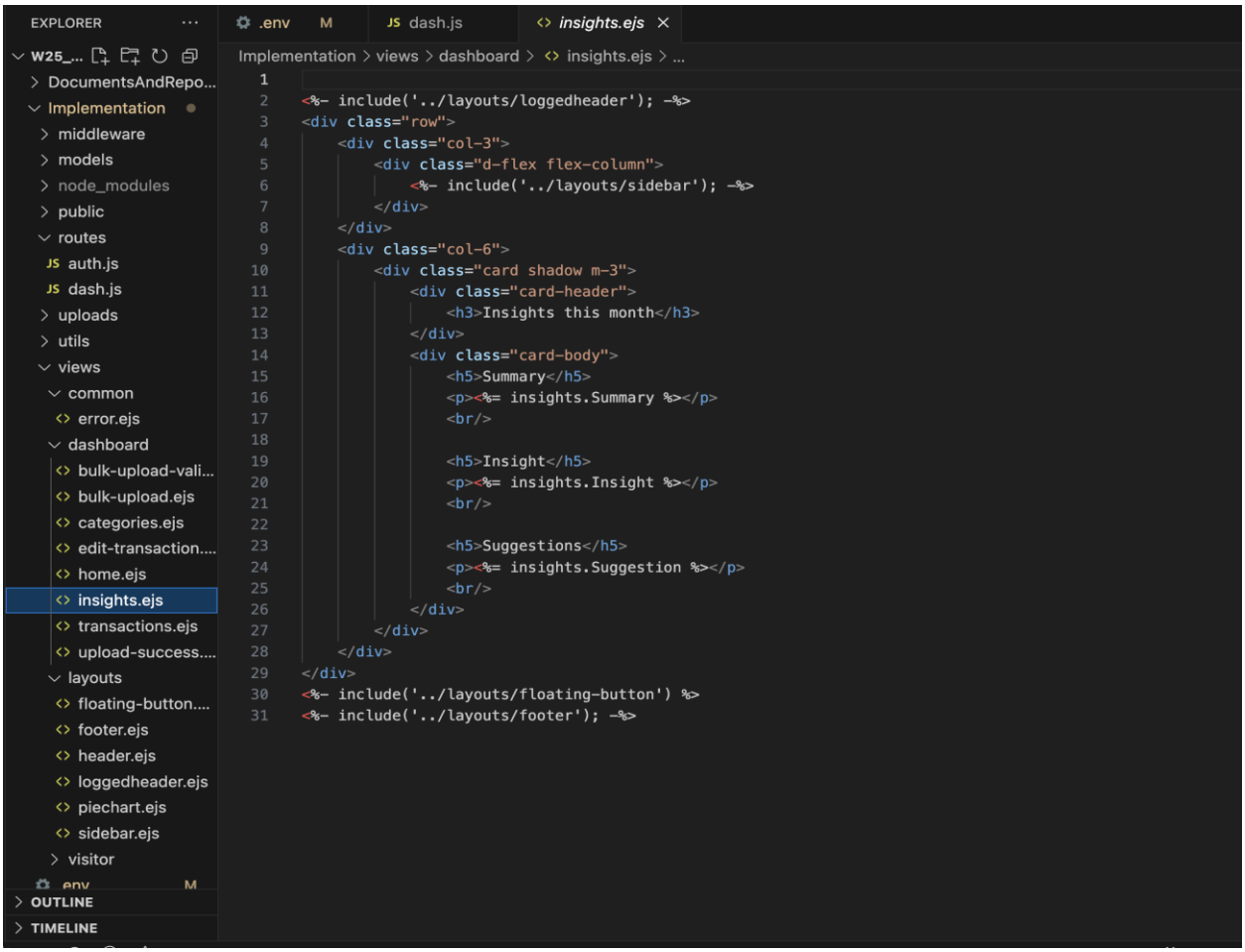
<% } %>

<%- include('../layouts/piechart') %>

<%- include('../layouts/floating-button') %>
<%- include('../layouts/footer'); -%>



insights.ejs


EXPLORER
.env M
dash.js
insights.ejs
Implementation > views > dashboard > insights.ejs > ...
W25_...
DocumentsAndRepo...
Implementation
middleware
models
node_modules
public
routes
auth.js
dash.js
uploads
utils
views
common
error.ejs
dashboard
bulk-upload-vali...
bulk-upload.ejs
categories.ejs
edit-transaction....
home.ejs
insights.ejs
transactions.ejs
upload-success....
layouts
floating-button....
footer.ejs
header.ejs
loggedheader.ejs
piechart.ejs
sidebar.ejs
visitor
.env M
OUTLINE
TIMELINE
<%- include('../layouts/loggedheader'); -%>

<%- include('../layouts/sidebar'); -%>

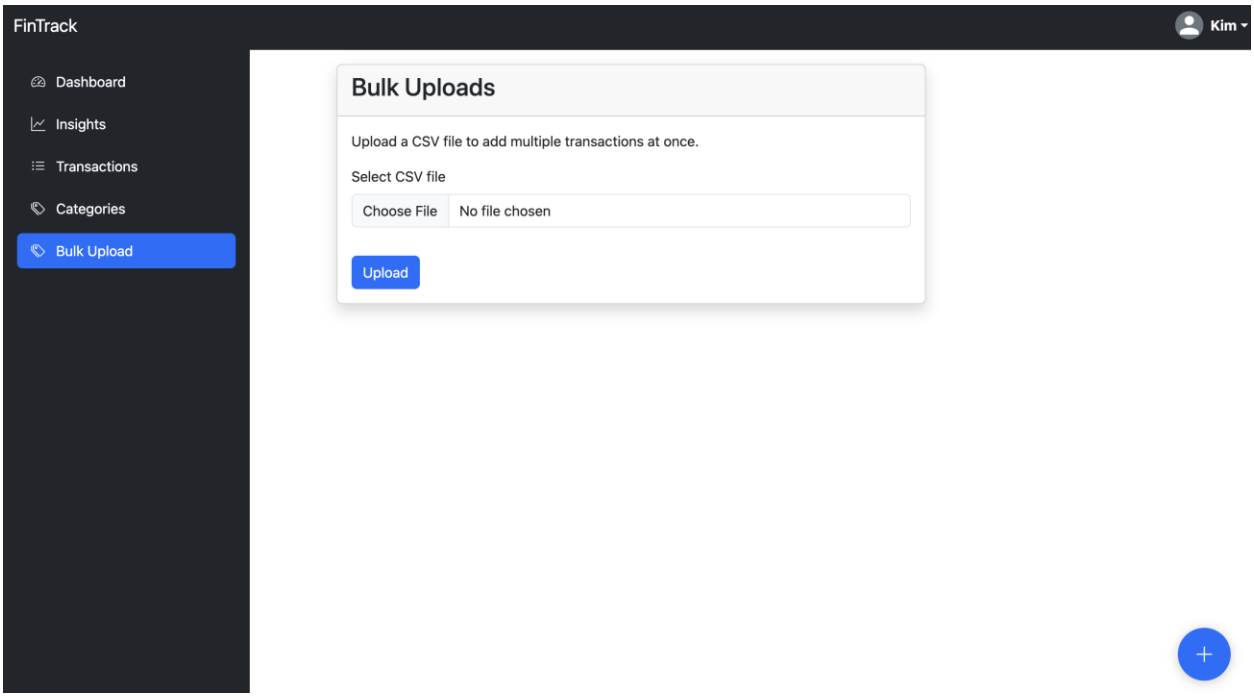
<h3>Insights this month</h3>

<h5>Summary</h5>
<%= insights.Summary %>

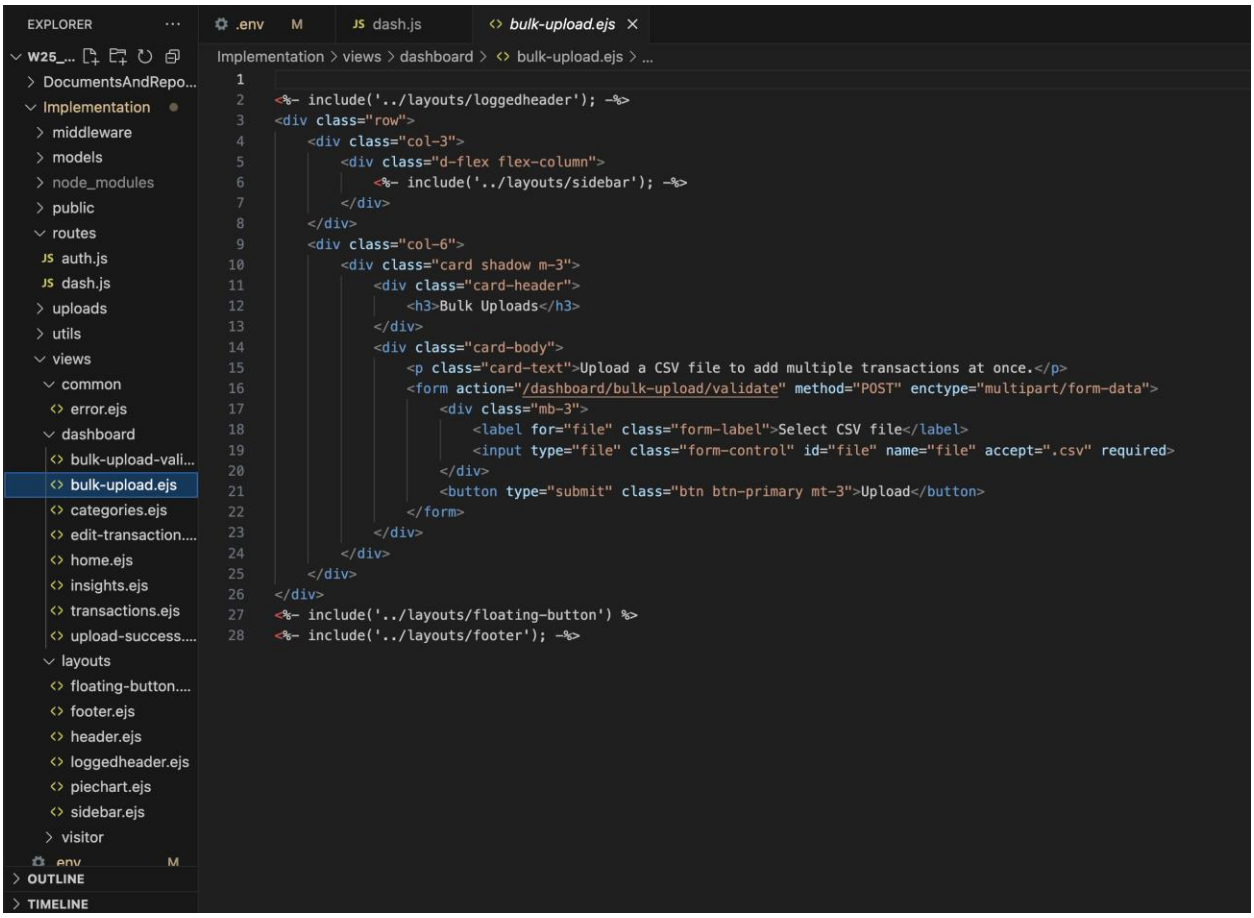
<h5>Insight</h5>
<%= insights.Insight %>

<h5>Suggestions</h5>
<%= insights.Suggestion %>

<%- include('../layouts/floating-button') %>
<%- include('../layouts/footer'); -%>



Bulk-upload.ejs


EXPLORER
.env M
dash.js
bulk-upload.ejs
Implementation > views > dashboard > bulk-upload.ejs > ...
W25_...
DocumentsAndRepo...
Implementation
middleware
models
node_modules
public
routes
auth.js
dash.js
uploads
utils
views
common
error.ejs
dashboard
bulk-upload-vali...
bulk-upload.ejs
categories.ejs
edit-transaction....
home.ejs
insights.ejs
transactions.ejs
upload-success....
layouts
floating-button....
footer.ejs
header.ejs
loggedheader.ejs
piechart.ejs
sidebar.ejs
visitor
env M
OUTLINE
TIMELINE
<%- include('../layouts/loggedheader'); -%>

