RAJALAKSHMI ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

CS23332 DATABASE MANAGEMENT

SYSTEMS LAB

EXPENSE TRACKER SYSTEM

A MINI PROJECT REPORT

Submitted By

KAVIYAM M S	**231501075**
MAHAASHREE ANBURAJ	**231501091**
MANOPRASAD V	**231501095**

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING IN

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS) THANDALAM

CHENNAI-602105

2024 - 2025

BONAFIDE CERTIFICATE

Certified that this project report "**EXPENSE TRACKER SYSTEM**"

is the Bonafide work of **"KAVIYAM(231501075),MAHAASHREE ANBURAJ(231501091),MANOPRASAD V(231501095)"** who carried out the project work under my supervision.

Submitted for the Practical Examination held on ____________________

SIGNATURE

Priya Thiyagarajan

Assistant Professor (SS)

AIML,

Rajalakshmi Engineering College

(Autonomous),

Thandalam, Chennai - 602 105

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

An expense tracker is a digital tool designed to help users efficiently manage and monitor their financial activities. This project focuses on developing a user-friendly application that records, categorizes, and visualizes daily expenses.

Leveraging MongoDB as the backend database, the system ensures secure and flexible storage of financial data. Users can add, update, or delete expenses while viewing detailed summaries and insights through graphs and charts. The app supports real-time data management, enabling users to track spending patterns and set budgetary goals. With features like advanced analytics, categorization, and multi-device synchronization, this expense tracker promotes financial discipline and provides a scalable solution for personal and organizational finance management.

TABLE OF CONTENTS

INTRODUCTION

The *Expense Tracker* is a modern web application designed to simplify personal financial management by enabling users to track and analyze their income and expenses efficiently. Built using the MERN (MongoDB, Express.js, React.js, Node.js) stack, this project offers a seamless and interactive interface for monitoring financial transactions.

With features such as *adding, deleting, and updating income and expense records*, the application provides users with complete control over their financial data. Users can visualize their transaction history through interactive charts and obtain insights into their total income, expenses, and balance. The application's intuitive design ensures a user-friendly experience, making it an ideal tool for managing personal finances.

Key Features

1. *Add Transactions*

   Users can add detailed records of incomes or expenses, including descriptions and categories.

2. *Update Transactions*

   Existing transactions can be edited to reflect changes accurately.

3. *Delete Transactions*

   Users can easily remove any transaction from their record.

4. *Visual Data Representation*

   A dynamic chart displays the income and expense trends, offering an at-a-glance view of financial performance over time.

5. *Summary Section*

The dashboard provides a concise summary of:

- Total Income

- Total Expense

- Total Balance

6. *Transaction History*

A categorized list of recent transactions allows users to review and manage their financial activity effortlessly.

SOFTWARE DESCRIPTION

Visual studio Code

Visual Studio Code combines the simplicity of a source code editor with powerful developer tooling, like IntelliSense code completion and debugging. First and foremost, it is an editor that gets out of your way. The delightfully frictionless edit-build-debug cycle means less time fiddling with your environment, and more time executing on your ideas.

2.2 LANGUAGES

2.2.1 HTML

- **Hypertext**: text (often with embeds such as images, too) that is organized in order to connect related items
- **Markup**: a style guide for typesetting anything to be printed in hardcopy or soft copy format
- **Language**: a language that a computer system understands and uses to interpret commands.
- HTML determines the structure of web pages. This structure alone is not enough to make a web page look good and interactive. So you'll use assisted technologies such as CSS and JavaScript to make your HTML beautiful and add interactivity, respectively

2.2.2 CSS

CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, layout designs,variations in display for different devices and screen sizes as well as a variety of other effects.

2.2.3 JAVASCRIPT

JavaScript is a versatile, high-level programming language widely used for web development to create dynamic and interactive user experiences. Initially designed for client-side scripting, it now powers both front-end and back-end development, thanks to platforms like Node.js. JavaScript’s dynamic typing, lightweight nature, and event-driven architecture make it flexible and efficient for building web applications. It works seamlessly with HTML and CSS to enhance user interfaces with features like animations, interactive forms, and real-time updates. Beyond the web, JavaScript is also used for mobile app development through frameworks like React Native and Ionic, demonstrating its versatility. Its cross-platform compatibility, prototypal inheritance model, and growing ecosystem of tools and libraries have established JavaScript as a cornerstone of modern software development.

2.2.4 MONGODB

MongoDB is a popular NoSQL database known for its flexibility, scalability, and performance. It uses a document-based data model, storing data in JSON-like formats, making it ideal for handling diverse and unstructured data. With features like indexing, aggregation, and horizontal scaling, MongoDB excels in managing large datasets for modern applications. Its compatibility with various programming languages and ease of integration with web and mobile apps have made it a preferred choice for developers building scalable and real-time applications.

```
}
```

PROGRAM CODE

FRONTEND

Expense

```
const ExpenseSchema = require("../models/ExpenseModel")

exports.addExpense = async (req, res) => {
    const {title, amount, category, description, date}  = req.body


    const income = ExpenseSchema({
        title,
        amount,
        category,
        description,
        date
    })


    try {
        //validations
        if(!title || !category || !description || !date){
            return res.status(400).json({message: 'All fields are required!'})
        }
        if(amount <= 0 || !amount === 'number'){
            return res.status(400).json({message: 'Amount must be a positive number!'})
        }
        await income.save()
        res.status(200).json({message: 'Expense Added'})
    } catch (error) {
        res.status(500).json({message: 'Server Error'})
    }

    console.log(income)
}

exports.getExpense = async (req, res) =>{
    try {
        const incomes = await ExpenseSchema.find().sort({createdAt: -1})
        res.status(200).json(incomes)
    } catch (error) {
        res.status(500).json({message: 'Server Error'})
    }
}

exports.deleteExpense = async (req, res) =>{
    const {id} = req.params;
    ExpenseSchema.findByIdAndDelete(id)
        .then((income) =>{
            res.status(200).json({message: 'Expense Deleted'})
        })
        .catch((err) =>{
            res.status(500).json({message: 'Server Error'})
        })
}
```

Income

```
const IncomeSchema= require("../models/IncomeModel")



exports.addIncome = async (req, res) => {
    const {title, amount, category, description, date}  = req.body

    const income = IncomeSchema({
        title,
        amount,
        category,
        description,
        date
    })

    try {
        //validations
        if(!title || !category || !description || !date){
            return res.status(400).json({message: 'All fields are required!'})
        }
        if(amount <= 0 || !amount === 'number'){
            return res.status(400).json({message: 'Amount must be a positive number!'})
        }
        await income.save()
        res.status(200).json({message: 'Income Added'})
    } catch (error) {
        res.status(500).json({message: 'Server Error'})
    }

    console.log(income)
}

exports.getIncomes = async (req, res) =>{
    try {
        const incomes = await IncomeSchema.find().sort({createdAt: -1})
        res.status(200).json(incomes)
    } catch (error) {
        res.status(500).json({message: 'Server Error'})
    }
}

exports.deleteIncome = async (req, res) =>{
    const {id} = req.params;
    IncomeSchema.findByIdAndDelete(id)
```

```
        .then((income) =>{
            res.status(200).json({message: 'Income Deleted'})
        })
        .catch((err) =>{
            res.status(500).json({message: 'Server Error'})
        })
}
```

