FINSIGHT : AUTOMATED EXPENSES RECORDING AND MONTITORING SYSTEM

A Project Report for Industrial Training and Internship

submitted by

SUBHRA KOLAY

CHANDRIMA RAY

AVIK BERA

AKASH MAITY

AYAN DAS

BRISTI DENRE

In the partial fulfilment of the award of the degree of

BCA

in the

[BCA(CSE)]

Of

JIS UNIVERSITY

Ardent Computech Pvt. Ltd.

ARDENT
COMPUTECH PVT. LTD.
High-End Technology Training and Project

Ardent Computech Pvt. Ltd.
Drives you to the Industry
Module 132, SDF Building, Sector V, Salt Lake, Pin - 700091
www.ardentcollaborations.com

CERTIFIED ISO 9001:2015 CERTIFIED

UDYAM REGISTRATION MSME

N·E·A·T
National Educational Alliance for Technology

CERTIFICATE FROM SUPERVISOR

This is to certify that **Subhra Kolay, Chandrima Ray, Avik Bera, Akash Maity, Ayan Das, Bristi Denre, 23CS2061173, 23CS2061048, 23CS2061040, 23CS2061007, 23CS2061041, 23CS2061046** have completed the project titled Technician Assignment Manager under my supervision during the period from **16/06/2025 to 05/07/2026** which is in partial fulfilment of requirements for the award of the BCA degree and submitted to the Department of **BCA(CSE)** of **JIS University.**

____

Signature of the Supervisor

Date: 05/07/2025

Name of the Project Supervisor: Subhojit Santra

22 YEARS OF EXCELLENCE

Ardent Computech Pvt. Ltd.

Drives you to the Industry

Module 132, SDF Building, Sector V, Salt Lake, Pin - 700091

www.ardentcollaborations.com

ARDENT COMPUTECH PVT. LTD.

High-End Technology Training and Project

ISO 9001:2015 CERTIFIED

UDYAM REGISTRATION MSME

N·E·A·T National Educational Alliance for Technology

BONAFIDE CERTIFICATE

Certified that this project work was carried out under my supervision

Finsight: Automated expenses recording and monitoring system is the Bonafide work of

Name of the student: Subhra Koley	Signature: Subhra Koley
Name of the student: Chandrima Ray	Signature: Chandrima Ray
Name of the student: Akash Maity	Signature: Akash Maity
Name of the student: Avik Bera	Signature: Avik Bera
Name of the student: Ayan Das	Signature : [illegible]
Name of the student: Bristi Denre	Signature: Bristi Denre.

SIGNATURE

Name :

PROJECT MENTOR

SIGNATURE

Name:

EXAMINERS

Ardent Original Seal

ACKNOWLEDGEMENT

The achievement that is associated with the successful completion of any task would be incomplete without mentioning the names of those people whose endless cooperation made it possible. Their constant guidance and encouragement made all our efforts successful.

We take this opportunity to express our deep gratitude towards our project mentor, ***[Subhojit Santra]*** for giving such valuable suggestions, guidance and encouragement during the development of this project work.

Last but not the least we are grateful to all the faculty members of **Ardent Computech Pvt. Ltd.** for their support.

CONTENT PAGE

1. COMPANY PROFILE

ARDENT (Ardent Computech Pvt. Ltd.), formerly known as Ardent Computech Private Limited, is an ISO 9001 :2015 certified Software Development and Training Company based in India. Operating independently since 2003, the organization has recently undergone a strategic merger with ARDENT Technologies, enhancing its global outreach and service offerings.

ARDENT Technologies
ARDENT Technologies delivers high-end IT services across the UK, USA, Canada, and India. Its core competencies lie in the development of customized application software, encompassing end-to-end solutions including system analysis, design, development, implementation, and training. The company also provides expert consultancy and electronic security solutions. Its clientele spans educational institutions, entertainment companies, resorts, theme parks, the service industry, telecom operators, media, and diverse business sectors.

ARDENT Collaborations
ARDENT Collaborations, the Research, Training, and Development division of ARDENT (Ardent Computech Pvt. Ltd.), offers professional IT-enabled services and industrial training programs. These are tailored for freshers and professionals from B. Tech, M. Tech, MBA, MCA, BCA, and MSc backgrounds. ARDENT (Ardent Computech Pvt. Ltd.) provides Summer Training, Winter Training, and Industrial Training to eligible candidates. High-performing students may qualify for stipends, scholarships, and additional benefits based on performance and mentor recommendations.

Associations and Accreditations
ARDENT (Ardent Computech Pvt. Ltd.) is affiliated with the National Council ofVocational Training (NCVT) under the Directorate General of Employment & Training (DGET), Ministry of Labour & Employment, Government of India. The institution upholds strict quality standards under ISO 9001 :2015 certification and is dedicated to bridging the gap between academic knowledge and industry skills through innovative training programs.