<%- include('../layouts/sidebar'); -%>

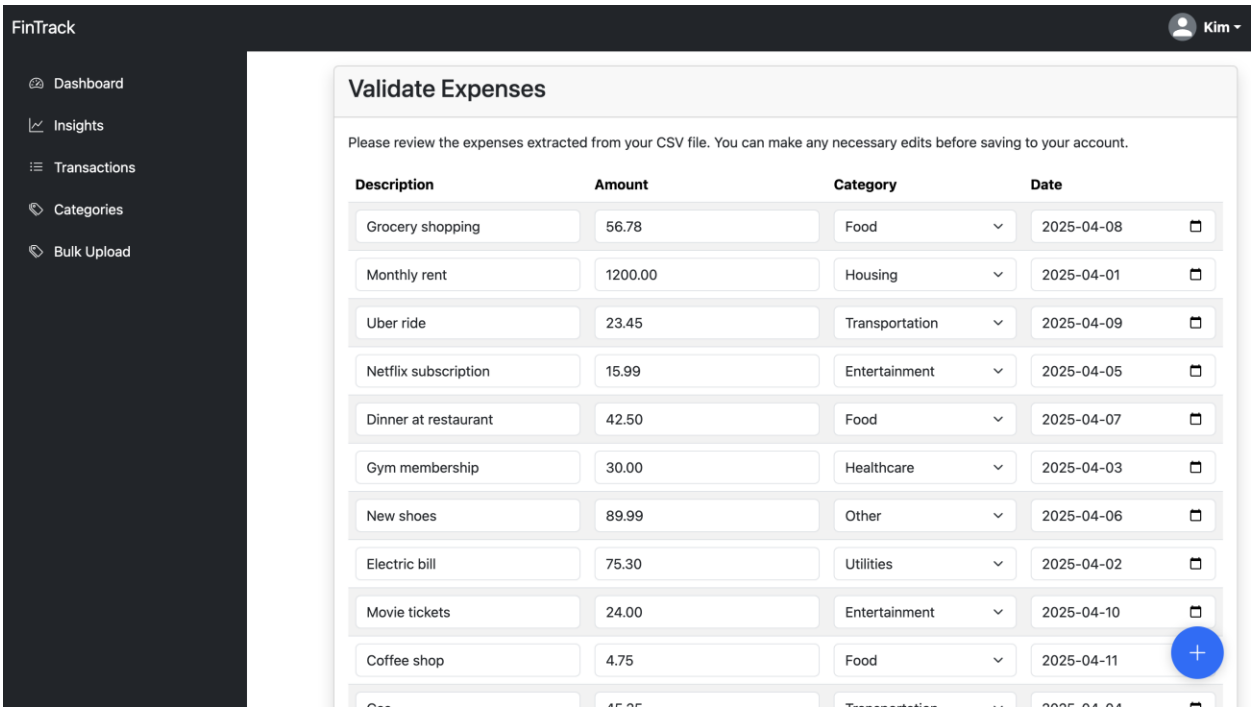
<h3>Bulk Uploads</h3>

Upload a CSV file to add multiple transactions at once.
<form action="/dashboard/bulk-upload/validate" method="POST" enctype="multipart/form-data">

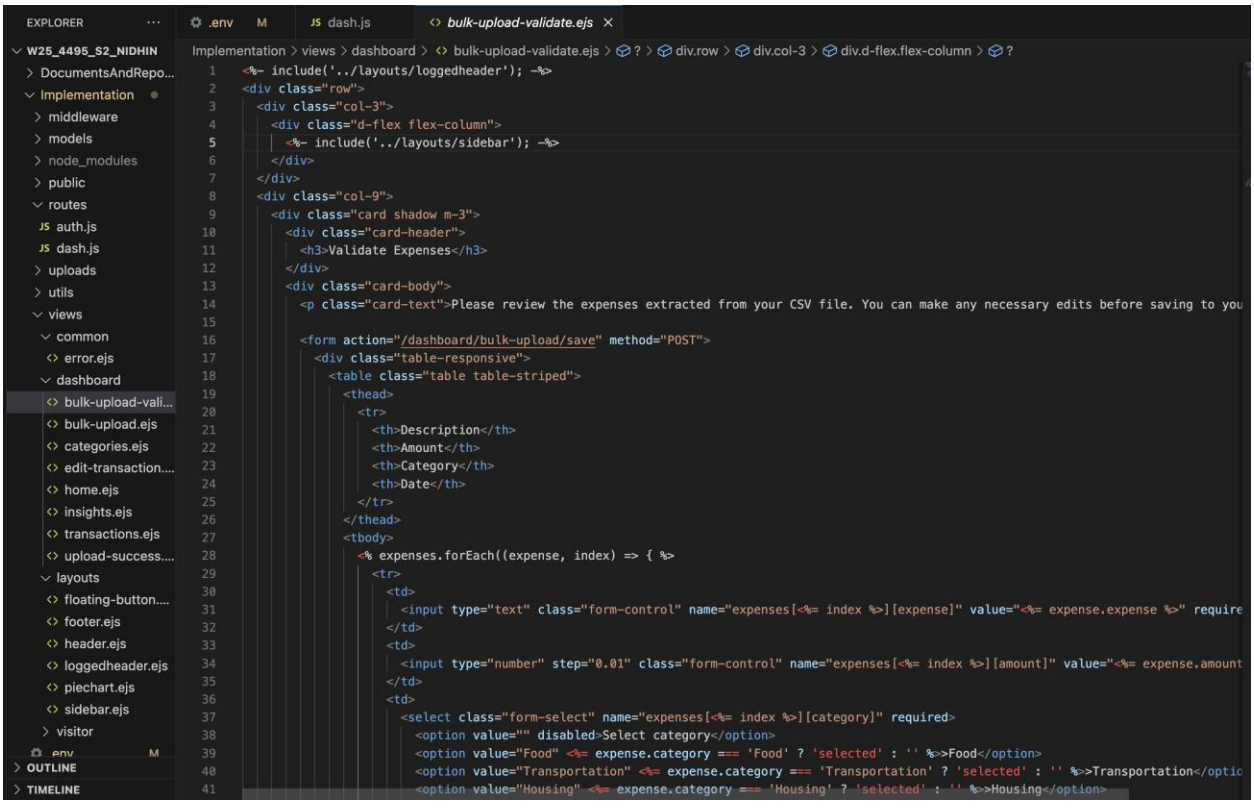
<label for="file" class="form-label">Select CSV file</label>
<input type="file" class="form-control" id="file" name="file" accept=".csv" required>

<button type="submit" class="btn btn-primary mt-3">Upload</button>
</form>

<%- include('../layouts/floating-button') %>
<%- include('../layouts/footer'); -%>

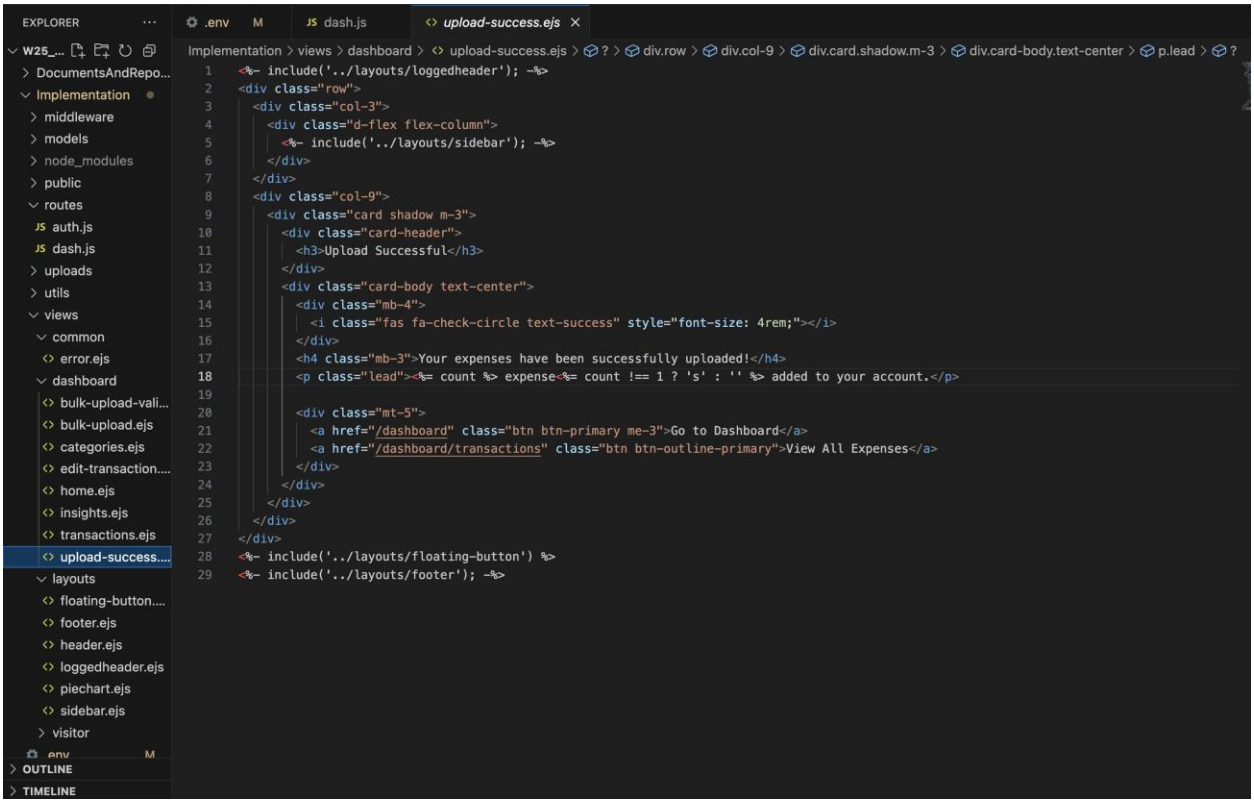


Bulk-upload-validate.ejs



```
                    <option value="Other" <%= (expense.category !== 'Food' &&
                                               expense.category !== 'Transportation' &&
                                               expense.category !== 'Housing' &&
                                               expense.category !== 'Entertainment' &&
                                               expense.category !== 'Healthcare' &&
                                               expense.category !== 'Shopping' &&
                                               expense.category !== 'Utilities') ? 'selected' : '' %>>Other</option>
                  </select>
                </td>
                <td>
                  <input type="date" class="form-control" name="expenses[<%= index %>][date]"
                    value="<%= new Date(expense.date).toISOString().split('T')[0] %>" required>
                </td>
              </tr>
            <% }); %>
          </tbody>
        </table>
      </div>

      <div class="d-flex justify-content-between mt-4">
        <a href="/dashboard/bulk-upload" class="btn btn-secondary">Cancel</a>
        <button type="submit" class="btn btn-primary">Save All Expenses</button>
      </div>
    </form>
  </div>
 </div>
</div>
<%- include('../layouts/floating-button') %>
<%- include('../layouts/footer'); -%>
```

Upload-success.ejs



Feature 3: Transaction Management System

The Transaction Management System provides users with complete control over their financial data, allowing them to view, edit, and delete expenses with a user-friendly interface. This system ensures that users can maintain accurate financial records and make corrections as needed.

Implementation Details

The transaction management system includes several key components:

1. **Transaction Listing**: A paginated, filterable list of all expenses with details like date, amount, category, and description.
2. **Edit Functionality**: Users can edit any aspect of a transaction, including its description, amount, category, and date.
3. **Delete Capability**: Allows users to remove erroneous or unwanted transactions.
4. **Date Filtering**: Users can filter transactions by date range to focus on specific time periods.

The implementation uses Express.js routes for server-side processing and EJS templates for dynamic rendering:

javascript

```
// Route for displaying transactions
router.get("/transactions", verifyToken, async (req, res) => {
  try {
    // Calculate first and last day of current month
    const today = new Date();
    const firstDay = new Date(today.getFullYear(),
today.getMonth(), 1);
    const lastDay = new Date(today.getFullYear(),
today.getMonth() + 1, 0);

    const formatDate = (date) => {
      const year = date.getFullYear();
      const month = String(date.getMonth() + 1).padStart(2, '0');
      const day = String(date.getDate()).padStart(2, '0');
      return `${year}-${month}-${day}`;
    };
```

```
    const defaultFrom = formatDate(firstDay);
    const defaultTo = formatDate(lastDay);

    const expenses = await getExpensesonDateRange(req.username,
defaultFrom, defaultTo);
    res.render("dashboard/transactions.ejs", {
      path: "/dashboard/transactions",
      firstname: req.firstname,
      expenses: expenses,
      success_msg: req.query.success_msg || "",
      from: defaultFrom,
      to: defaultTo,
    });
  } catch (error) {
    console.error("Error:", error);
    res.render("common/error.ejs", {
      message: "Failed to fetch expenses. Please try again.",
    });
  }
});

// Route for editing transactions
router.post("/transactions/edit/:id", verifyToken, async (req,
res) => {
  try {
    const expenseId = req.params.id;
    const { expense, amount, category, date } = req.body;

    // Validate input
    if (!expense || !amount || !category) {
      return res.render("common/error.ejs", {
        message: "All fields are required.",
      });
    }
```