Db

```
const mongoose = require('mongoose');


const db = async () => {
    try {
        await mongoose.connect(process.env.MONGO_URI, {
            useNewUrlParser: true,
            useUnifiedTopology: true,
        });
        console.log('Database connected successfully');
    } catch (error) {
        console.log('DB Connection Error:', error.message);
    }
};


module.exports = { db };
```

Models

ExpenseModel

```
const mongoose = require('mongoose');



const ExpenseSchema = new mongoose.Schema({
    title: {
        type: String,
        required: true,
        trim: true,
        maxLength: 50
    },
    amount: {
        type: Number,
        required: true,
        maxLength: 20,
        trim: true
    },
    type: {
        type: String,
        default:"expense"
    },
    date: {
        type: Date,
        required: true,
        trim: true
    },
    category: {
        type: String,
        required: true,
        trim: true
    },
    description: {
        type: String,
        required: true,
        maxLength: 20,
        trim: true
    },
}, {timestamps: true})

module.exports = mongoose.model('Expense', ExpenseSchema)
```

IncomeModel

```
const mongoose = require('mongoose');



const IncomeSchema = new mongoose.Schema({
    title: {
        type: String,
        required: true,
        trim: true,
        maxLength: 50
    },
    amount: {
        type: Number,
        required: true,
        maxLength: 20,
        trim: true
    },
    type: {
        type: String,
        default:"income"
    },
    date: {
        type: Date,
        required: true,
        trim: true
    },
    category: {
        type: String,
        required: true,
        trim: true
    },
    description: {
        type: String,
        required: true,
        maxLength: 20,
        trim: true
    },
}, {timestamps: true})

module.exports = mongoose.model('Income', IncomeSchema)
```

App

```
const express = require('express')
const cors = require('cors');
const { db } = require('./db/db');
const {readdirSync} = require('fs')
const app = express()

require('dotenv').config()

const PORT = process.env.PORT
//middlewares
app.use(express.json())
app.use(cors())
```

```
//routes

readdirSync('./routes').map((route) => app.use('/api/v1', require('./routes/' + route)))


const server = () => {

    db()

    app.listen(PORT, () => {

        console.log('listening to port:', PORT)

    })

}


server()
```

FRONTEND

Index

```
<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="utf-8" />

    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />

    <meta name="viewport" content="width=device-width, initial-scale=1" />

    <meta name="theme-color" content="#000000" />

    <meta

      name="description"

      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Nunito:wght@400;500;600;700;800&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.3.0/css/all.min.css"
        integrity="sha512-SzlrxWUlpfuzQ+pcUCosxcglQRNAq/DZjVsC0lE40xsADsfeQoEypE+enwcOiGjk/bSuGGKHEyjSoQ1zVisanQ=="
        crossorigin="anonymous" referrerpolicy="no-referrer" />

    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the public folder during the build.
      Only files inside the public folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
```

```
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running npm run build.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run npm start or yarn start.
      To create a production bundle, use npm run build or yarn build.
    -->
  </body>
</html>
```

Manifest

```
{
  "short_name": "React App",
  "name": "Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```

Styles

-->GlobalStyles.js

-->Layout.js

GlobalStyles

```
import {createGlobalStyle} from 'styled-components'


export const GlobalStyle = createGlobalStyle`
   *{
       margin: 0;
       padding: 0;
       box-sizing: border-box;
       list-style: none;
   }


   :root{
       --primary-color: #222260;
       --primary-color2: 'color: rgba(34, 34, 96, .6)';
       --primary-color3: 'color: rgba(34, 34, 96, .4)';
       --color-green: #42AD00;
       --color-grey: #aaa;
       --color-accent: #F56692;
       --color-delete: #FF0000;
   }
```

```
body{
    font-family: 'Nunito', sans-serif;
    font-size: clamp(1rem, 1.5vw, 1.2rem);
    overflow: hidden;
    color: rgba(34, 34, 96, .6);
}

h1, h2, h3, h4, h5, h6{
    color: var(--primary-color);
}

.error{
    color: red;
    animation: shake 0.5s ease-in-out;
    @keyframes shake {
        0%{
            transform: translateX(0);
        }
        25%{
            transform: translateX(10px);
        }
        50%{
            transform: translateX(-10px);
            }
            75%{
                transform: translateX(10px);
            }
            100%{
                transform: translateX(0);
            }
        }
    }
`;
```

Layout

```
import styled from "styled-components";


export const MainLayout = styled.div`
    padding: 2rem;
    height: 100%;
    display: flex;
    gap: 2rem;
`;
export const InnerLayout = styled.div`
    padding: 2rem 1.5rem;
    width: 100%;
`
```

Utils

-->dateFormat.js

-->Icons.js

-->menuItems.js

-->useWindowsize.js

DateFormat

```
import moment from 'moment'



export const dateFormat = (date) =>{
    return moment(date).format('DD/MM/YYYY')
}
```

Icons

```
export const dashboard = <i className="fa-solid fa-chart-line"></i>
export const transactions = <i className="fa-solid fa-credit-card"></i>
export const categories = <i className="fa-solid fa-tags"></i>
export const accounts = <i className="fa-solid fa-wallet"></i>
export const settings = <i className="fa-solid fa-cog"></i>
export const logout = <i className="fa-solid fa-sign-out"></i>
export const trend = <i className="fa-solid fa-money-bill-trend-up"></i>
export const expenses = <i className="fa-solid fa-money-bill-transfer"></i>
export const money = <i className="fa-solid fa-money-bill"></i>

export const freelance = <i className ="fa-solid fa-earth-americas"></i>

export const stocks = <i className="fa-solid fa-arrow-trend-up"></i>

export const bitcoin = <i className="fa-brands fa-bitcoin"></i>

export const piggy = <i className="fa-solid fa-piggy-bank"></i>

export const yt = <i className="fa-brands fa-youtube"></i>

export const card = <i className="fa-brands fa-cc-visa"></i>

export const users = <i className="fa-solid fa-users-between-lines"></i>

export const dollar = <i className="fa-solid fa-dollar-sign"></i>

export const calender = <i className="fa-solid fa-calendar"></i>

export const comment = <i className="fa-solid fa-comment"></i>

export const plus = <i className="fa-solid fa-plus"></i>

export const trash = <i className="fa-solid fa-trash"></i>

export const signout = <i className="fa-solid fa-right-from-bracket"></i>

export const takeaway = <i className="fa-solid fa-utensils"></i>

export const clothing = <i className="fa-solid fa-shirt"></i>

export const book = <i className="fa-solid fa-book-open"></i>

export const food = <i className="fa-solid fa-bowl-food"></i>

export const medical = <i className="fa-solid fa-briefcase-medical"></i>

export const tv = <i className="fa-solid fa-tv"></i>

export const circle = <i className="fa-solid fa-circle-dot"></i>
```

MenuItems

```
import {dashboard, expenses, transactions, trend} from '../utils/Icons'