2. INTRODUCTION

An **Automated Expense Recording and Monitoring System** is a digital solution designed to streamline and simplify the entire process of managing financial expenditures within an organization or for individuals. In today's fast-paced world, manual expense tracking can be time-consuming, error-prone, and inefficient, often leading to delayed reimbursements, inaccurate financial reports, and difficulty in adhering to budgets. This system leverages technology to automate key tasks such as, automatically recording expenses through various methods like receipt scanning (using OCR/AI), manual entry, or direct integration with bank accounts/credit cards. Intelligent categorization of expenses to provide clear insights into spending patterns. Routing expenses through defined approval hierarchies for quick and efficient authorization. Real-time tracking of spending against pre-set budgets, providing alerts for potential overspending. Generating comprehensive reports and visual dashboards to offer insights into financial health, identify cost-saving opportunities, and ensure compliance.

By centralizing expense data and automating repetitive tasks, these systems empower users with greater control over their finances, enhance transparency, reduce administrative overhead, and improve overall financial accuracy and efficiency. They are essential tools for modern businesses and individuals aiming to optimize their financial management processes.

2A.OBJECTIVE

The primary objectives of an Automated Expense Recording and Monitoring System are to enhance financial efficiency, transparency, and control for individuals or organizations. Here are the key objectives:

- **Streamline Expense Recording:** To provide an intuitive and efficient way for users to record expenses, minimizing manual data entry and reducing the time spent on administrative tasks. This includes supporting various input methods like manual entry and automated data extraction from receipts (OCR).
- **Improve Data Accuracy:** To minimize errors associated with manual data entry through automation, validation rules, and AI-powered data extraction, ensuring that financial records are precise and reliable.
- **Enhance Financial Visibility:** To offer real-time insights into spending patterns, categorized expenditures, and overall financial status through comprehensive reports and dashboards, enabling better financial planning and decision-making.
- **Facilitate Budget Adherence:** To enable users and administrators to set and monitor budgets effectively, providing automated alerts and notifications when spending approaches or exceeds predefined limits, thereby promoting fiscal responsibility.
- **Accelerate Reimbursement and Approval Processes:** To automate and expedite the expense approval workflow, ensuring timely processing of expense claims and quicker reimbursements

for employees.

- **Reduce Administrative Overhead:** To significantly decrease the administrative burden on individuals and finance departments by automating repetitive tasks, freeing up valuable time for more strategic activities.

By achieving these objectives, an Automated Expense Recording and Monitoring System transforms a typically tedious and complex process into a seamless, accurate, and strategically valuable function.

2B.SCOPE

To develop a web-based system that allows users to automatically record, monitor, and manage their daily expenses efficiently, helping them track financial habits and control unnecessary spending**.**

1. User Authentication
   - Register and login/logout functionality.
   - Secure password handling
2. Expense Management:
   - Add, edit, and delete expense records.
   - Store details like amount, date, category, and description.
3. Expense Categorization
   - Organize expenses into categories (e.g., food, travel, utilities).
   - Option to create custom categories.
4. Data Visualization:
   - Display charts (pie, bar) for monthly or category-wise expense breakdown.
   - Dashboard with total, daily, and monthly summaries.
5. Search and Filter:
   - Filter by date range, category, or amount.
   - Search by keyword in description.

3. SYSTEM ANALYSIS

3A.1DENTIFICATION OF NEED

In today's digital world, managing personal or business finances manually has become inefficient, prone to errors, and time-consuming. Individuals often forget to log expenses, misplace receipts, or lack insights into their spending patterns. To address the above challenges, there is a clear need for an **Automated Expenses Recording and Monitoring System** that can:

- Record expenses instantly and accurately
- Categorize and store expenses securely
- Monitor spending habits using dashboards and charts
- Help users make better financial decisions
- Provide reminders or alerts to stay within budget

.

3B.FEASIBILITY STUDY

The **feasibility study** is conducted to determine whether the proposed system is viable and worth developing. It evaluates the project from multiple perspectives to ensure success in real-world conditions.

Technical Feasibility: These technologies are open-source, lightweight, and scalable, making them ideal for modern web applications.

Operational Feasibility: Users can quickly learn to add, edit, and monitor expenses without prior training.

Economic Feasibility: Development uses free tools and frameworks, reducing the overall cost. Offers lengthy-time period value savings and high go back on funding via green useful resource use.

Legal Feasibility: If user login and data storage are used, you can implement basic privacy policies and terms of service.

Schedule Feasibility: Can be advanced and deployed inside an inexpensive timeframe (3—6 months).

3C.WORKFLOW

The **Waterfall Model** is a **linear and sequential approach** to software development. Each phase must be completed before moving on to the next.