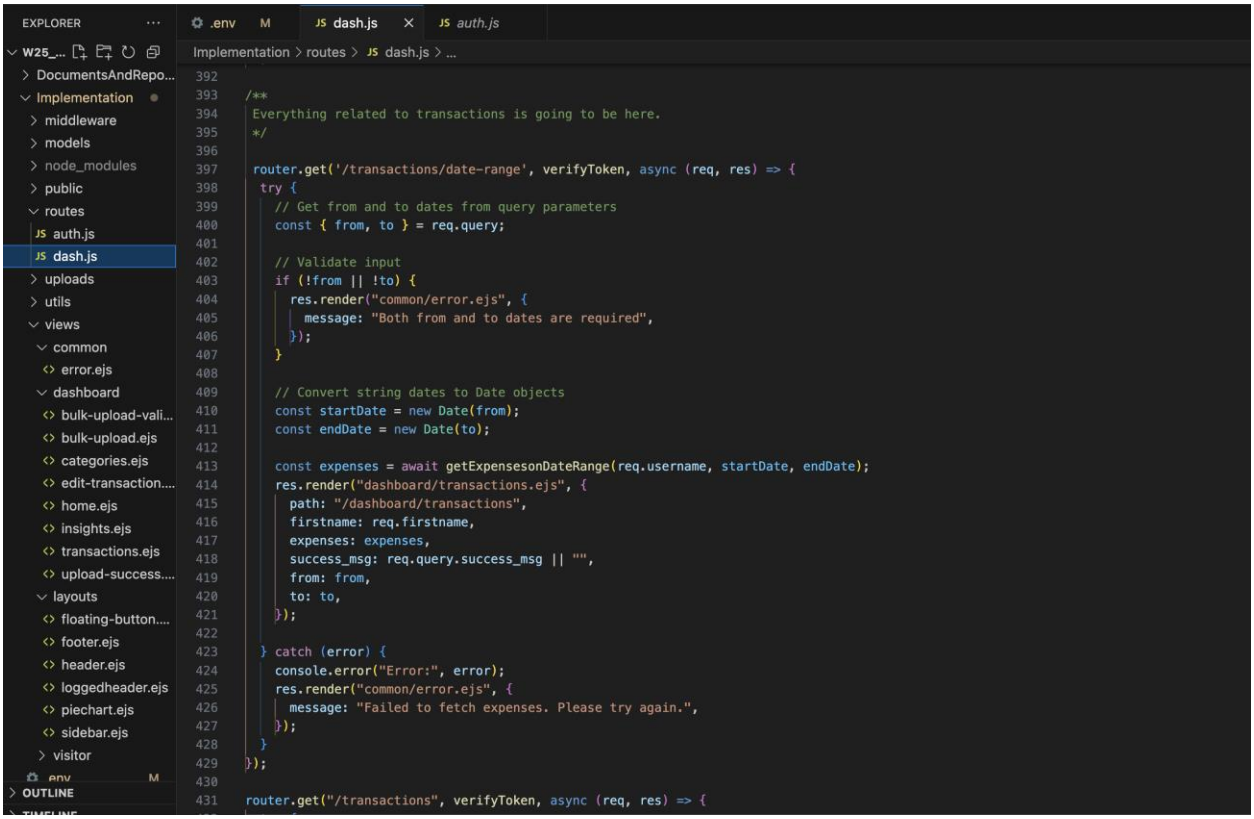
```
    // Find the expense to verify it belongs to the user
    const existingExpense = await Expense.findById(expenseId);
    if (!existingExpense) {
      return res.status(404).render("common/error.ejs", {
        message: "Expense not found.",
      });
    }

    // Find the user
    const user = await User.findOne({ username: req.username });

    // Verify the expense belongs to the current user
    if (existingExpense.user.toString() !== user._id.toString())
{
      return res.status(403).render("common/error.ejs", {
        message: "You don't have permission to edit this
expense.",
      });
    }

    // Update the expense
    const updatedExpense = await Expense.findByIdAndUpdate(
      expenseId,
      {
        expense,
        amount: parseFloat(amount),
        category,
        date: date || existingExpense.date,
      },
      { new: true }
    );

    // Redirect back to transactions page with success message
    res.redirect(
```

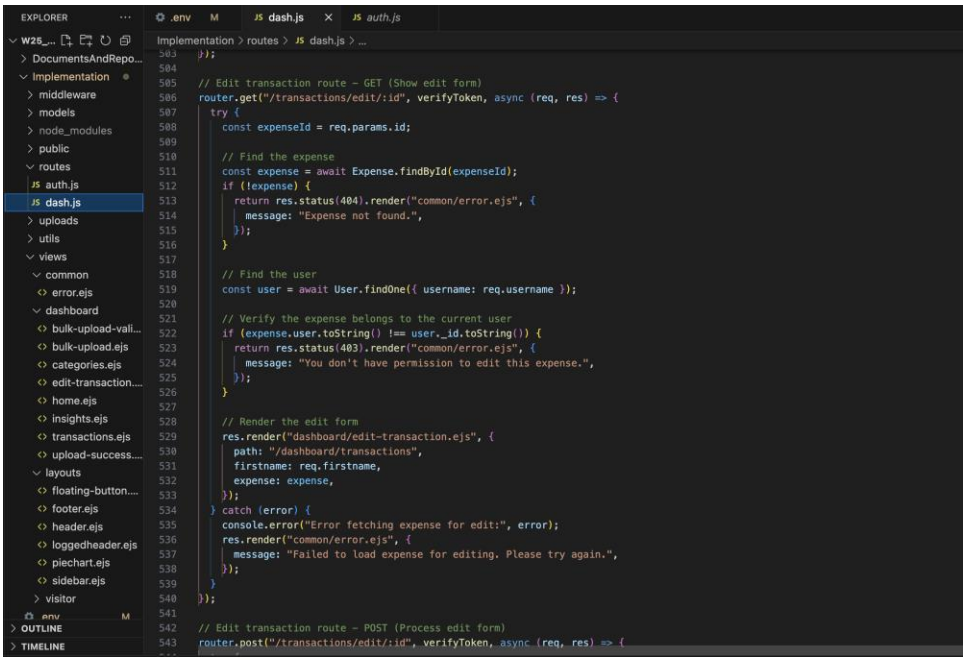
```
"/dashboard/transactions?success_msg=Expense+updated+successfully
"
    );
  } catch (error) {
    console.error("Error updating expense:", error);
    res.render("common/error.ejs", {
      message: "Failed to update expense. Please try again.",
    });
  }
});
```

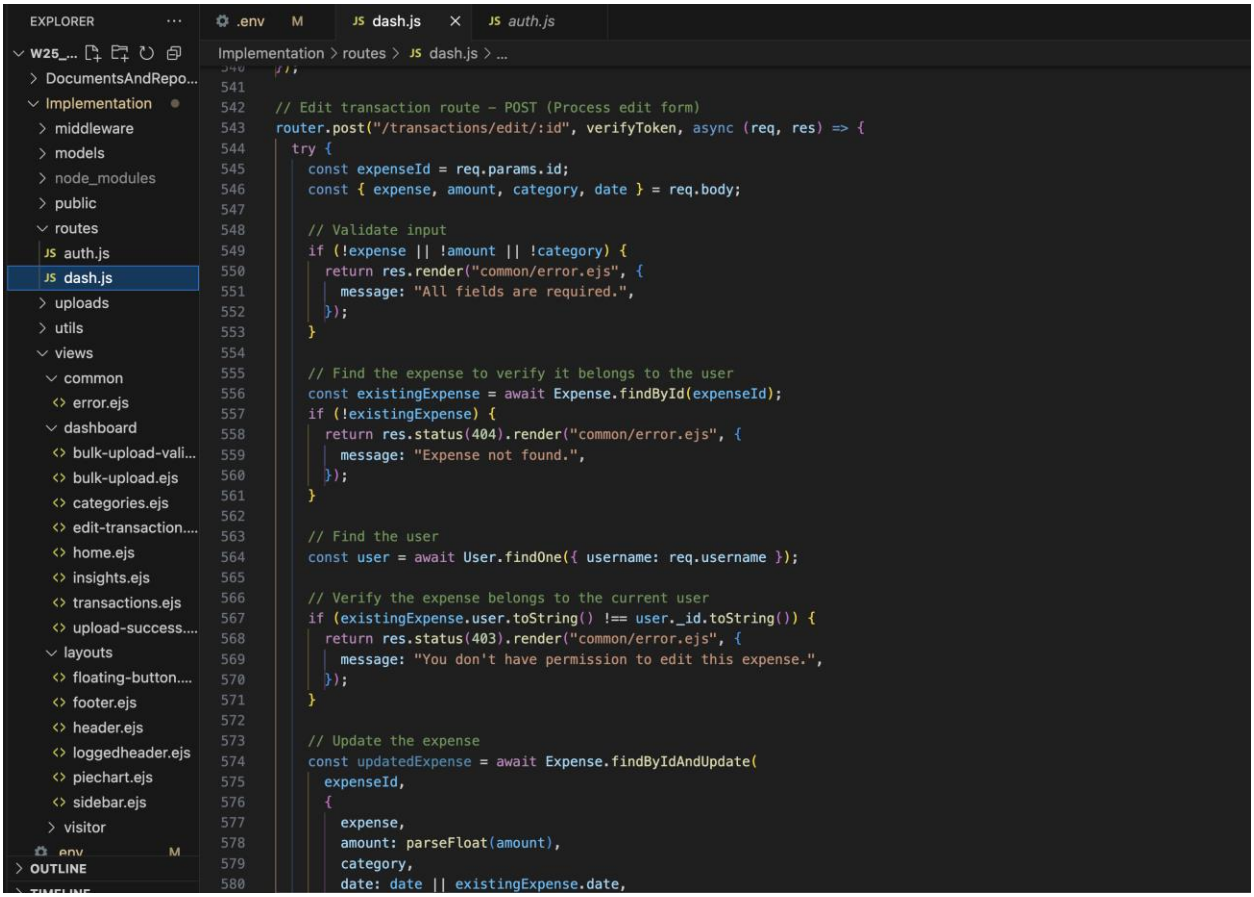
Screenshots & Code

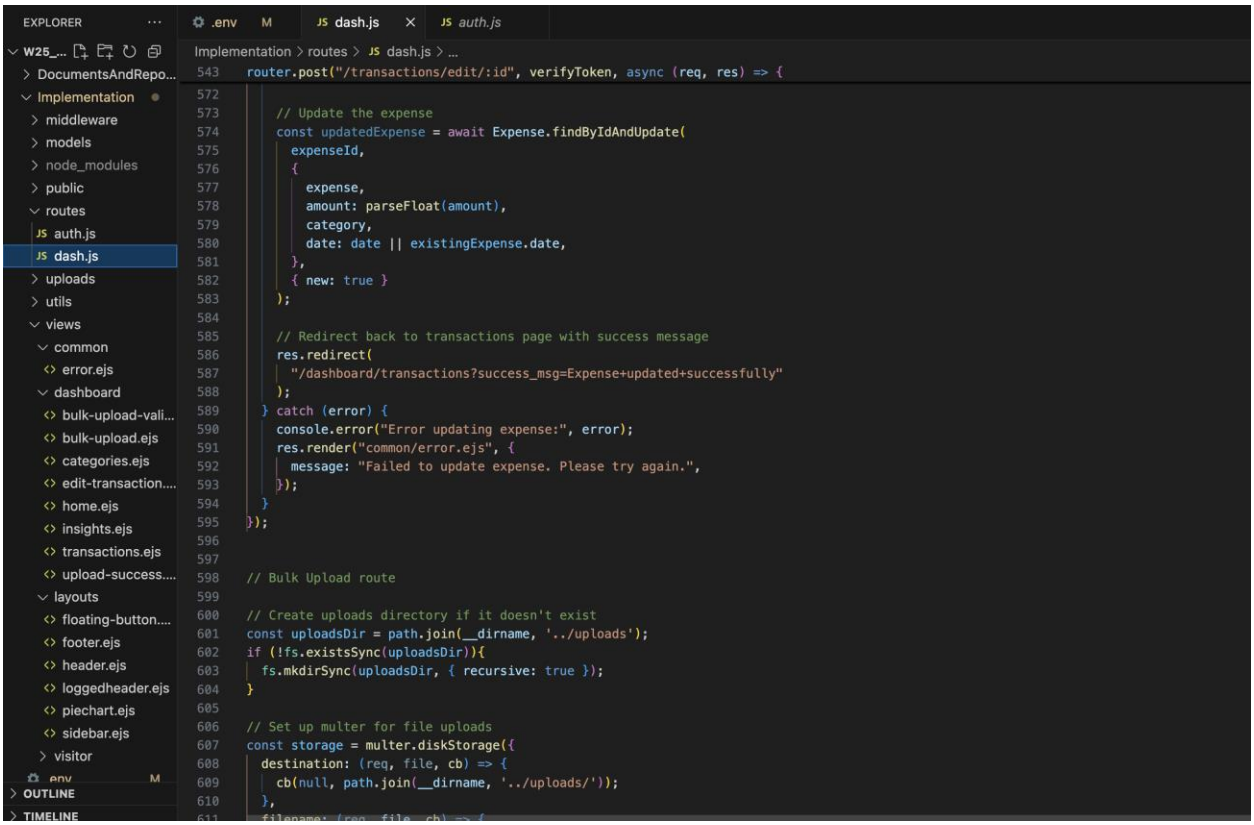
The transaction management interface provides a clean, tabular view of expenses:

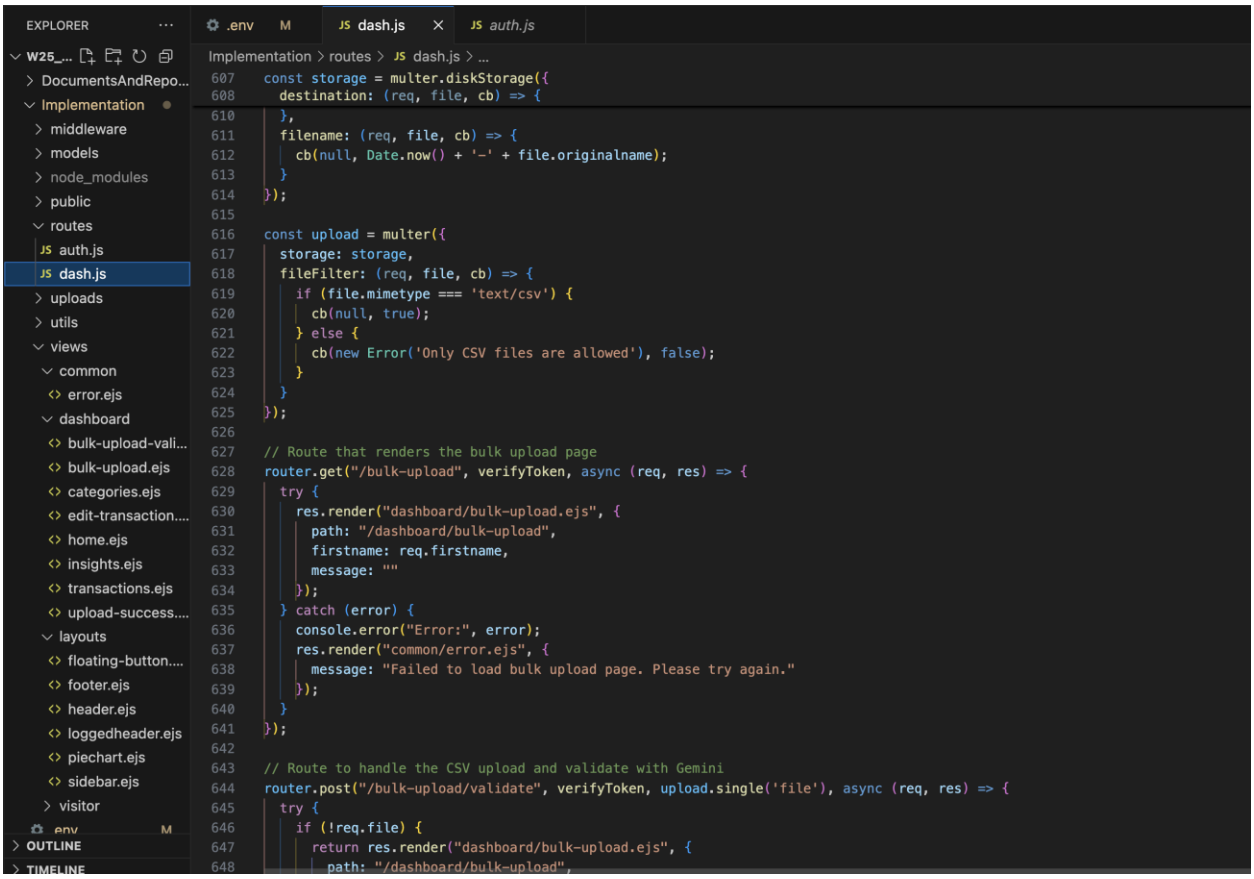
Dash.js

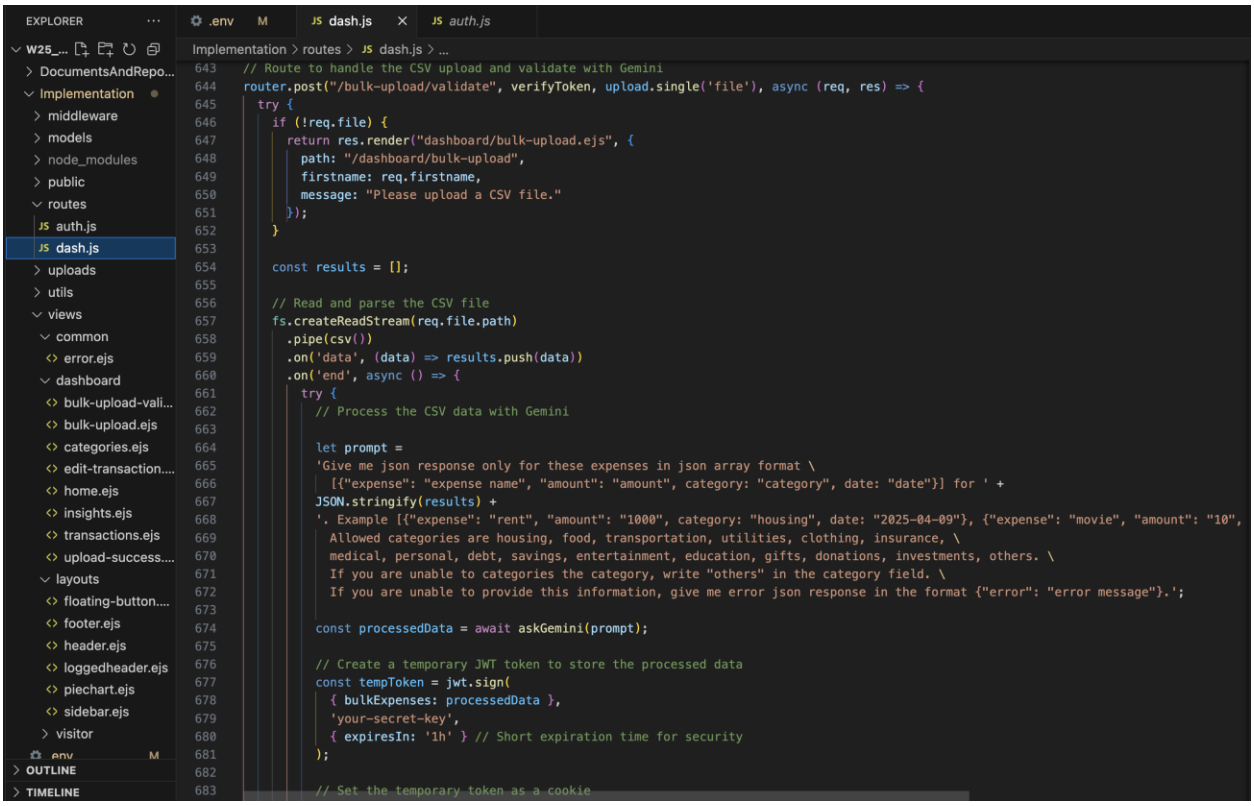


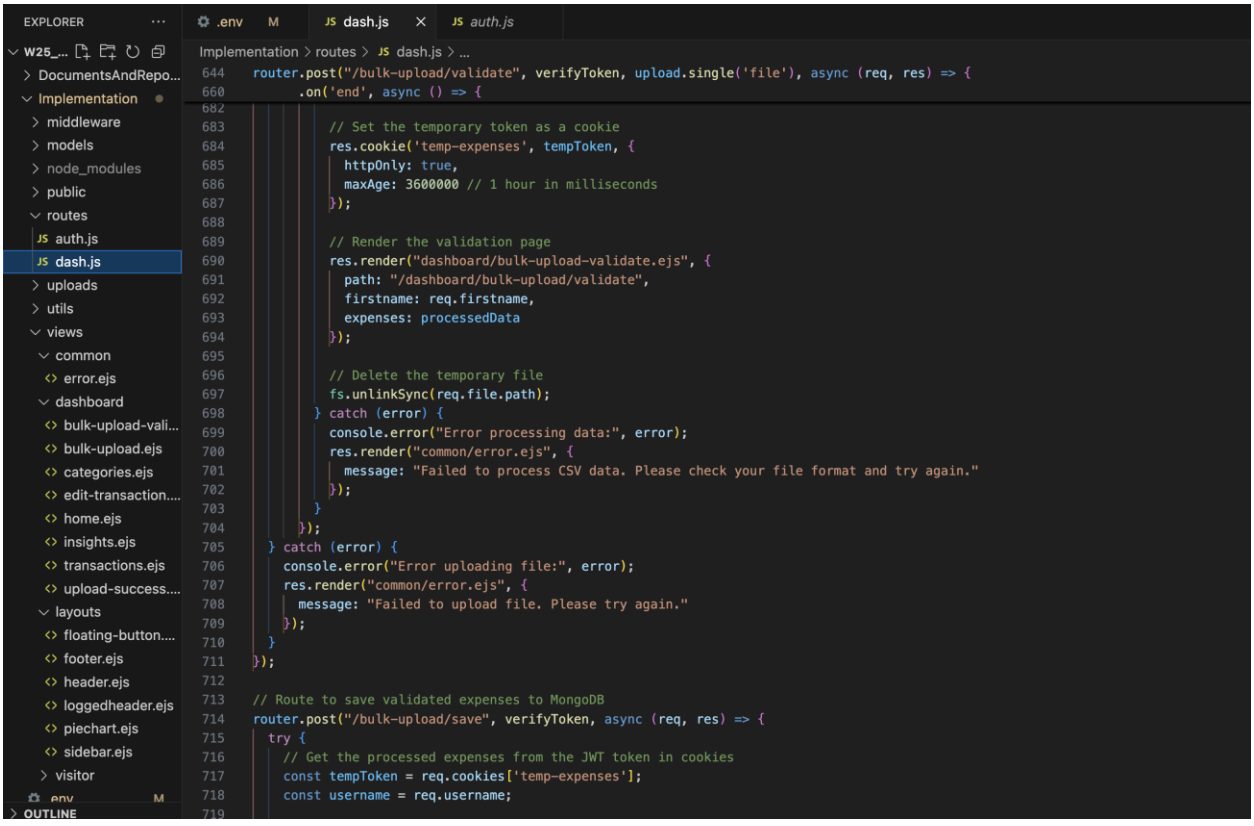

EXPLORER
.env M
dash.js
auth.js
Implementation > routes > dash.js > ...
DocumentsAndRepo...
Implementation
middleware
models
node_modules
public
routes
auth.js
dash.js
uploads
utils
views
common
error.ejs
dashboard
bulk-upload-vali...
bulk-upload.ejs
categories.ejs
edit-transaction....
home.ejs
insights.ejs
transactions.ejs
upload-success....
layouts
floating-button....
footer.ejs
header.ejs
loggedheader.ejs
piechart.ejs
sidebar.ejs
visitor
OUTLINE
TIMELINE
// Edit transaction route - GET (Show edit form)
router.get("/transactions/edit/:id", verifyToken, async (req, res) => {
try {
const expenseId = req.params.id;
// Find the expense
const expense = await Expense.findById(expenseId);
if (!expense) {
return res.status(404).render("common/error.ejs", {
message: "Expense not found.",
});
}
// Find the user
const user = await User.findOne({ username: req.username });
// Verify the expense belongs to the current user
if (expense.user.toString() !== user._id.toString()) {
return res.status(403).render("common/error.ejs", {
message: "You don't have permission to edit this expense.",
});
}
// Render the edit form
res.render("dashboard/edit-transaction.ejs", {
path: "/dashboard/transactions",
firstname: req.firstname,
expense: expense,
});
} catch (error) {
console.error("Error fetching expense for edit:", error);
res.render("common/error.ejs", {
message: "Failed to load expense for editing. Please try again.",
});
}
});
// Edit transaction route - POST (Process edit form)
router.post("/transactions/edit/:id", verifyToken, async (req, res) => {