export const menuItems = [
  {
    id: 1,
    title: 'Dashboard',
    icon: dashboard,
    link: '/dashboard'
  },
  {
    id: 2,
    title: "View Transactions",
    icon: transactions,
    link: "/dashboard",
  },
  {
    id: 3,
    title: "Incomes",
    icon: trend,
    link: "/dashboard",
  },
  {
    id: 4,
    title: "Expenses",
    icon: expenses,
    link: "/dashboard",
  },
]
```

useWindowsize

```
import { useEffect, useState } from "react"
export const useWindowSize = () =>{
    const [size, setSize] = useState([window.innerWidth, window.innerHeight])

    useEffect(() => {
        const updateSize = () => {
            setSize([window.innerWidth, window.innerHeight])
        }
        window.addEventListener('resize', updateSize)

        return () => window.removeEventListener('resize', updateSize)
    }, [])

    return {
        width: size[0],
        height: size[1]
    }
}
```

App

```
import React, {useState, useMemo} from 'react'
import styled from "styled-components";
import bg from './img/bg.png'
import {MainLayout} from './styles/Layouts'
import Orb from './Components/Orb/Orb'
import Navigation from './Components/Navigation/Navigation'
import Dashboard from './Components/Dashboard/Dashboard';
import Income from './Components/Income/Income'
import Expenses from './Components/Expenses/Expenses';
import { useGlobalContext } from './context/globalContext';

function App() {
  const [active, setActive] = useState(1)

  const global = useGlobalContext()
  console.log(global);

  const displayData = () => {
    switch(active){
      case 1:
```

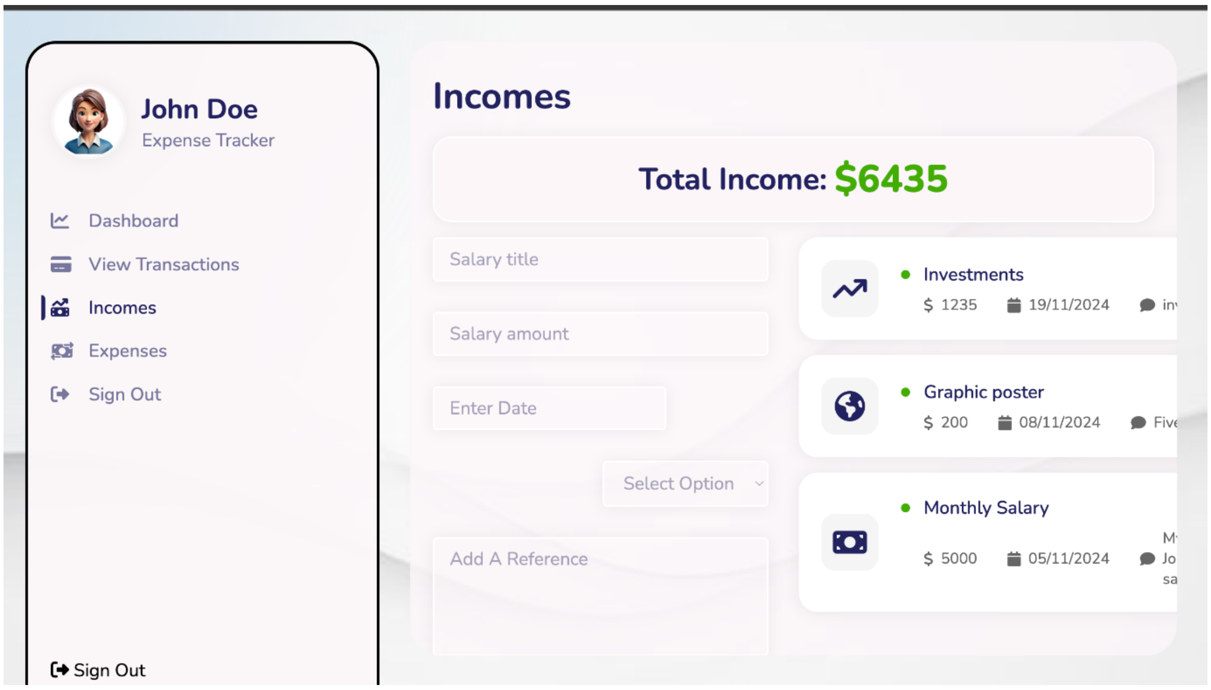
```
      return <Dashboard />
    case 2:
      return <Dashboard />
    case 3:
      return <Income />
    case 4:
      return <Expenses />
    default:
      return <Dashboard />
  }
}

const orbMemo = useMemo(() => {
  return <Orb />
},[])

return (
  <AppStyled bg={bg} className="App">
    {orbMemo}
    <MainLayout>
      <Navigation active={active} setActive={setActive} />
      <main>
        {displayData()}
      </main>
      </MainLayout>
    </AppStyled>
  );
}

const AppStyled = styled.div`
  height: 100vh;
  background-image: url(${props => props.bg});
  position: relative;
  main{
    flex: 1;
    background: rgba(252, 246, 249, 0.78);
    border: 3px solid #FFFFFF;
    backdrop-filter: blur(4.5px);
    border-radius: 32px;
    overflow-x: hidden;
    &::-webkit-scrollbar{
      width: 0;
    }
  }
`;

export default App;
```

Index

```
import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

import { GlobalProvider } from './context/globalContext';

import { GlobalStyle } from './styles/GlobalStyle';


