**Project:** Cloud Native Expense Tracker with CI/CD.

It is a **Web Application** that helps users keep track of their daily expenses easily and efficiently. (MERN)

* **Front End:** React.js
* **Backend** built with Node.js and express.
* Need of **Database (Used MongoDB).**

**Motto of the project:**

The main goal of your project is to help users manage their money better by tracking expenses in real-time, providing a simple interface, and ensuring reliable, scalable deployment using modern cloud and DevOps technologies.

**Who Should Use Your Project?**

* **Individuals** who want to track personal expenses and manage budgets.
* **Small businesses or departments** that need to monitor spending and control cash flow.
* **Remote teams** who want to submit and track expenses digitally.
* Anyone who wants **financial transparency and control** over their daily spending.

**How Is It Useful for Real-Time Industry Purposes?**

* **Personal Finance Management:** Helps users understand where their money goes, avoid overspending, and plan budgets.
* **Business Expense Control:** Companies can monitor expenses, generate reports, and optimize spending.
* **Cloud-Native Deployment:** Using Kubernetes and AWS ensures high availability, scalability, and easy maintenance.
* **DevOps Automation:** CI/CD pipelines enable fast, reliable updates and testing, reducing downtime and errors.
* **Security:** Using secrets management and secure connections protects sensitive financial data.
* **Real-Time Monitoring:** Users see instant updates on their expenses, reflecting real-world usage and financial decisions.

**Questions:**

1. **What problem does your project solve?**

It helps users track and manage their expenses easily, preventing overspending and improving financial planning.

1. **Why did you choose AWS?**

For scalability, high availability, and ease of managing deployments in a cloud environment

1. **How does CI/CD improve your project?**

It automates testing and deployment, ensuring quick, reliable updates with minimal downtime.

1. **What real-world users benefit most?**

Individuals managing personal budgets, small businesses controlling expenses, and remote teams submitting expenses digitally.

1. **What future enhancements do you plan?**

Adding user authentication, budget alerts, detailed analytics, and mobile app integration.

**Why this project is useful?**

Many people struggle to manage their finances. Keeping all income and expenses in one place.

**Giving visual reports (graphs)** so users can quickly understand their spending habits. It’s cloud-native, meaning it can scale and run smoothly online. It includes CI/CD, so updates and new features can be added automatically without downtime.

**Technologies Used & Why (in simple terms):**

1. **React.js (Frontend)**

React helps build **dynamic, fast, and interactive** web pages. When users click or enter data, it updates the screen quickly without reloading the whole page.

**Layman Example:**

Like how Instagram or Gmail updates content without refreshing the whole page.

**2. Express.js + Node.js (Backend)**

It helps connect the frontend to the database. It handles requests like:

Add a new expense

Fetch all expenses

Add income

**Layman Example:**

Think of it as the waiter between the user (frontend) and the kitchen (database). It carries messages back and forth.

**3. MongoDB (Database)**

MongoDB is great for storing flexible data like:

**Layman Example:**

It’s like a digital notebook where you write down each transaction — income or expense.

**4. Docker (or Kubernetes) (Containerization/Deployment)**

Docker packages the whole app so it can run the same way on any computer, server, or cloud platform.

Kubernetes (if used) helps in scaling and managing multiple parts of the app (like auto-restarting if it fails).

**Layman Example:**

Think of Docker as putting your app in a sealed lunchbox wherever you take it, it stays the same.

Kubernetes is like a robotic lunch delivery system that makes sure everyone gets their lunch and replaces it if spilled.

**5. CI/CD (Continuous Integration / Continuous Deployment):**

It helps you automate testing and deployment. When you push new code, it automatically:

Runs tests

Builds the app

Deploys it to the cloud

**Layman Example:**

It’s like a vending machine for code updates. You push a button (code), and it builds, tests, and serves it live — no manual steps.

**Why Did I Use These Technologies (in 1 Line Each)?**

\* React.js – for fast, interactive, modern frontend

\* Express.js – for simple, powerful server-side logic

\* MongoDB – for flexible, schema-less data storage

\* Docker/Kubernetes – to make app easy to deploy and scale

\* CI/CD – to automate deployment and reduce human errors

**Docker:**

Think of it like a shipping container for your app. Just like a shipping container can hold furniture, food, or electronics, Docker holds your app and everything it needs to run — code, libraries, tools, system settings. It makes sure your app works the same way on any computer or server, no matter where it runs.

**CI/CD:**

Let’s say I have an application. If I want to run it on another server without automation, I’d have to set everything up manually from scratch. But with **CI/CD and Docker**, the process becomes automatic: CI/CD builds the app, creates a **Docker image** (a secure, self-contained package), and then **automatically deploys** it to the server — without manual steps.

**Docker** = A neatly packed suitcase with everything your app needs.

**CI/CD** = A robot that packs the suitcase, checks it, and ships it to the destination automatically.

**Pending Task:**

* 1. Testing report of CI/CD
  2. Validation in the income add date and email login
  3. Kubernetes
  4. Alert when u cross the amount of spent than earn.

**Imagination Task:**

1. Using Gemini API – Chat bot assistant.
2. Based on location currency change.
3. Language Change