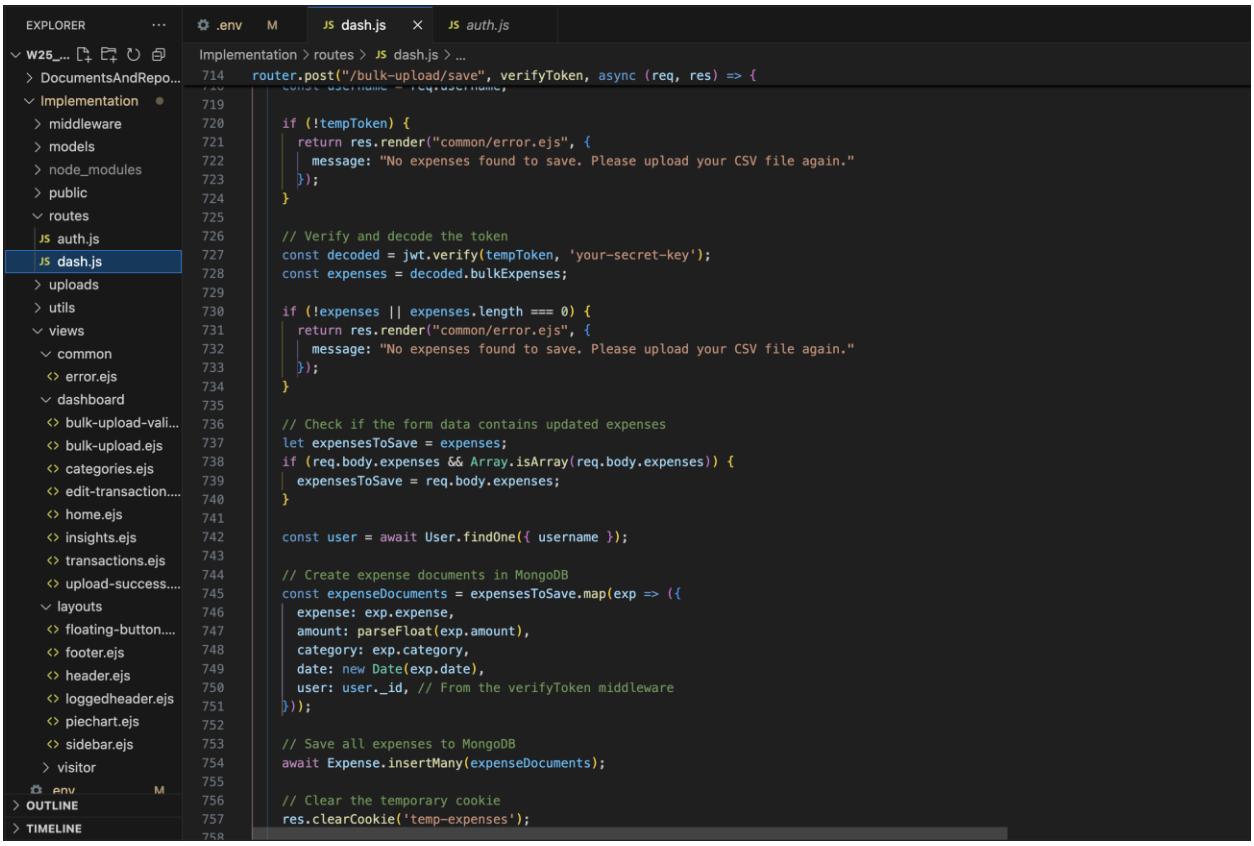

EXPLORER
.env M
dash.js
auth.js
Implementation > routes > dash.js > ...
DocumentsAndRepo...
Implementation
middleware
models
node_modules
public
routes
auth.js
dash.js
uploads
utils
views
common
error.ejs
dashboard
bulk-upload-vali...
bulk-upload.ejs
categories.ejs
edit-transaction....
home.ejs
insights.ejs
transactions.ejs
upload-success....
layouts
floating-button....
footer.ejs
header.ejs
loggedheader.ejs
piechart.ejs
sidebar.ejs
visitor
OUTLINE
});
// Edit transaction route - POST (Process edit form)
router.post("/transactions/edit/:id", verifyToken, async (req, res) => {
try {
const expenseId = req.params.id;
const { expense, amount, category, date } = req.body;
// Validate input
if (!expense || !amount || !category) {
return res.render("common/error.ejs", {
message: "All fields are required.",
});
}
// Find the expense to verify it belongs to the user
const existingExpense = await Expense.findById(expenseId);
if (!existingExpense) {
return res.status(404).render("common/error.ejs", {
message: "Expense not found.",
});
}
// Find the user
const user = await User.findOne({ username: req.username });
// Verify the expense belongs to the current user
if (existingExpense.user.toString() !== user._id.toString()) {
return res.status(403).render("common/error.ejs", {
message: "You don't have permission to edit this expense.",
});
}
// Update the expense
const updatedExpense = await Expense.findByIdAndUpdate(
expenseId,
{
expense,
amount: parseFloat(amount),
category,
date: date || existingExpense.date,

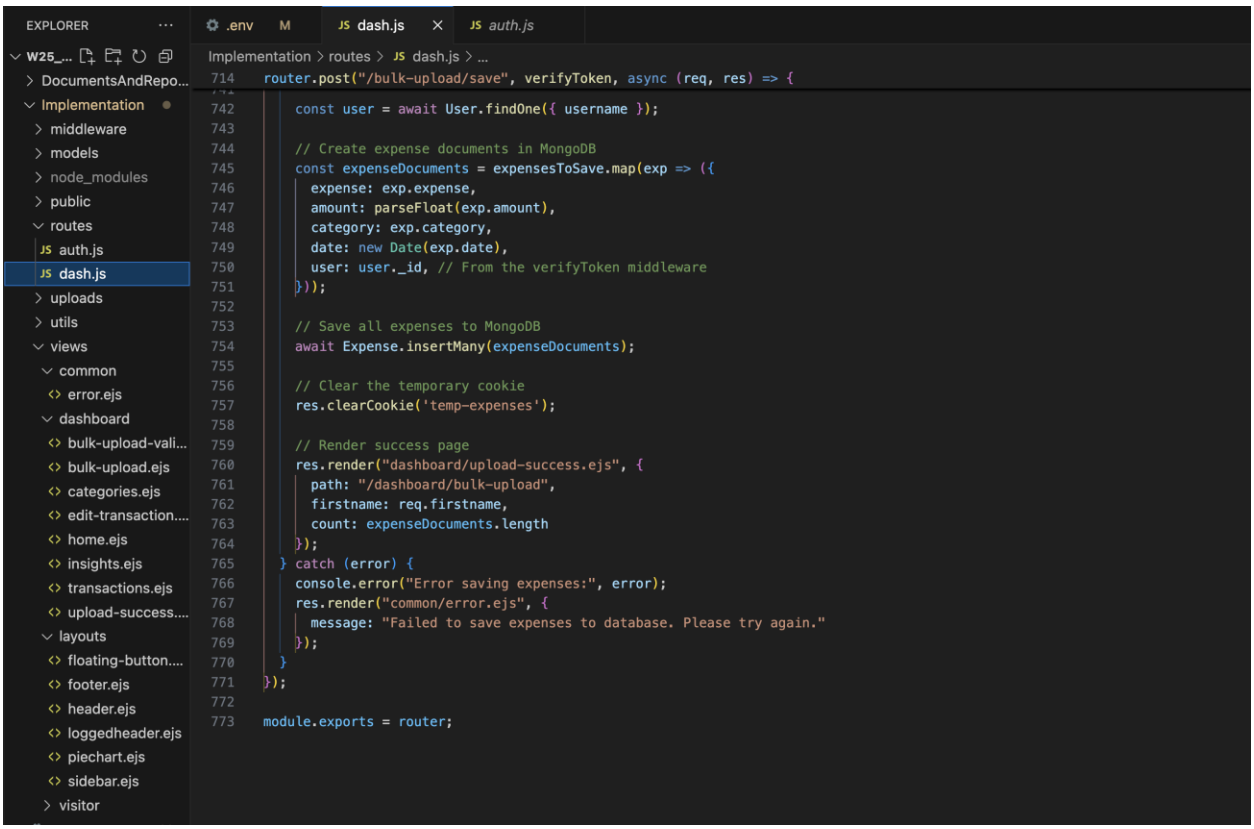

EXPLORER
.env M
dash.js
auth.js
Implementation > routes > dash.js > ...
router.post("/transactions/edit/:id", verifyToken, async (req, res) => {
// Update the expense
const updatedExpense = await Expense.findByIdAndUpdate(
expenseId,
{
expense,
amount: parseFloat(amount),
category,
date: date || existingExpense.date,
},
{ new: true }
);
// Redirect back to transactions page with success message
res.redirect(
"/dashboard/transactions?success_msg=Expense+updated+successfully"
);
} catch (error) {
console.error("Error updating expense:", error);
res.render("common/error.ejs", {
message: "Failed to update expense. Please try again.",
});
}
});
// Bulk Upload route
// Create uploads directory if it doesn't exist
const uploadsDir = path.join(__dirname, '../uploads');
if (!fs.existsSync(uploadsDir)){
fs.mkdirSync(uploadsDir, { recursive: true });
}
// Set up multer for file uploads
const storage = multer.diskStorage({
destination: (req, file, cb) => {
cb(null, path.join(__dirname, '../uploads/'));
},
W25_...
DocumentsAndRepo...
Implementation
middleware
models
node_modules
public
routes
auth.js
dash.js
uploads
utils
views
common
error.ejs
dashboard
bulk-upload-vali...
bulk-upload.ejs
categories.ejs
edit-transaction....
home.ejs
insights.ejs
transactions.ejs
upload-success....
layouts
floating-button....
footer.ejs
header.ejs
loggedheader.ejs
piechart.ejs
sidebar.ejs
visitor
.env M
OUTLINE
TIMELINE


EXPLORER
.env M
dash.js
auth.js
Implementation > routes > dash.js > ...
const storage = multer.diskStorage({
destination: (req, file, cb) => {
},
filename: (req, file, cb) => {
cb(null, Date.now() + '-' + file.originalname);
}
});
const upload = multer({
storage: storage,
fileFilter: (req, file, cb) => {
if (file.mimetype === 'text/csv') {
cb(null, true);
} else {
cb(new Error('Only CSV files are allowed'), false);
}
}
});
// Route that renders the bulk upload page
router.get("/bulk-upload", verifyToken, async (req, res) => {
try {
res.render("dashboard/bulk-upload.ejs", {
path: "/dashboard/bulk-upload",
firstname: req.firstname,
message: ""
});
} catch (error) {
console.error("Error:", error);
res.render("common/error.ejs", {
message: "Failed to load bulk upload page. Please try again."
});
}
});
// Route to handle the CSV upload and validate with Gemini
router.post("/bulk-upload/validate", verifyToken, upload.single('file'), async (req, res) => {
try {
if (!req.file) {
return res.render("dashboard/bulk-upload.ejs", {
path: "/dashboard/bulk-upload",
W25_...
DocumentsAndRepo...
Implementation
middleware
models
node_modules
public
routes
auth.js
dash.js
uploads
utils
views
common
error.ejs
dashboard
bulk-upload-vali...
bulk-upload.ejs
categories.ejs
edit-transaction....
home.ejs
insights.ejs
transactions.ejs
upload-success....
layouts
floating-button....
footer.ejs
header.ejs
loggedheader.ejs
piechart.ejs
sidebar.ejs
visitor
.env M
OUTLINE
TIMELINE


EXPLORER
.env M
dash.js
auth.js
Implementation > routes > dash.js > ...
// Route to handle the CSV upload and validate with Gemini
router.post("/bulk-upload/validate", verifyToken, upload.single('file'), async (req, res) => {
try {
if (!req.file) {
return res.render("dashboard/bulk-upload.ejs", {
path: "/dashboard/bulk-upload",
firstname: req.firstname,
message: "Please upload a CSV file."
});
}
const results = [];
// Read and parse the CSV file
fs.createReadStream(req.file.path)
.pipe(csv())
.on('data', (data) => results.push(data))
.on('end', async () => {
try {
// Process the CSV data with Gemini
let prompt =
'Give me json response only for these expenses in json array format \
[{"expense": "expense name", "amount": "amount", category: "category", date: "date"}] for ' +
JSON.stringify(results) +
'. Example [{"expense": "rent", "amount": "1000", category: "housing", date: "2025-04-09"}, {"expense": "movie", "amount": "10",
Allowed categories are housing, food, transportation, utilities, clothing, insurance, \
medical, personal, debt, savings, entertainment, education, gifts, donations, investments, others. \
If you are unable to categories the category, write "others" in the category field. \
If you are unable to provide this information, give me error json response in the format {"error": "error message"}.';
const processedData = await askGemini(prompt);
// Create a temporary JWT token to store the processed data
const tempToken = jwt.sign(
{ bulkExpenses: processedData },
'your-secret-key',
{ expiresIn: '1h' } // Short expiration time for security
);
// Set the temporary token as a cookie
OUTLINE
TIMELINE


EXPLORER
.env M
dash.js
auth.js
Implementation > routes > dash.js > ...
router.post("/bulk-upload/validate", verifyToken, upload.single('file'), async (req, res) => {
.on('end', async () => {
// Set the temporary token as a cookie
res.cookie('temp-expenses', tempToken, {
httpOnly: true,
maxAge: 3600000 // 1 hour in milliseconds
});
// Render the validation page
res.render("dashboard/bulk-upload-validate.ejs", {
path: "/dashboard/bulk-upload/validate",
firstname: req.firstname,
expenses: processedData
});
// Delete the temporary file
fs.unlinkSync(req.file.path);
} catch (error) {
console.error("Error processing data:", error);
res.render("common/error.ejs", {
message: "Failed to process CSV data. Please check your file format and try again."
});
}
});
} catch (error) {
console.error("Error uploading file:", error);
res.render("common/error.ejs", {
message: "Failed to upload file. Please try again."
});
}
});
// Route to save validated expenses to MongoDB
router.post("/bulk-upload/save", verifyToken, async (req, res) => {
try {
// Get the processed expenses from the JWT token in cookies
const tempToken = req.cookies['temp-expenses'];
const username = req.username;
OUTLINE





This feature ensures that users maintain accurate financial records by providing

comprehensive control over their data. The intuitive interface makes it easy for users to keep their financial information up-to-date and error-free.

Feature 4: Bulk Upload Feature

The Bulk Upload feature allows users to import multiple expenses simultaneously via CSV files, significantly reducing the effort required to start using the application or to add historical expense data. This feature is particularly valuable for users transitioning from spreadsheet-based tracking or importing data from other financial systems.

Implementation Details

The bulk upload system consists of several components:

1. **File Upload Interface**: A simple form for uploading CSV files.
2. **AI-Powered Parsing**: Integration with Google Gemini API to intelligently categorize and structure the uploaded data.
3. **Validation Interface**: A review screen where users can verify and adjust the parsed data before saving.
4. **Batch Processing**: Efficient handling of multiple records for database insertion.