const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <React.StrictMode>

    <GlobalStyle />

    <GlobalProvider>

      <App />

    </GlobalProvider>

  </React.StrictMode>

);
```

HOMEPAGE

INCOMES



John Doe
Expense Tracker

Dashboard
View Transactions
Incomes
Expenses
Sign Out

Sign Out

Incomes

Total Income: $6435

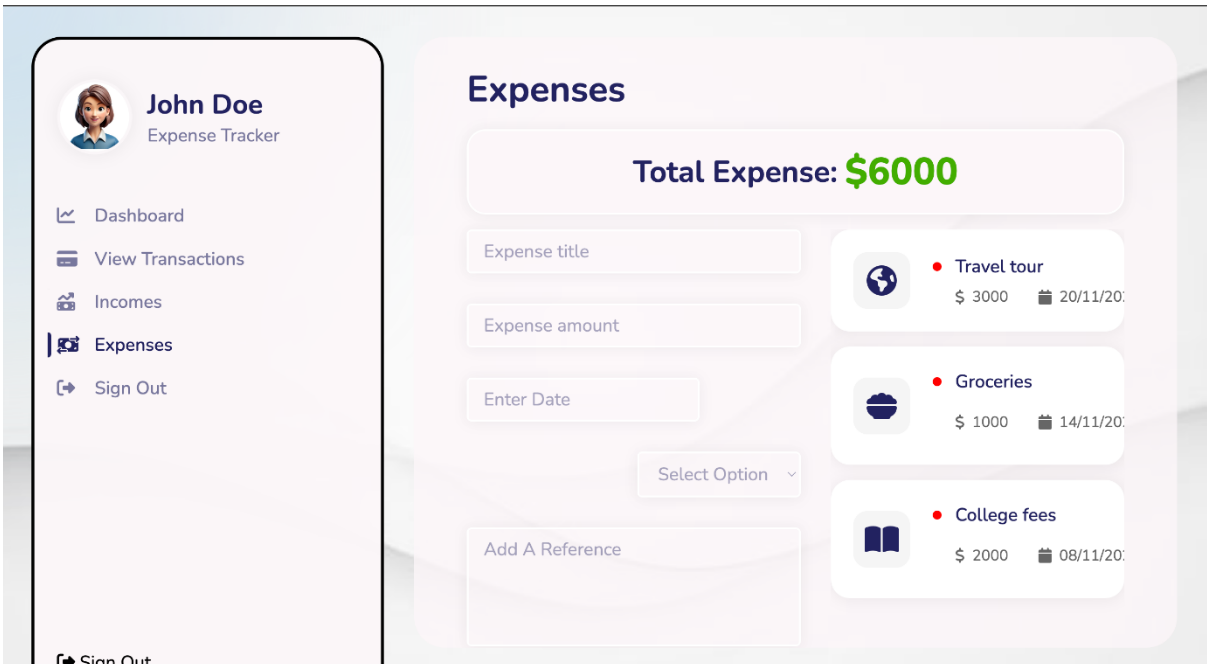
Salary title
Salary amount
Enter Date
Select Option
Add A Reference

Investments
$ 1235 19/11/2024 in

Graphic poster
$ 200 08/11/2024 Five

Monthly Salary
$ 5000 05/11/2024 M Jo sa

EXPENSES



John Doe
Expense Tracker

Dashboard
View Transactions
Incomes
Expenses
Sign Out

Sign Out

Expenses

Total Expense: $6000

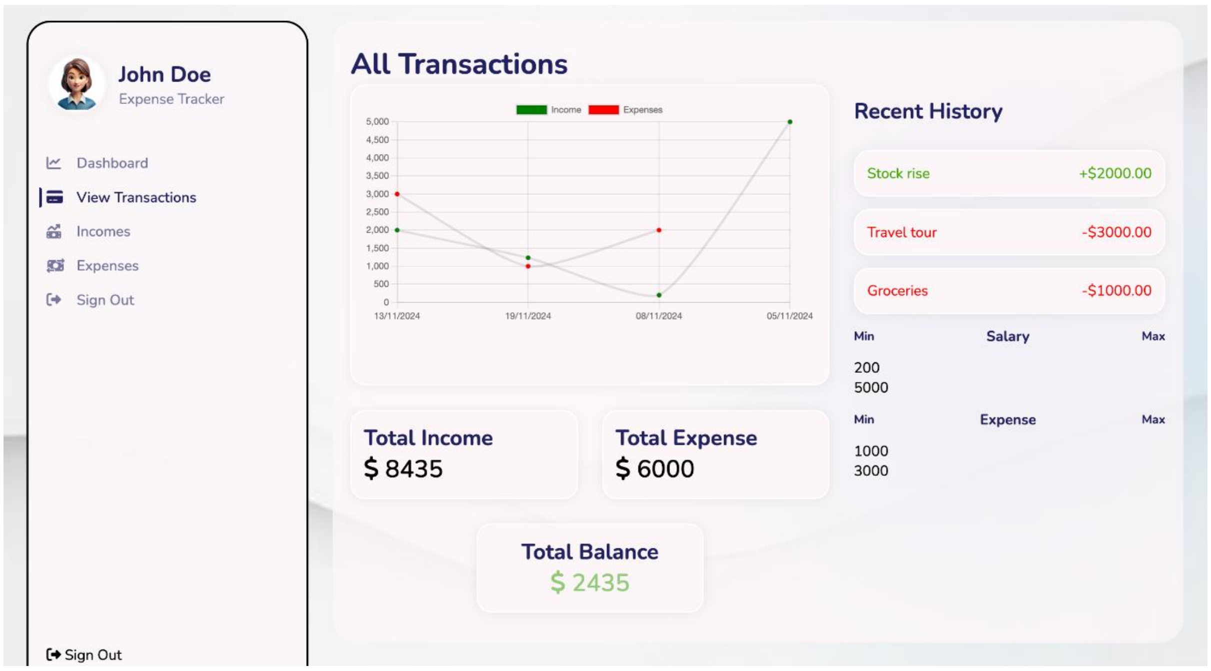
Expense title
Expense amount
Enter Date
Select Option
Add A Reference

Travel tour
$ 3000 20/11/20

Groceries
$ 1000 14/11/20

College fees
$ 2000 08/11/20

ALL TRANSACTIONS



John Doe
Expense Tracker

All Transactions

Recent History

Stock rise	+$2000.00
Travel tour	-$3000.00
Groceries	-$1000.00

Min Salary Max
200
5000