Phases in the Waterfall Model:

1. **Requirement Analysis** – Identify user needs and define system goals.
2. **System Design** – Create architecture, database schema, and UI layout.
3. **Implementation** – Develop the actual code for frontend and backend.
4. **Testing** – Verify the system using functional and non-functional testing.
5. **Deployment** – Make the system live and available to users.
6. **Maintenance** – Fix bugs, add updates, and support users' post-launch.

Waterfall Model Design:

The waterfall approach was the first SDLC Model to be used widely in Software Engineering to ensure the success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In the Waterfall model, typically, the Outcome of one phase acts as the input for the next phase sequentially.

Iterative Waterfall Design:

Definition: The Iterative Waterfall Model is an enhanced version of the traditional Waterfall Model. It follows the same linear steps as the original Waterfall but allows feedback and modification after each phase. If issues are discovered in later phases like testing or implementation, the development team can go back to previous phases, make corrections, and proceed again. The sequential phases in Iterative Waterfall model are:

➢ Requirement Gathering and Analysis: All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.

➢ System Design: The requirement specifications from the first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.

➢ Implementation: With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.

➢ Integration and Testing: All the units developed in the implementation phase are integrated into a system after testing each unit. Post integration the entire system is tested for any faults and failures.

➢ Deployment of the system: Once the functional and non-functional testing is done, the product is deployed in the customer environment or released into the market.

➢ Maintenance: Some issues come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in progress and are seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for the previous phase and it is signed off, so the name "Iterative Waterfall Model". In this model, phases do not overlap.

- Advantages:

1. Improved Flexibility:

Allows going back to previous phases (like design or requirement) if issues are found during testing. Supports changes without starting from scratch.

2. Early Error Detection:

Bugs and issues can be found and fixed earlier, reducing time and cost in later stages.

3. Better Risk Management:

Problems discovered at any stage can be reviewed and handled in the previous phase before moving forward.

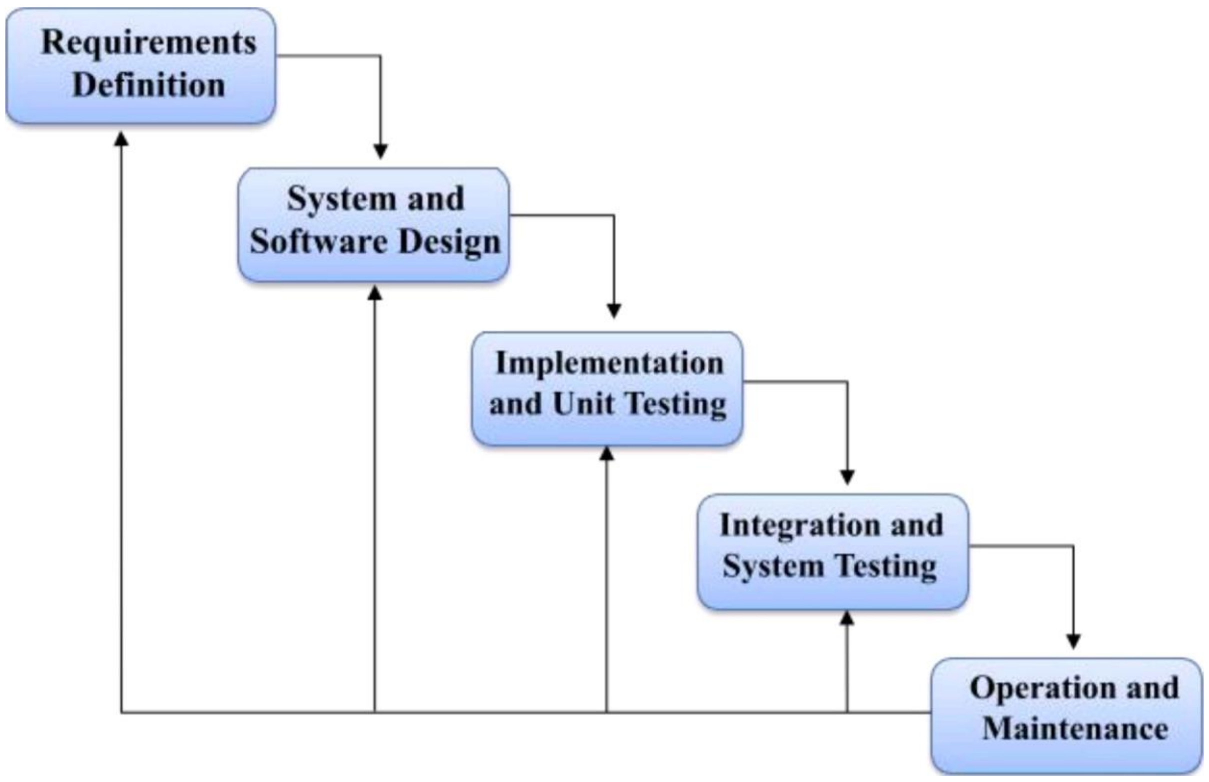
- Disadvantages:

  Incomplete Requirement Risks

  If the initial requirements are not well understood, it may affect multiple phases and lead to repeated iterations.