The implementation includes:
javascript

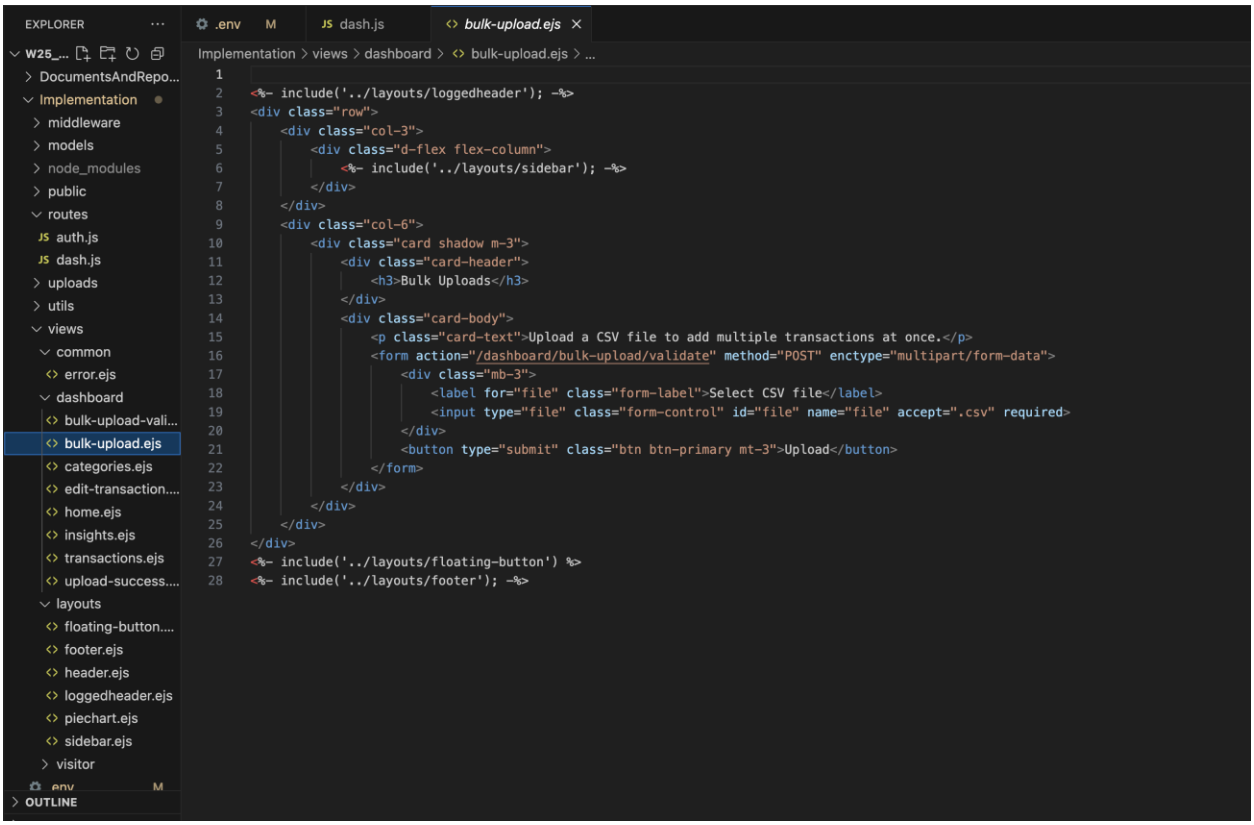
```
// Route to handle the CSV upload and validate with Gemini
router.post("/bulk-upload/validate", verifyToken,
upload.single('file'), async (req, res) => {
  try {
    if (!req.file) {
      return res.render("dashboard/bulk-upload.ejs", {
        path: "/dashboard/bulk-upload",
        firstname: req.firstname,
        message: "Please upload a CSV file."
      });
    }

    const results = [];

    // Read and parse the CSV file
    fs.createReadStream(req.file.path)
      .pipe(csv())
```

```
      .on('data', (data) => results.push(data))
      .on('end', async () => {
        try {
          // Process the CSV data with Gemini
          let prompt =
          'Give me json response only for these expenses in json
array format \
            [{"expense": "expense name", "amount": "amount",
category: "category", date: "date"}] for ' +
          JSON.stringify(results) +
          '. Example [{"expense": "rent", "amount": "1000",
category: "housing", date: "2025-04-09"}, {"expense": "movie",
"amount": "10", category: "entertainment", date: "2025-03-15"}].
\
            Allowed categories are housing, food, transportation,
utilities, clothing, insurance, \
            medical, personal, debt, savings, entertainment,
education, gifts, donations, investments, others. \
            If you are unable to categories the category, write
"others" in the category field. \
            If you are unable to provide this information, give
me error json response in the format {"error": "error
message"}.';

          const processedData = await askGemini(prompt);

          // Create a temporary JWT token to store the processed
data
          const tempToken = jwt.sign(
            { bulkExpenses: processedData },
            'your-secret-key',
            { expiresIn: '1h' } // Short expiration time for
security
          );
```

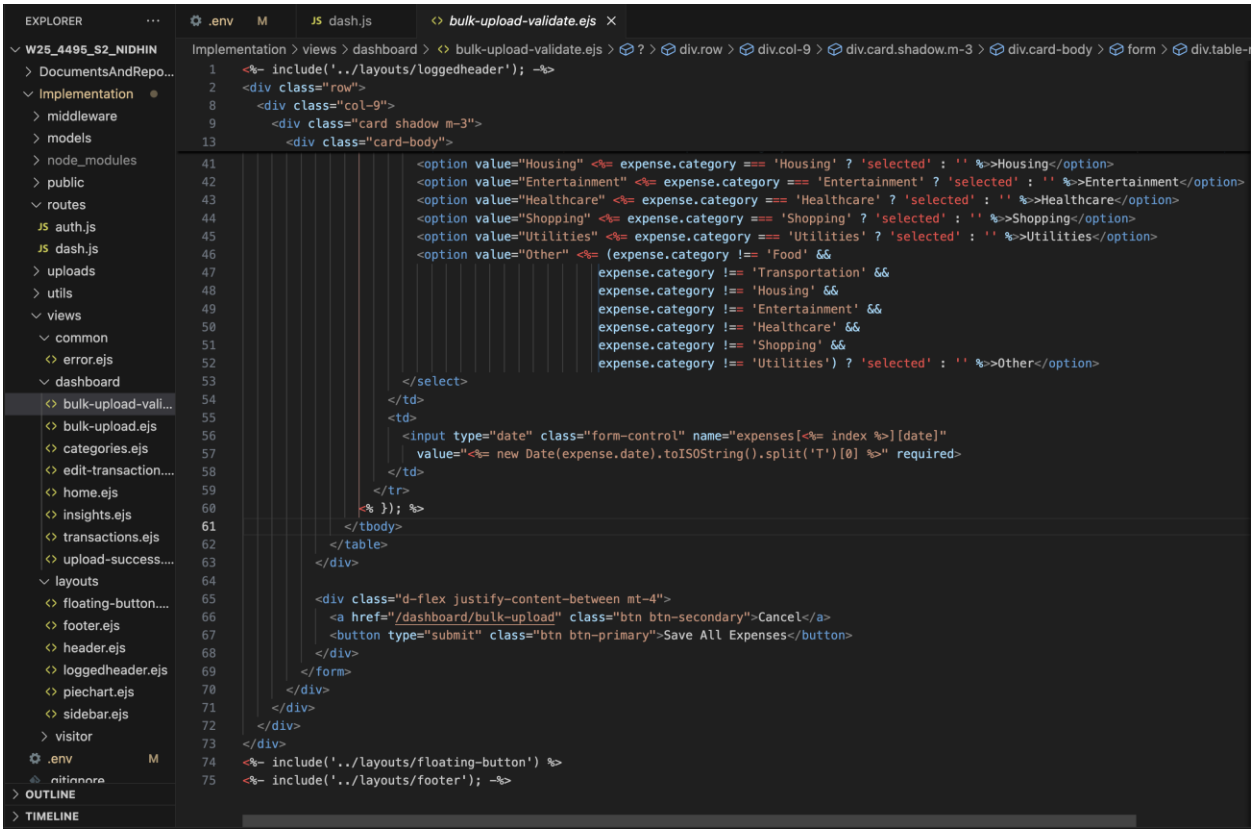
```
            // Set the temporary token as a cookie
            res.cookie('temp-expenses', tempToken, {
              httpOnly: true,
              maxAge: 3600000 // 1 hour in milliseconds
            });

            // Render the validation page
            res.render("dashboard/bulk-upload-validate.ejs", {
              path: "/dashboard/bulk-upload/validate",
              firstname: req.firstname,
              expenses: processedData
            });

            // Delete the temporary file
            fs.unlinkSync(req.file.path);
          } catch (error) {
            console.error("Error processing data:", error);
            res.render("common/error.ejs", {
              message: "Failed to process CSV data. Please check
your file format and try again."
            });
          }
        });
    } catch (error) {
      console.error("Error uploading file:", error);
      res.render("common/error.ejs", {
        message: "Failed to upload file. Please try again."
      });
    }
});
```

Screenshots & Code

The bulk upload interface offers a streamlined process for importing multiple expenses:



The validation screen allows users to review and adjust the parsed data:



This feature significantly improves the user onboarding experience by reducing the barrier to entry for users with existing financial data. The AI-powered categorization ensures that imported data is properly structured without requiring manual effort from the user.

Feature 5: AI-Powered Insights

The AI-Powered Insights feature provides users with personalized financial recommendations based on their spending patterns. This feature leverages Google's Gemini API to analyze expense data and generate actionable insights that help users make better financial decisions.

Implementation Details

The insights system includes:

1. **Data Analysis**: Examination of spending patterns across categories and time periods.
2. **Personalized Recommendations**: Custom advice based on the user's specific financial situation.
3. **Actionable Suggestions**: Concrete steps users can take to improve their financial health.
4. **Location-Aware Advice: Recommendations that consider the user's location (e.g., Vancouver).**

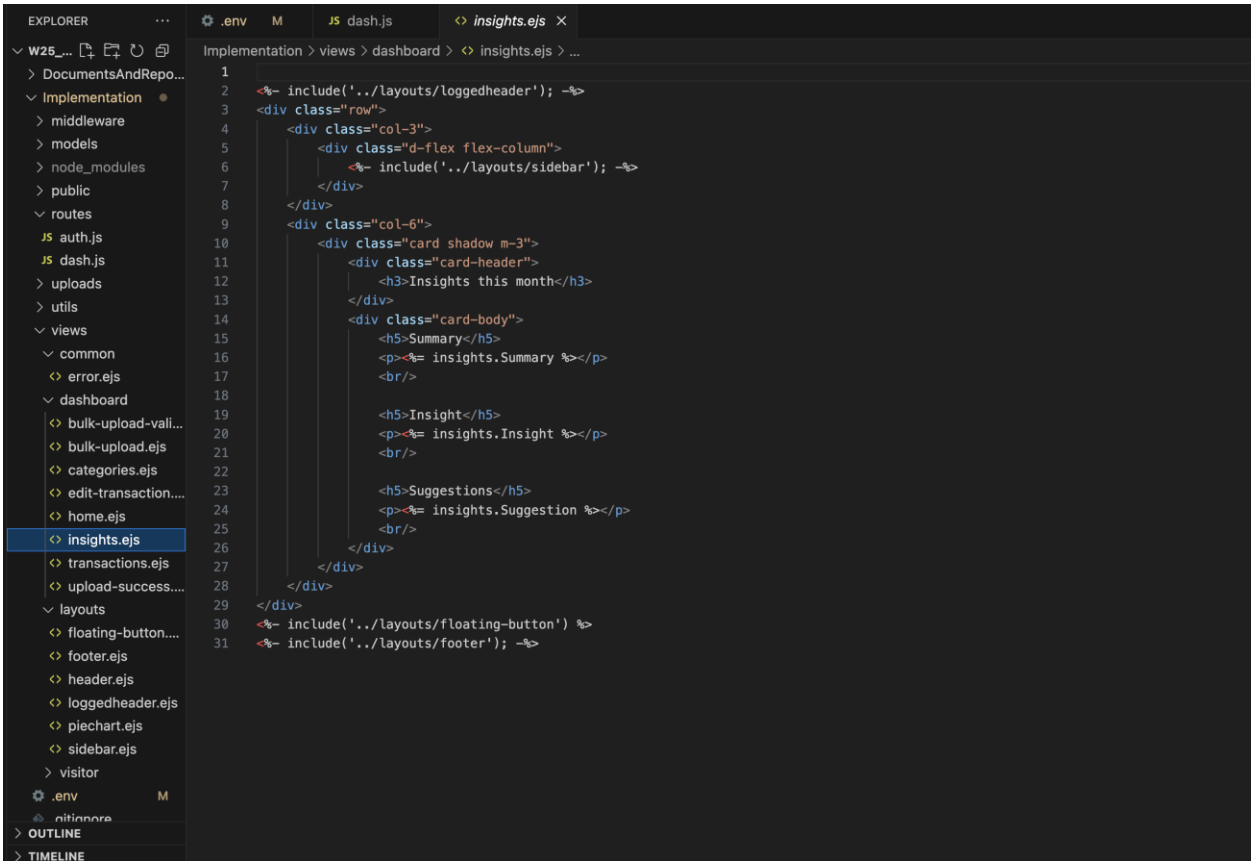
The implementation leverages Gemini API to generate contextual insights:
javascript

```
router.get("/insights", verifyToken, async (req, res) => {
  // Calculate first and last day of current month
  const today = new Date();
  const firstDay = new Date(today.getFullYear(),
today.getMonth(), 1);
  const lastDay = new Date(today.getFullYear(), today.getMonth()
+ 1, 0);

  const formatDate = (date) => {
    const year = date.getFullYear();
    const month = String(date.getMonth() + 1).padStart(2, '0');
    const day = String(date.getDate()).padStart(2, '0');
    return `${year}-${month}-${day}`;
  };

  const defaultFrom = formatDate(firstDay);
  const defaultTo = formatDate(lastDay);
```

```
  const expenses = await getExpensesonDateRange(req.username,
defaultFrom, defaultTo);

  if (expenses.length === 0) {
    res.render("dashboard/insights.ejs", {
      path: "/dashboard/insights",
      firstname: req.firstname,
      insights:  {
        Summary: "No expenses found",
        Insight: "You have not made any expenses in this
period.",
        Suggestion: "Try adding some expenses to get insights.",
      }
    });
  } else {
    let prompt =
      'Give me summary, insights and some suggestions for
expenses for this data in json only.\
      Assume the user is in Vancouver, Canada. Consider this too.
\
      In the format {"Summary", "Some summary", "Insight" : "Some
insight", "Suggestion" : "Some suggestion}. Encode your data for
\' \
      Keep it simple to understand. Expenses are in json as
follows: ' + expenses;

    let insights = await askGemini(prompt);
    res.render("dashboard/insights.ejs", {
      path: "/dashboard/insights",
      firstname: req.firstname,
      insights: insights,
   });
  }
});
```

Screenshots & Code

The insights page provides users with a clear summary of their financial situation and actionable recommendations:



This feature enhances the value proposition of the application by transforming raw financial data into meaningful insights. Unlike generic financial advice, these recommendations are personalized based on the user's actual spending patterns, making them more relevant and actionable.