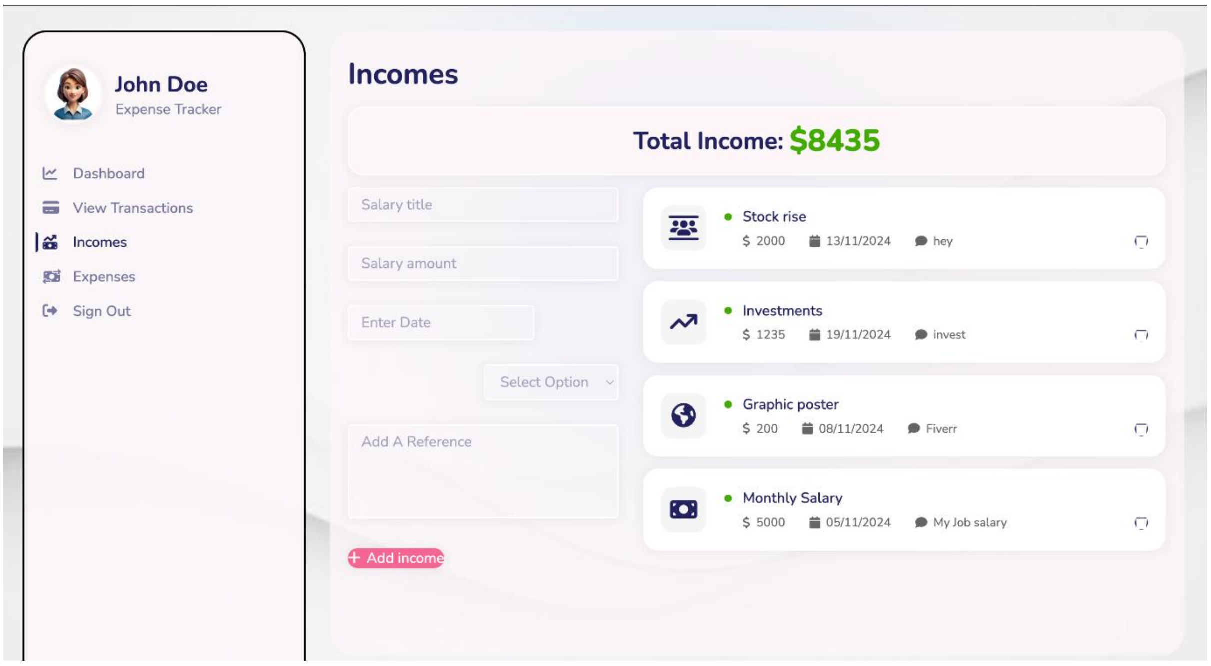
Min Expense Max
1000
3000

Total Income
$ 8435

Total Expense
$ 6000

Total Balance
$ 2435

INCOMES



John Doe
Expense Tracker

Incomes

Total Income: $8435

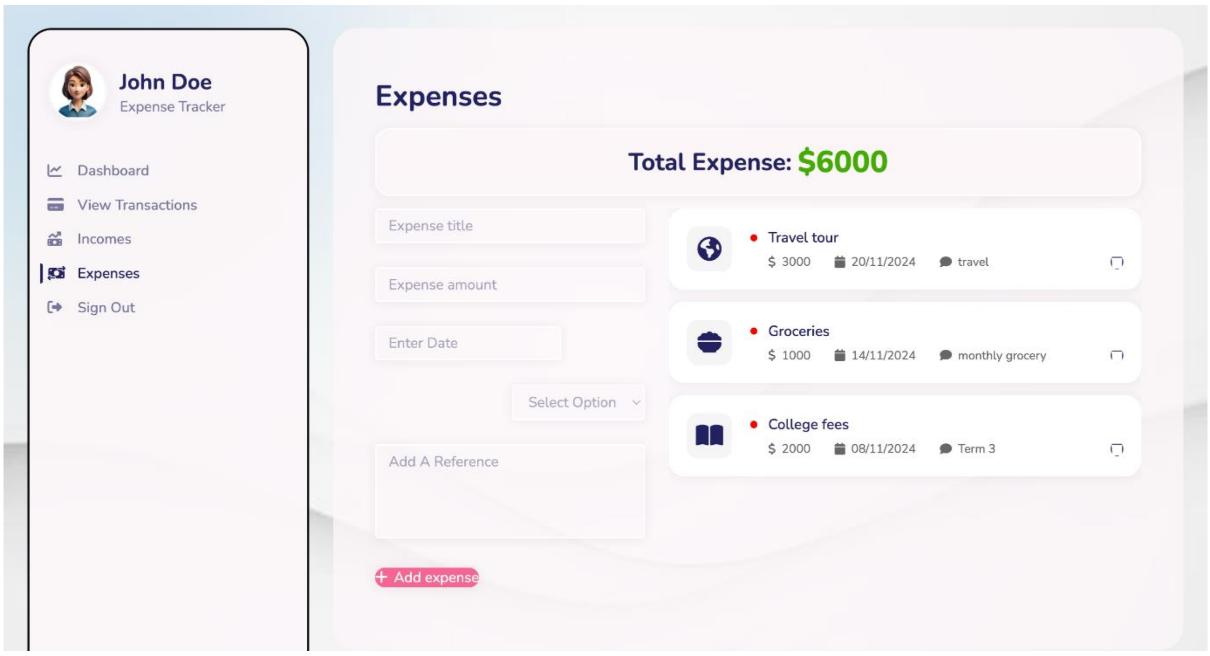
Stock rise
$ 2000 13/11/2024 hey

Investments
$ 1235 19/11/2024 invest

Graphic poster
$ 200 08/11/2024 Fiverr

Monthly Salary
$ 5000 05/11/2024 My Job salary

EXPENSES



Expenses

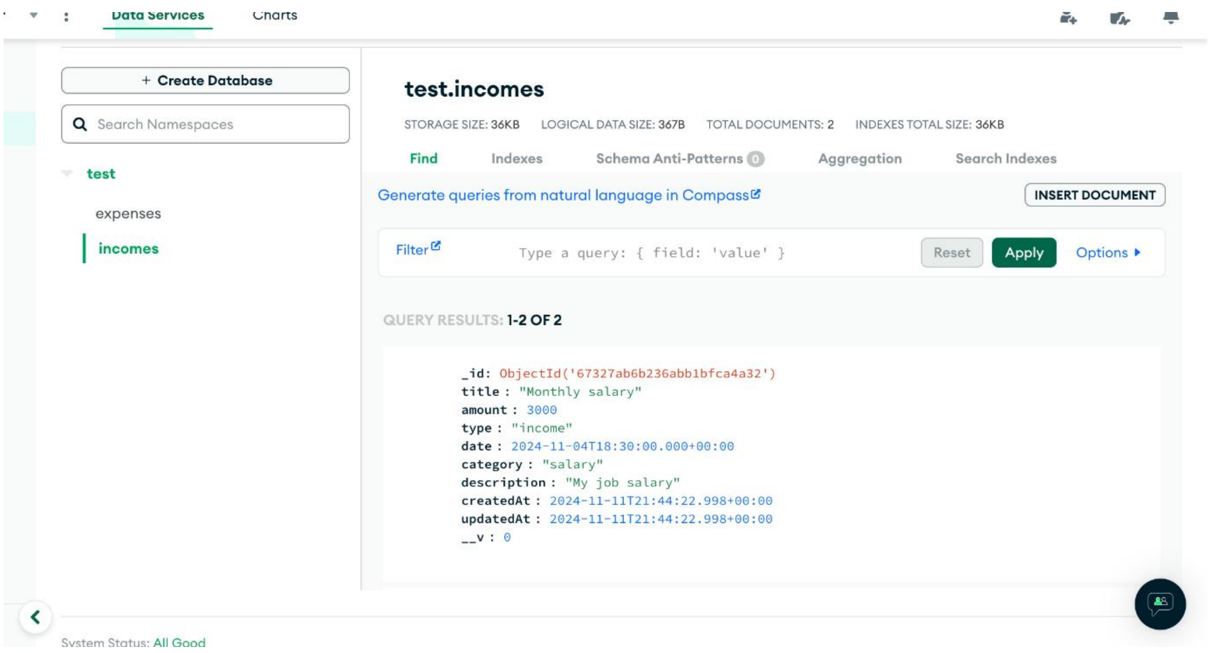
Total Expense: $6000

Travel tour
$ 3000 20/11/2024 travel

Groceries
$ 1000 14/11/2024 monthly grocery

College fees
$ 2000 08/11/2024 Term 3

TEST INCOMES



test.incomes

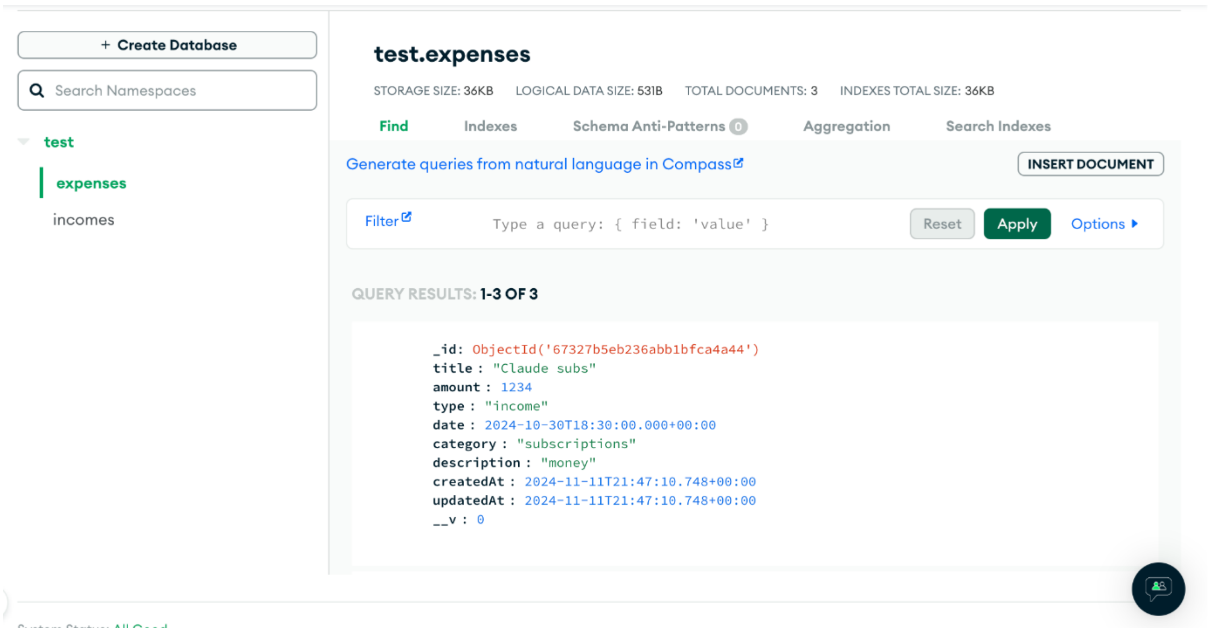
STORAGE SIZE: 36KB LOGICAL DATA SIZE: 367B TOTAL DOCUMENTS: 2 INDEXES TOTAL SIZE: 36KB

QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('67327ab6b236abb1bfca4a32')
title : "Monthly salary"
amount : 3000
type : "income"
date : 2024-11-04T18:30:00.000+00:00
category : "salary"
description : "My job salary"
createdAt : 2024-11-11T21:44:22.998+00:00
updatedAt : 2024-11-11T21:44:22.998+00:00
__v : 0
```

