Here’s a detailed report for my project:

Project Report: Expense Tracking Application

Introduction

This report outlines the development of an expense tracking application using modern web development technologies. The project leverages a full-stack JavaScript approach, combining MongoDB for database management, Express.js for backend services, and React.js for the frontend interface. The application allows users to manage, track, and visualize their expenses effectively.

Objectives

- To develop a system for users to add, delete, update, and view their expenses.
- To integrate real-time data updates and provide visual expense reports using pie charts.
- To create an intuitive and user-friendly frontend interface for ease of interaction.
- To build a robust backend for handling CRUD (Create, Read, Update, Delete) operations on expense data.

Technologies Used

1. Backend:
   - Node.js: The core runtime used for building the backend services.
   - Express.js: A web framework for building RESTful APIs.
   - MongoDB Atlas: The cloud-based database used to store user expense data. It is a scalable NoSQL database.
   - Mongoose: An ODM (Object Data Modeling) library for MongoDB and Node.js.
   - Node-cron: A library used for scheduling tasks in Node.js, implemented for periodic background services.

2. Frontend:
   - React.js: A JavaScript library for building user interfaces. React's component-based architecture was used to create dynamic and responsive UI elements.
   - MUI (Material-UI) X Charts: A charting library to visualize the expenses in a pie chart format.
   - React Icons (FaTrash, FaEdit, FaWindowClose): For interactive icons used throughout the app.

3. Environment & Configuration:
   - dotenv: Used to load environment variables from a `.env` file into `process.env`.

- Cors: Middleware used to handle cross-origin requests between the frontend and backend.

Project Phases

1. Backend Development

The first phase focused on creating the backend services using Node.js, Express, and MongoDB.

- Setting Up the Backend:
  - Established a connection to MongoDB Atlas using Mongoose.
  - Implemented middleware for CORS and JSON data parsing.
  - Defined routes for handling expense data (create, read, update, delete).

```
javascript
const express = require("express");
const cors = require("cors");
const mongoose = require("mongoose");
const dotenv = require("dotenv");
const expenseRoute = require("./routes/expense");

dotenv.config();
const app = express();

// MIDDLEWARE
app.use(cors());
app.use(express.json());

// ROUTES
app.use("/expenses", expenseRoute);

// DB Connection
mongoose.connect(process.env.DB_CONNECT)
  .then(() => console.log('DB connection is successful'))
  .catch(err => console.log(err));

app.listen(process.env.PORT, () => {
  console.log(`Server is running on Port ${process.env.PORT}`);
});
```

2. Backend Services for Periodic Updates
Using node-cron, a background service was added to refresh the data periodically.

- Email Notifications for Expenses:
  The backend service was set up to run every second and trigger an expense email service.

```
javascript
const cron = require("node-cron");
const { expenseEmail } = require("./EmailService/Expense");

cron.schedule("* * * * * *", () => {
  expenseEmail();
});
```

3. Frontend Development
The frontend was developed using React.js, providing users with an interface to view, add, update, and delete expenses.

- Expense Tracker Interface:
  Users can enter expense details such as label, amount, and date, which get saved to the backend. The UI displays a list of expenses with options to edit or delete them.

- Expense Chart:
  Using MUI X Charts, the application generates a pie chart based on the expenses, grouped by categories (labels). This gives users a visual overview of their spending patterns.

```
javascript
const groupExpensesByLabel = (expenses) => {
  return Object.entries(expenses.reduce((acc, expense) => {
    acc[expense.label] = (acc[expense.label] || 0) + expense.value;
    return acc;
  }, {})).map(([label, value]) => ({ label, value }));
};
```

Final Functionalities

1. Expense CRUD Operations: Users can create, update, delete, and view their expenses.
2. Expense Visualization: A pie chart is generated to show a breakdown of expenses by category.
3. Real-time Updates: Expenses are updated in real time, and users are notified via email about their expenses.
4. Responsive UI: The application’s interface adapts to various screen sizes and devices.

Challenges & Solutions

- Database Connection Handling: Ensuring stable connections with MongoDB Atlas using Mongoose required retries and proper error handling.
- Data Visualization: Grouping expenses by labels and visualizing them in real-time was a technical challenge that was solved using React's state management and MUI Charts.

Conclusion

This project successfully demonstrated the creation of a full-stack web application for tracking and visualizing expenses. The combination of Node.js, Express, MongoDB, and React provided a scalable, maintainable, and user-friendly solution. Future enhancements could include user authentication and more detailed analytics.

This report summarizes the complete workflow and technical stack used in the development of this expense tracker project.