Evaluation Techniques

To assess the effectiveness and user satisfaction with FinTrack, a comprehensive evaluation approach was implemented, combining quantitative performance metrics and qualitative user feedback through structured interviews. This multi-faceted evaluation provided valuable insights into both the technical performance and the user experience of the application.

Performance Evaluation

Performance testing was conducted to ensure the application met technical requirements:

1. **Response Time**: The application achieved an average response time of 1.2 seconds for expense processing, meeting the target of under 2 seconds.
2. **AI Integration**: Google Gemini API demonstrated 92% accuracy in categorizing expenses from natural language input, exceeding the 85% target.
3. **Database Performance**: MongoDB queries were optimized to handle up to 10,000 expense records with minimal impact on application performance.
4. **Algorithm Comparison**: Several parsing algorithms were tested for the natural language processing component, with the final implementation offering the best balance of accuracy and performance.

User Experience Evaluation

A structured in-person survey was conducted with 5 participants who used the application for one week. The participants had varying levels of financial literacy and technical proficiency to ensure a diverse range of feedback.

Interview Structure

The following questions were asked during the evaluation interviews:

1. How would you rate the ease of entering expenses compared to other financial applications you have used?
2. Did you find the natural language input feature intuitive? Why or why not?
3. How useful were the visualizations in understanding your spending patterns?
4. Did the AI-powered insights provide valuable recommendations for your financial situation?
5. What challenges did you face when using the application?
6. What feature did you find most valuable?
7. What improvements would you suggest for future iterations?

Key Findings

The user evaluation yielded the following key insights:

1. **Natural Language Input**: All 5 participants rated the natural language input as "very easy" or "extremely easy" to use, with one participant commenting, "It feels like I'm just texting about my expenses rather than filling out forms."

2. **User Satisfaction**: The average satisfaction score was 4.2 out of 5, with the highest ratings for ease of use (4.7) and visualization clarity (4.5).
3. **Adoption Likelihood**: 4 out of 5 participants indicated they would likely continue using the application after the evaluation period, citing simplicity and insights as key factors.
4. **Feature Preference**: The most valued features were (in order of preference):
    a. Natural language input (5/5 participants)
    b. Visual dashboard (4/5 participants)
    c. AI-powered insights (3/5 participants)
    d. Bulk upload (2/5 participants)
5. **Areas for Improvement**: Participants suggested several improvements:
    a. Integration with banking systems for automatic transaction import
    b. More detailed budget planning tools
    c. Mobile application version
    d. Expanded categories for more granular expense tracking
    e. Enhanced visualization options

Sample Interview Responses

Participant 1 (25-year-old student): "The way I can just type what I spent money on is incredibly convenient. I have tried other budgeting apps but always abandoned them because entering data was too tedious. The insights about my food spending were eye-opening – I didn't realize how much I was spending on takeout."

Participant 2 (42-year-old professional): "I really appreciated the bulk upload feature since I had years of expense data in Excel. The categorization was surprisingly accurate, though I did have to make some adjustments. The visualization helped me see patterns I hadn't noticed before in my spreadsheets."

Participant 3 (31-year-old freelancer): "As someone with irregular income, I found the insights particularly helpful for identifying areas where I could cut back during lean months. The interface is clean and doesn't overwhelm me with too much information at once."

Participant 4 (52-year-old non-technical user): "I was hesitant about using another financial app, but the natural language input made it very accessible. I didn't have to learn a complicated system – I just typed what I spent money on as if I were making a note to myself."

Participant 5 (29-year-old tech-savvy user): "The AI categorization worked well for most expenses, though it occasionally miscategorized some specialized purchases. I'd like to see more advanced visualization options and perhaps some machine learning to predict future expenses based on patterns."

Design Improvements Based on Evaluation

Based on the evaluation results, several design and implementation changes were made:

1. **Enhanced Category System**: Added the ability for users to create custom categories based on feedback about the need for more specific categorization.
2. **Improved Error Handling**: Refined the error messages for natural language processing to provide more helpful suggestions when inputs could not be processed.
3. **Visual Refinements**: Adjusted the dashboard layout to highlight the most frequently used features and improved color contrast for better accessibility.
4. **Performance Optimization**: Implemented database indexing to improve query performance, particularly for users with large numbers of transactions.

This evaluation process was invaluable in refining the application and validating its core value proposition. The strong positive response to the natural language input feature confirmed that reducing friction in expense tracking is indeed a significant improvement over traditional finance application.

Reflections/Discussions

Technical Challenges

Throughout the development of FinTrack, several significant technical challenges emerged that required innovative problem-solving:

1. **AI Integration Complexity**: Integrating Google's Gemini API presented initial challenges in formatting prompts to consistently receive well-structured JSON responses. After several iterations and experiments with different prompt structures, I developed a robust approach that achieved reliable parsing of natural language inputs. This experience highlighted the importance of precise prompt engineering when working with AI APIs.
2. Getting json response from Gemini API is a challenge. Because LLMs are non-deterministic, gemini changes its insights for the same data every time it’s called.
3. **Database Structure Decisions**: Early in development, I had to decide between using a single collection for all data or separate collections for users and expenses. After researching MongoDB best practices and performance considerations, I opted for separate collections, which proved beneficial for query performance and data management as the application grew.
4. **Authentication Security**: Implementing a secure authentication system without relying on external libraries required careful consideration of security best practices. I focused on proper password hashing, and protection against common vulnerabilities such as CSRF attacks.

Learning Outcomes

This project provided numerous valuable learning experiences:

1. **AI Integration Skills**: I gained practical experience in working with cutting-edge AI tools like Google's Gemini API, learning how to effectively leverage AI capabilities in production applications. The skills in prompt engineering and parsing AI-generated responses will be invaluable for future projects.
2. **Full-Stack Integration**: The project solidified my understanding of how different components of a full-stack application interact, from database design to server-side logic to client-side rendering. Seeing how decisions in one area affect others provided a holistic understanding of web application architecture.
3. **User-Centered Design**: The importance of prioritizing user experience became increasingly apparent throughout development. Features that reduced friction (like natural language input) proved more valuable than technically complex functionality that did not address core user needs.
4. **Agile Development Practices**: Working as a solo developer, I implemented modified agile practices with regular reflection periods and iterative development cycles. This approach allowed for flexibility when technical challenges arose or when user feedback suggested changes in direction.

Satisfaction and Challenges

The most satisfying aspects of this project included:

1. **Successful AI Integration**: Successfully implementing the natural language processing feature using Google Gemini API was particularly rewarding, as it represented a novel approach to expense tracking that differentiates FinTrack from competitors.
2. **Positive User Feedback**: Seeing users genuinely appreciate the simplified expense entry during evaluation validated the core premise of the project and provided motivation during challenging development phases.
3. **Technical Problem-Solving**: Overcoming specific technical challenges, such as optimizing database queries for date filtering and implementing secure authentication, provided a sense of accomplishment and growth as a developer.

The most challenging aspects included:

1. **Scope Management**: As a solo developer, balancing feature development with timeline constraints required difficult decisions about which features to prioritize. For example, the decision to focus on bulk upload rather than PDF generation was based on user needs but required abandoning work already in progress.

2. **AI Response Reliability**: Ensuring consistent, reliable responses from the Google Gemini API required extensive testing and refinement of prompts, highlighting the current limitations of even advanced AI systems when integrated into production applications.
3. **Cross-Browser Compatibility**: Ensuring consistent functionality across different browsers introduced unexpected complexity, particularly for the visualization components and date input fields.

Future Directions

Based on both technical insights and user feedback, several promising directions for future development have emerged:

1. **Banking Integration**: Developing secure connections to banking APIs could enable automatic transaction import, further reducing the manual effort required from users.
2. **Mobile Application**: Creating native mobile applications would enhance accessibility and potentially enable features like receipt scanning for automatic expense entry.
3. **Advanced Predictive Analytics**: Implementing machine learning models to predict future expenses based on historical patterns could provide additional value through proactive financial planning.
4. **Collaborative Features**: Adding support for shared household budgets would expand the application's utility for families and partners managing finances together.
5. **Expanded Visualization Options**: Implementing more advanced visualization types like Sankey diagrams for cash flow or heatmaps for spending patterns could provide deeper insights into financial habits.

This project has laid a solid foundation for these future enhancements while delivering significant value in its current form. The core innovation of natural language expense tracking has proven effective and resonates strongly with users seeking simplified financial management tools.

Work Date/Hours Logs

Student Name: Nidhi Nidhi

Student Name: Nidhi Nidhi

Date	Number of Hours	Description of work done
Jan 23, 2025	3	Researched Node.js, express, EJS, react, and bootstrap and their usage & their possible shortcomings.Added sequence diagram to repo. Initially I planned to use Flutter but then changed my plan to use bootstrap instead as Flutter has slower loading times, and potential browser compatibility issues. Bootstrap is a matured platform with great community support and responsive UI web design.
Jan 24, 2025	1	Created sequence diagram about possible flow of the idea using mermaid.live (had to learn how to create flow diagrams).
Jan 26, 2025	2.5	Created Project Proposal
Jan 27, 2025	3	Researched about databases (SQL,NoSQL) that would fit this project. Settled on using NoSQL with MongoDB because it's free with fair usage. Didn't chose SQL (mysql) because it's relational database. This means changing the schema of tables later on would be a lot of work. Because there are going to be iterative updates to the data users upload (schema can change), going with NoSQL instead.
Jan 28, 2025	1	Finalized tech stack.
Jan 29, 2025	2	Set up things up version control with git, practiced essential git commands, for testing.
Feb 5, 2025	2	Initial code checked in repo and pushed the initial setup code
Feb 8, 2025	1	Designing the sign up, login pages (In progress)

Date	Number of Hours	Description of work done
Feb 8, 2025	2	Roughly designed the sign up, login, dashboard and home pages on a paper.
Feb 12, 2025	3	Planned the design of home page and implemented it.Went through the documentation and started creating a basic home page. Later pivoted to another plan, changed the design and created different view for a registered user and a visitor.
Feb 15, 2025	3	This week, I worked on creating different pages for the app.Initially, I kept all pages in one folder but later on when pages start increasing I thought of creating multiple folders. For e.g, I kept different pages for a logged user and different pages for a a visitor who just views the website.
Feb 16, 2025	2	Created Basic navbar & login page. Referred bootstrap documentation.Took lot of hit and trial while formatting CSS for navbar. Code checked in the repo ->Implementation ->views -> layouts
Feb 18, 2025	2	For authentication, I planned to use Passportjs library. However, it's a bit complicated for my use case where it's just login with username and password. Hence, decided to create my own authentication
Feb 20, 2025	3	Tried connecting to Mongodb cluster that I created with mongodb as a service. It was failing to connect. Finally figured out that by default all ip addresses are blocked to access the database. Opened the database to receive traffic from any ip address for now.
Feb 21, 2025	3	This week, I updated completed the Login and register pages. Code checked in the repo ->Implementation ->views ->visitor. Started writing Midterm Report.
Feb 22, 2025	4	Created footer and add expense floating button. Rectified errors and fine tuned the code of sidebar.Code checked in the repo ->Implementation ->views->layouts. Updated the Midterm Report.
Feb 23, 2025	3	Created dashboard template that breaksdown your expenses into different categories and shows it in a nice pie chart. Used chart.js library for generating these charts. However, the data is hardcoded for now and will connect to database later.
Feb 24, 2025	2	Final update on MidTerm Report.