System Status: All Good

TEST EXPENSES



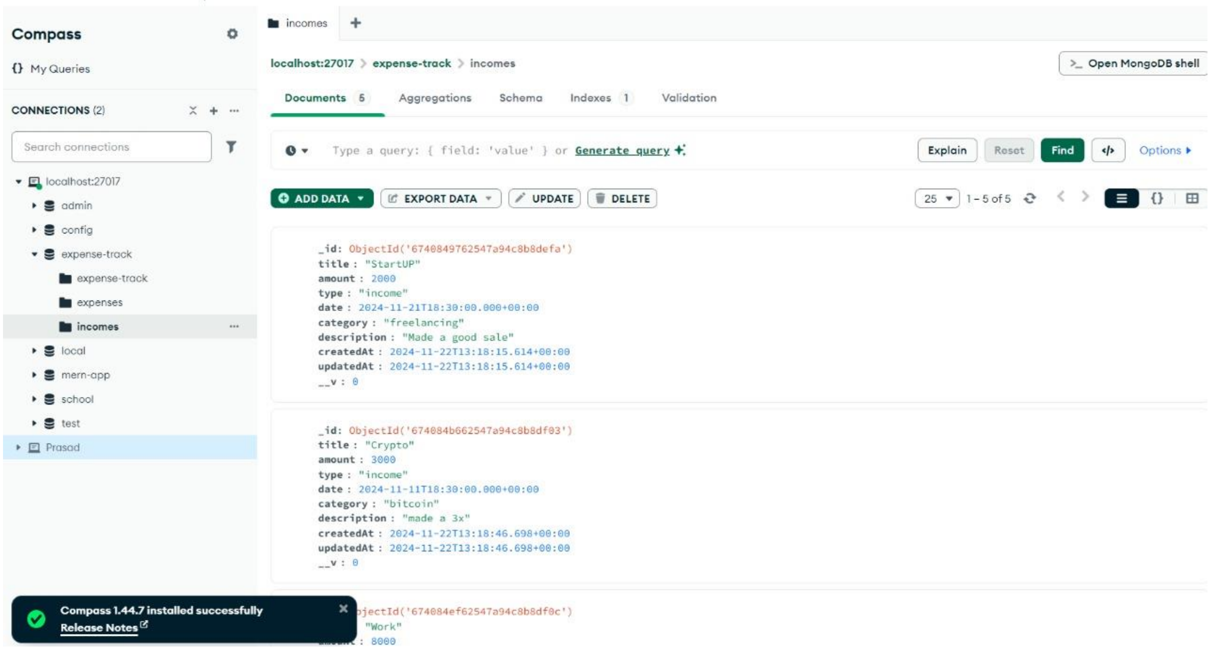
test.expenses

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 531B TOTAL DOCUMENTS: 3 INDEXES TOTAL SIZE: 36KB

Type a query: { field: 'value' }

QUERY RESULTS: 1-3 OF 3

```
_id: ObjectId('67327b5eb236abb1bfca4a44')
title : "Claude subs"
amount : 1234
type : "income"
date : 2024-10-30T18:30:00.000+00:00
category : "subscriptions"
description : "money"
createdAt : 2024-11-11T21:47:10.748+00:00
updatedAt : 2024-11-11T21:47:10.748+00:00
__v : 0
```



localhost:27017 > expense-track > incomes

Type a query: { field: 'value' } or Generate query

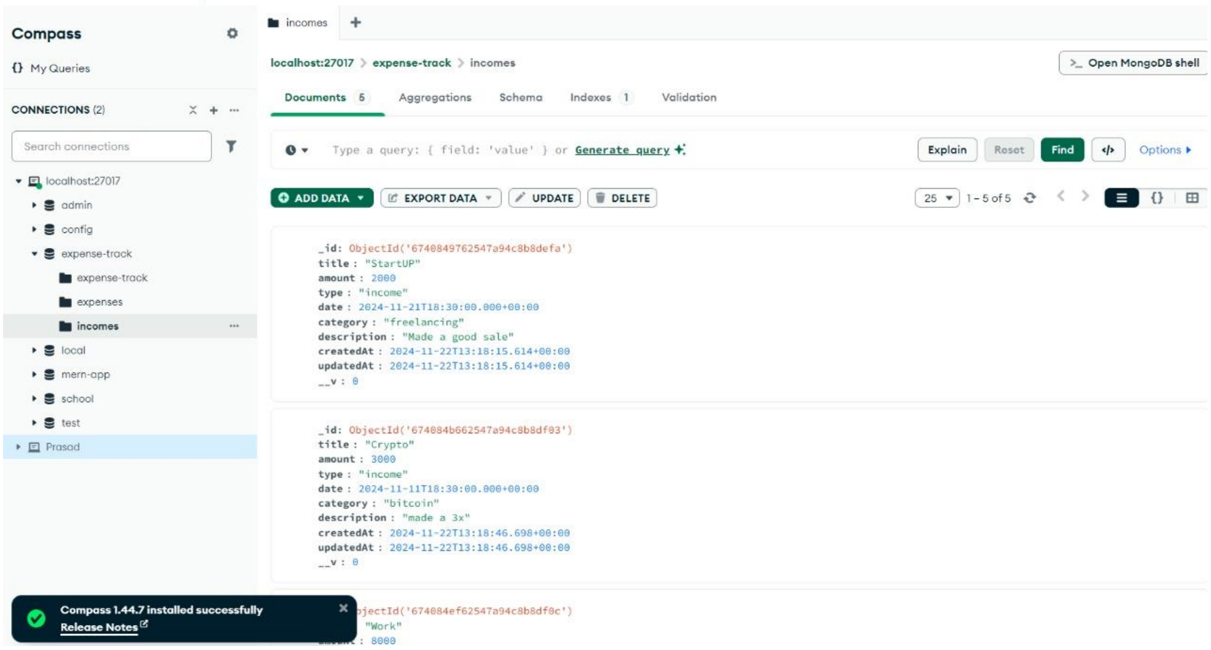
1 – 5 of 5

```
_id: ObjectId('6740849762547a94c8b8defa')
title : "StartUP"
amount : 2000
type : "income"
date : 2024-11-21T18:30:00.000+00:00
category : "freelancing"
description : "Made a good sale"
createdAt : 2024-11-22T13:18:15.614+00:00
updatedAt : 2024-11-22T13:18:15.614+00:00
__v : 0
```

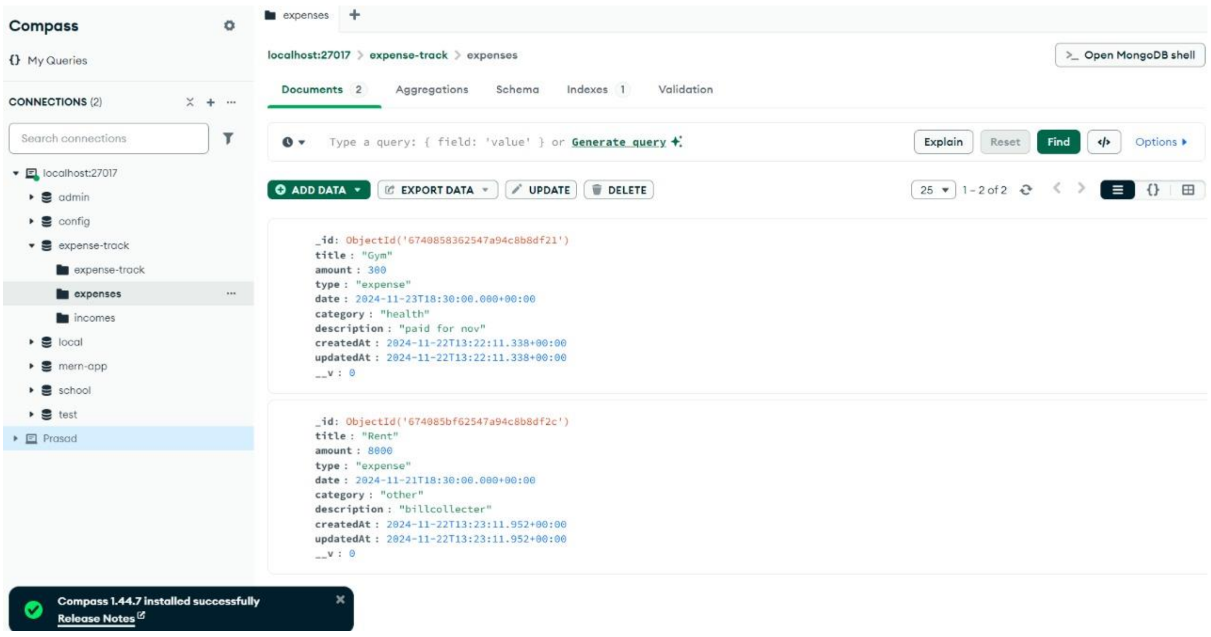
```
_id: ObjectId('674084b662547a94c8b8df03')
title : "Crypto"
amount : 3000
type : "income"
date : 2024-11-11T18:30:00.000+00:00
category : "bitcoin"
description : "made a 3x"
createdAt : 2024-11-22T13:18:46.698+00:00
updatedAt : 2024-11-22T13:18:46.698+00:00
__v : 0
```

```
bjectId('674084ef62547a94c8b8df0c')
 "Work"
 : 8000
```