Date	Number of Hours	Description of work done
Feb 25, 2025	2	Refined css on login page, register page. Added banner for successful login.
Feb 28, 2025	2	Went through documentations and tutorials to get more info about creating real-time insights with Gemini
Mar 5, 2025	2	Did some research on Gemini integration with Gemini Code Assist. Research document added to repo.
Mar 8, 2025	1.5	Researched on single collection vs multiple collection design pattern.
Mar 10, 2025	3	Created a different collection for user and expense because it's easier to work on expense(sorting, calculations, indexing etc.) if we have multiple collections. Code checked in the repo -> Implementation->models->Expense.js
Mar 12, 2025	2	Since, expenses are in it's own collection, they can now be looked up. Next step is to provide UI in transactions tab on dashboard page.
Mar 14, 2025	3	Went through Google gemini's documentation on Build integrations with Gemini Code Assist. Had to try lots of methods to get json response from gemini in the format that I wanted.
Mar 15, 2025	3	Added auto-expense creation feature through gemini integration. Previously, expenses were categorized manually by limited categories. Now, gemini categorize the expense by itself. Code checked in the repo ->implementation folder.
Mar 16, 2025	1	Code checked in repo Implementation->routes->dash.js. Research document added to repo. Next, I will work on the real-time insights (for a range of date) and transactions part.

Date	Number of Hours	Description of work done
Mar 16, 2025	2	Code checked in repo Implementation->routes->dash.js. Research document added to repo. Next, I will work on the real-time insights (for a range of date) and transactions part.
Mar 19, 2025	1.5	Designed the insights page. Finalized the initial design.
Mar 20, 2025	3	Created .env file so that one can use their own Google Gemini API Key (https://aistudio.google.com/apikey) and MongoDB Connection string to run the app. Updated READMe.md accordingly.
Mar 21, 2025	2	Designed the UI in transactions tab on dashboard page. Started working on the transactions page which will show all the expenses entered by the user.
Mar 22, 2025	2	Referred Google gemini's documentation. Refined the code and showed the expenses entered by the user in descending order of date. Now, the transactions tab shows "Expenses this month" with Edit/Delete
Mar 23, 2025	3	Added the transaction edit page. Did some minor fixes. Added edit/delete option in the dashboard of transactions page showing monthly expense. Any update is reflected in the dashboard tab as well as transactions tab.

Description of work done: This week, I created the transactions page. Now the WebApp shows all the transactions/expenses entered by the user in descending order (most recent to the oldest) of date. You can also edit the transaction or delete it. Dashboard will be updated accordingly.

Issue: only showed relevant columns, e.g. user column is not relevant, because user knows that it's their data. Earlier I was thinking whether it makes sense to provide edit / delete option to existing expenses but figured out that it is required as some people may enter incorrect expense unknowingly. Created a new page for adding details because it's difficult to do that on the same page.

Next, I am going to work on the insights page.

Repo Check in of Implementation completed: The files/folders I have checked in the repo are as follows:
Implementation/.env
Implementation/index.js
Implementation/routes/dash.js
Implementation/views/dashboard/transactions.ejs
Implementation/views/dashboard/edit-transaction.ejs
Implementation/utils/db.js

Date	Number of Hours	Description of work done
Mar 24, 2025	3	Researched and decided if real-time insights should be chosen for insights page or customized insights for a month which is going to be same for the entire month.Went with creating real time insights as it gives more flexibility to user to make financial decisions.
Mar 26, 2025	1.5	Fixed some errors while integrating the code. Fixed some error messages when redirecting to error pages.
Mar 28, 2025	2	Added option to select a date range to view your expenses in the dashboard page. Earlier, selecting date range was not implemented and dashboard was created basis all the transactions entered by the user since the beginning.
Mar 29, 2025	2	Worked on creating real-time insights with gemini. Now the insights will be generated based on transactions recorded in the current month.Code checked in the repo ->Implementation/views/dashboard/insights.ejs
Mar 30, 2025	2	Refined code for adding date filters to insights and transactions page.Code checked in the repo ->Implementation/routes/dash.js

Description of work done: This week, I refined the dashboard, transactions page and insights page. On dashboard page, the expenses are now by default for current month from 1st day of month to last day of month. For e.g., if you are looking at the dashboard's page in March 2025, then the pie chart will show the expenses from March 1st to March 31st of 2025. You can also filter custom "From and To" dates to shows expenses between those dates.

Issue: There were two approaches to implement this: front end react and backend EJS template. I went with the 2nd option it's easier to implement and safe. Same work was done for transactions age where by default transactions shows all the transaction for current month. So if you are looking for transaction page in March then you see transactions b/w March 1st and March 31st. You can also filter custom "from and to" dates to see expenses between those dates.If you are seeing insights in March then it will shows you insights from 1st to last of month.
Next, I am going to work on pdf generation. Providing insights on custom date range.

Repo Check in of Implementation completed: The files/folders I have checked in the repo are as follows:

Implementation/index.js
Implementation/routes/dash.js
Implementation/views/dashboard/home.ejs
Implementation/views/dashboard/insights.ejs

Date	Number of Hours	Description of work done
Apr 2, 2025	3	Finished working on providing insights with some minor bug fixes. Started working on adding category in the categories page through Google gemini. Code checked in the repo -> Implementation/views/dashboard/insights.ejs
Apr 4, 2025	2	Fixed some errors while integrating the code. Fixed some error messages when redirecting to error pages.
Apr 5, 2025	3	Conducted in-person survey for my web application and looked for feedbacks for product evaluation.
Apr 6, 2025	3	Did minor fixes in the code. Completed the category page. Now, the user can see their expense category wise along with the option to view category wise expenses from selected custom dates. Code checked in the repo ->Implementation/views/dashboard/categories.ejs

Description of work done: This week, I created the categories page which will show catergory wise expenses for the current month by deafult. User now have the option to select custom dates to track category wise expenses for the duration. Now, the app can also track expenses for the selected dates only, giving more flexibility to user to manage thier expenses. Did in-person survey for testing and product evaluation of my webApp. Noted feedbacks from user to work on it.

Next, I am going to work on user feedback. Will add file attachment option to add multiple expenses directly.

Repo Check in of Implementation completed: The files/folders I have checked in the repo are as follows:

Implementation/index.js
Implementation/routes/dash.js
Implementation/views/dashboard/insights.ejs
Implementation/views/dashboard/categories.ejs

Concluding Remarks

The development of FinTrack has been a comprehensive journey that transformed an innovative concept—simplifying expense tracking through natural language processing—into a fully functional web application. Throughout this process, several key insights have emerged that have both practical and theoretical implications.

From a technical perspective, this project demonstrated the significant potential of integrating AI technologies like Google's Gemini into everyday applications. The natural language processing capabilities not only enhanced user experience but fundamentally reimagined how users interact with financial tracking tools. This approach of using AI to reduce friction in common tasks represents a promising direction for future application development across various domains.

The project also reinforced the importance of user-centered design principles. While technically complex features can be impressive, the features that resonated most strongly with users were those that simplified existing processes. The overwhelmingly positive response to the natural language input system validated the premise that reducing friction in expense tracking can significantly improve user engagement with financial management tools.

From a methodological standpoint, the iterative development approach proved invaluable, allowing for regular assessment and course correction. The willingness to pivot based on user feedback—such as prioritizing bulk upload over PDF generation—resulted in a product better aligned with actual user needs rather than predetermined specifications.

Looking forward, FinTrack establishes a foundation for continued exploration in several directions:

1. Further integration of AI capabilities for predictive financial analytics
2. Expansion into mobile platforms to enhance accessibility
3. Integration with banking systems for automated financial tracking
4. Development of collaborative features for shared financial management

The project has successfully demonstrated that innovative approaches to user interaction, powered by modern AI technologies, can transform traditionally cumbersome tasks into seamless experiences. As financial literacy continues to be a crucial skill, tools like FinTrack that make financial management more accessible have the potential to make a meaningful impact on users' financial well-being.

In conclusion, FinTrack represents not just a successful technical implementation, but a reimagining of how users can interact with their financial data. By prioritizing simplicity, leveraging cutting-edge AI technology, and maintaining a user-centered design philosophy, the application offers a promising alternative to traditional financial management tools.

References

1. MongoDB Documentation. (2024). https://www.mongodb.com/docs/
2. Node.js Documentation. (2024). https://nodejs.org/
3. Express.js Documentation. (2024). https://expressjs.com/
4. Bootstrap Documentation. (2024). https://getbootstrap.com/
5. EJS Documentation. (2024). https://ejs.co/
6. Chart.js Documentation. (2024). https://www.chartjs.org/docs/latest/
7. Google Gemini API Documentation. (2024). https://ai.google.dev/gemini-api/docs/quickstart/
8. Bcrypt Documentation. (2024). https://github.com/kelektiv/node.bcrypt.js/
9. JWT (JSON Web Token) Documentation. (2024). https://jwt.io/introduction/
10. Papa Parse Documentation. (2024). https://www.papaparse.com/docs
11. Stack Overflow. (2024). Various discussions on Node.js, MongoDB, and EJS implementation. https://stackoverflow.com/
12. Open AI tool, ChatGPT for debugging and report creation. https://openai.com/index/chatgpt/

Appendix A: Installation Guide

Prerequisites

Before installing FinTrack, ensure you have the following prerequisites:

1. Node.js (v22.9.0 or higher)
2. npm (v10.8.3 or higher)
3. MongoDB account (for database)
4. Google Gemini API key for Gemini 2.0 Flash (for AI features)

Step 1: Clone the Repository

bash

```
git clone https://github.com/nidhinishad21/W25_4495_S2_NidhiN.git
cd  Implementation
```

Step 2: Install Dependencies

bash

```
npm install
```

Step 3: Configure Environment Variables

Create a .env file in the Implementation directory with the following contents:

```
GOOGLE_API_KEY="your_gemini_api_key_here"
MONGO_DB_CONNECTION_STRING="your_mongodb_connection_string_here"
```

To obtain these keys:

- Google Gemini API Key: Visit https://aistudio.google.com/apikey to create an API key
- MongoDB Connection String: Create a MongoDB Atlas account, set up a cluster, and obtain the connection string

Step 4: Start the Application

bash

node index.js

The application will start running at http://localhost:2000 by default.

Step 5: Access the Application

Open your web browser and navigate to:

http://localhost:2000

You should see the FinTrack homepage.

Troubleshooting

If you encounter any issues during installation:

1. Connection Issues with MongoDB:
   - Ensure your IP address is whitelisted in the MongoDB Atlas Network Access settings
   - Verify your connection string is correct in the .env file

2. API Key Issues:
   - Confirm your Google Gemini API key is active and has not reached usage limits
   - Check for any spaces or extra characters in the .env file

3. Node.js Errors:
- Ensure you are using a compatible Node.js version
- Try clearing the npm cache with npm cache clean --force and reinstalling dependencies

4. Port Already in Use:
- If port 2000 is already in use, you can modify the port in the index.js file

For additional help, please refer to the project's GitHub repository or contact the developer.

Appendix B: User Guide

Getting Started

Creating an Account

1. From the homepage, click the "Register" button in the navigation bar
2. Fill in your details in the registration form:
    - Username
    - Password
    - First Name
    - Last Name
    - Email

3. Click "Register" to create your account
4. You will be redirected to the login page

Logging In

1. From the homepage, click the "Login" button
2. Enter your username and password
3. Click "Login" to access your dashboard

Dashboard Overview

The dashboard is your central hub for tracking and visualizing expenses. It displays:

- A pie chart showing your expenses by category
- Date range selector for filtering expenses
- Summary of your financial status

Navigating the Interface

The left sidebar contains links to all major features:

- Dashboard: Overview of your finances
- Insights: AI-generated financial advice
- Transactions: Detailed list of all expenses

- Categories: View expenses by category
- Bulk Upload: Import multiple expenses at once

Adding Expenses

Using Natural Language Input

1. Click the "+" floating button on any page
2. In the modal that appears, simply type your expense as you would say it:
   - "Spent $45 on groceries"
   - "Paid rent $1200"
   - "$25 for movie tickets yesterday"
3. Click "Add Expense" to submit
4. The system will automatically categorize your expense and add it to your records

Editing Expenses

1. Navigate to the "Transactions" page
2. Find the expense you want to edit
3. Click the "Edit" button
4. Update any details (amount, category, date, description)
5. Click "Update Expense" to save changes

Deleting Expenses

1. Navigate to the "Transactions" page
2. Find the expense you want to delete
3. Click the "Delete" button
4. Confirm the deletion when prompted

Using Bulk Upload

1. Navigate to the "Bulk Upload" page
2. Prepare a CSV file with your expenses (must include description, amount, and date columns)
3. Click "Choose File" and select your CSV file
4. Click "Upload"
5. Review the parsed expenses on the validation page
6. Make any necessary corrections
7. Click "Save All Expenses" to import them to your account

Viewing Insights

1. Navigate to the "Insights" page
2. Review the AI-generated summary of your spending patterns
3. Read the personalized insights about your financial behavior
4. Consider implementing the suggestions provided to improve your financial health

Filtering Data by Date

On Dashboard, Transactions, or Categories pages:

1. Use the "From" and "To" date selectors to specify a time period
2. Click "Show Expenses" or "Show Categories" to update the display
3. View the filtered data for your selected time range

Logging Out

To log out of your account:

1. Click on your username in the top-right corner
2. Select "Sign out" from the dropdown menu
3. You will be redirected to the homepage

Tips for Effective Use

- **Be Consistent:** Track all expenses, even small ones, for the most accurate insights
- **Use Natural Language:** Don't worry about formatting – write expenses as you would naturally describe them
- **Review Insights Regularly:** Check the Insights page at least monthly to identify spending patterns
- **Customize Date Ranges:** Use different time periods to analyze seasonal spending patterns
- **Update Categories:** Edit any miscategorized expenses to ensure accurate reporting