Compass 1.44.7 installed successfully
Release Notes



```
_id: ObjectId('6740849762547a94c8b8defa')
title : "StartUP"
amount : 2000
type : "income"
date : 2024-11-21T18:30:00.000+00:00
category : "freelancing"
description : "Made a good sale"
createdAt : 2024-11-22T13:18:15.614+00:00
updatedAt : 2024-11-22T13:18:15.614+00:00
__v : 0

_id: ObjectId('674084b662547a94c8b8df03')
title : "Crypto"
amount : 3000
type : "income"
date : 2024-11-11T18:30:00.000+00:00
category : "bitcoin"
description : "made a 3x"
createdAt : 2024-11-22T13:18:46.698+00:00
updatedAt : 2024-11-22T13:18:46.698+00:00
__v : 0

bjectId('674084ef62547a94c8b8df0c')
"Work"
: 8000
```



```
_id: ObjectId('6740858362547a94c8b8df21')
title : "Gym"
amount : 300
type : "expense"
date : 2024-11-23T18:30:00.000+00:00
category : "health"
description : "paid for nov"
createdAt : 2024-11-22T13:22:11.338+00:00
updatedAt : 2024-11-22T13:22:11.338+00:00
__v : 0

_id: ObjectId('674085bf62547a94c8b8df2c')
title : "Rent"
amount : 8000
type : "expense"
date : 2024-11-21T18:30:00.000+00:00
category : "other"
description : "billcollecter"
createdAt : 2024-11-22T13:23:11.952+00:00
updatedAt : 2024-11-22T13:23:11.952+00:00
__v : 0
```

Future scope of the project

1. Integration with Third-Party APIs
   - Incorporate APIs for bank synchronization to automatically fetch and categorize transactions.
   - Integration with platforms like Google Pay, PayPal, or Stripe for real-time expense tracking.
2. Advanced Analytics and Insights
   - Implement predictive analytics using machine learning to forecast future expenses based on past data.
   - Provide monthly/quarterly financial summaries with visual insights like pie charts, graphs, and trends.
3. Multi-User Collaboration
   - Enable shared accounts for families, teams, or organizations to track group expenses.
   - Include features for assigning expense categories and roles for better collaboration.
4. Mobile and Cross-Platform Applications
   - Develop dedicated mobile apps for Android and iOS with real-time data sync using MongoDB Atlas.
   - Offer a progressive web app (PWA) to ensure cross-platform accessibility.
5. Budgeting and Savings Goals
   - Add features to set, track, and notify users about personalized budgets for different categories.
   - Introduce savings goal tracking with automated suggestions to cut down unnecessary expenses.
6. AI-Powered Expense Categorization

   - Use AI/ML algorithms to auto-categorize expenses based on descriptions, merchants, or patterns.
   - Provide recommendations for categorization corrections.

7. Multi-Currency and Localization Support
   - Support multiple currencies for users managing international transactions.
   - Add language localization features for broader usability.

8. Gamification and Rewards
   - Introduce gamified elements, such as milestones and badges, to encourage saving habits.
   - Partner with financial institutions or retailers to offer discounts or rewards for efficient expense management.

9. Secure Data Handling
   - Strengthen security using end-to-end encryption for sensitive data.
   - Implement two-factor authentication (2FA) for account security.

10. Audit Trails and Reports

- Provide detailed audit trails for financial reviews and compliance.
- Export reports in formats like PDF, Excel, or CSV for external use.

11. Cloud Integration with MongoDB Atlas

- Use MongoDB Atlas for seamless cloud-based data management, ensuring high availability and scalability.
- Enable real-time synchronization for data accessed across devices.

12. Integration with Tax Tools

- Introduce features to categorize expenses for tax deductions automatically.
- Integrate with tax software to simplify tax filing.

By implementing these future enhancements, the expense tracker can evolve into a comprehensive financial management tool, catering to diverse user needs while offering seamless, intelligent, and secure financial insights.

Implementation

Detailed Design of Implementation This phase of the systems development life cycle refines hardware and software specifications, establishes programming plans, trains users and implements extensive testing procedures, to evaluate design and operating specifications and/or provide the basis for further modification.

Technical Design

This activity builds upon specifications produced during new system design, addingdetailed technical specifications and documentation.

Test Specifications and Planning

This activity prepares detailed test specifications for individual modules and programs, job streams, subsystems, and for the system as a whole.

Programming and Testing

This activity encompasses actual development, writing, and testing of program units or modules. 116

User Training

This activity encompasses writing user procedure manuals, preparation of user training materials, conducting training programs, and testing procedures.

Acceptance Test

A final procedural review to demonstrate a system and secure user approval before a system becomes operational.

Installation Phase

In this phase the new Computerized system is installed, the conversion to new procedures is fully implemented, and the potential of the new system is explored.

System Installation

The process of starting the actual use of a system and training user personnel in its operation.

Review Phase

This phase evaluates the successes and failures during a systems development project, and to measure the results of a new Computerized T

Development Recap

A review of a project immediately after completion to find successes and potentialproblems in future work.

Post-Implementation Review

A review, conducted after a new system has been in operation for some time, to evaluate actual system performance against original expectations and projections for cost-benefit improvements. Also identifies maintenance projects to enhance or improve the system.

CONCLUSION

The Expense Tracking App using MongoDB provides an efficient and scalable solution for managing personal and organizational finances. By leveraging MongoDB’s powerful document-based data model, the app ensures flexibility in handling diverse expense data while maintaining high performance.

The application simplifies financial management by offering features such as expense categorization, real-time tracking, and visual insights through charts and reports. MongoDB's capabilities, like indexing and aggregation pipelines, enable seamless querying and advanced data analysis, ensuring users can retrieve and interpret their financial information effortlessly.

This project emphasizes:

- User-Centric Design: Intuitive interfaces for effortless data input and retrieval.
- Reliability and Scalability: MongoDB ensures the application can handle large volumes of data as user needs grow.
- Data Security: Implementation of secure data handling practices guarantees the privacy and integrity of user information.

In conclusion, this expense tracker project demonstrates how database systems like MongoDB can be harnessed to build modern, efficient, and scalable applications. With potential future enhancements such as AI-driven insights, multi-user collaboration, and cloud integration, the app can evolve into a comprehensive financial management tool that caters to diverse user needs. This project not only enhances financial accountability but also aligns with the growing demand for smart and digital financial solutions.

RESEARCH AND REFERENCE

1.*MERN Stack Resources*

- [MongoDB Documentation] (https://www.mongodb.com/docs/)
- [Express.js Guide](https://expressjs.com/)
- [React.js Documentation](https://react.dev/)
- [Node.js Official Docs](https://nodejs.org/en/docs/)

2. *Data Visualization Libraries*

- [Chart.js Documentation](https://www.chartjs.org/docs/)
- [Recharts Guide](https://recharts.org/en-US/)

3. *UI/UX Design Guidelines*

- Don't Make Me Think by Steve Krug (book reference for intuitive design principles)
- [Material Design Guidelines](https://material.io/design/)

4. *Financial Management Tools*

- Analysis of popular expense tracking applications like Mint, YNAB (You Need A Budget), and PocketGuard for feature comparison.

5. *Security Guidelines*

- OWASP Top Ten Security Risks for Web Applications
  ([OWASP Website](https://owasp.org/))
- JSON Web Tokens (JWT) for secure user authentication
  ([JWT.io](https://jwt.io/))

6. *Project Management and Development*

   - Agile development principles and version control using Git and GitHub.