  Complex Management
  Managing documentation and rework in multiple iterations can be difficult, especially with larger teams.

  Delayed Working Model
  A fully usable product may not be available until late in the development process, since integration happens later.


Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

- Applications:

The Iterative Waterfall Model is suitable for projects with evolving or unclear requirements. It is commonly used in software development projects where regular feedback and refinement are essential.

Additionally, it is applicable in scenarios where partial system delivery is beneficial, allowing stakeholders to assess progress and make adjustments.

3D . STUDY OF THE SYSTEM- TECHNICIAN ASSIGNMENT MANAGER

Modules: The modules used in this software are as follows:

- ❑ Register:

1 **User Register**: Here, the user will register to buy any course.
2 . **Admin Register:** Here, the admin will register to handle all the databases.

- **Verify Account:** When a user login an OTP sends to the user Email to check the validity of the data.

- Login:

  1 . **User Login:** Here users will login to see available courses and buy any course.
  2 . **Admin Login:** Here admin will log in to handle all the data.

- **Home:** This page is to see feedback received from learners.

- Courses:
  1. **User interface:** This page shows the available courses that the users can purchase. They can also search for courses if available.
  2. **Admin interface:** In this page, the Admin can view or delete courses.

- Lectures:
  1. **User interface**: If the users purchase any course then they can access the lectures inside the course.
  2. **Admin interface**: Admin can add lectures of any specific course.

- **About:** This page will show the details about the website developers.

- Account:
  1. **User Interface:**
     a) **My Profile:** Here, the user can see the details after login.
     b) **Dashboard:** Here, the user can see the courses he/she has purchased.
  2. Admin Interface:
     a) **Admin Profile:** Here, the Admin can see the details after login.
     b) **Admin Dashboard:** Here the Admin can manage courses and user roles.

3E. INPUT AND OUTPUT -

The main inputs, outputs and the major function the details are:

➢ INPUT:

- User Inputs

Input Field	Description
Username / Email	Used for user login and identification
Password	Required for secure authentication
Expense Title	Name of the expense (e.g., "Lunch", "Rent")
Amount	How much money was spent
Date	Date of the expense
Category	Type of expense (e.g., Food, Travel, Bills)
Description *(optional)*	Additional details about the expense
Filter / Search Query	Used to search specific expenses or filter records

- **Admin Inputs**

Input Field	Description
View Users	Admin can access a list of registered users
Manage Expenses	Admin can review or delete user expense records
System Settings	Admin can configure or adjust system parameters

- **system-Generated Inputs**

Field	Description
Timestamp	Automatically stores the time/date when the record is saved
User ID	Identifies which user made the expense
Expense ID	Unique identifier for each expense entry

➢ **OUTPUT:**

- Homepage (UI/React preview)
- MongoDB backend response (e.g., saving or showing expense records)
- Login / Registration page
- Form to enter expenses
- Expenses list display (table or cards)
- Dashboard with statistics or charts
- Complete running output (frontend + backend)

3F. SOFTWARE REQUIREMENT SPECIFICATIONS

Software Requirements Specification provides an overview of the entire project. It is a description of a software system to be developed, laying out functional and non-functional requirements. The software requirements specification document enlists enough necessary requirements that are required for the project development. To derive the requirements, we need to have a clear and thorough understanding of the project to be developed. This is prepared after detailed communication with the project team and the customer.

The developer is responsible for:

- Developing the system, which meets the SRS and solves all the requirements of the system.
- Demonstrating the system and installing the system at the client's location after acceptance testing is successful.
- Submitting the required user manual describing the system interfaces to work on it and also the documents of the system.

Functional Requirements:

A. **User Registration and Authentication:**
   1. Users should be able to create accounts securely.
   2. The system should authenticate users and manage login sessions.

B. **Browse and Search:**
   1. Users should be able to browse and search for courses.

.

   2. Add Expenses.

C. **Users can add expense entries with the following details:**

1. Title/Category (e.g., Food, Transport, Utilities)
2. Amount
3. Date
4. Notes/Description (optional)

D. Edit/Delete Expenses:

- Users can **edit** or **delete** previously recorded expenses.

E. View Expenses:

- Users can view a **list or table** of all their recorded expenses.
- Can filter by **date range, category, or amount**.

F. Search Functionality:

- A search bar should allow users to search through their expenses.

G. Expense Reports/Summary:

- Show **monthly, weekly, or yearly summaries** of expenses.
- Include **total spent** and **category-wise breakdown**.

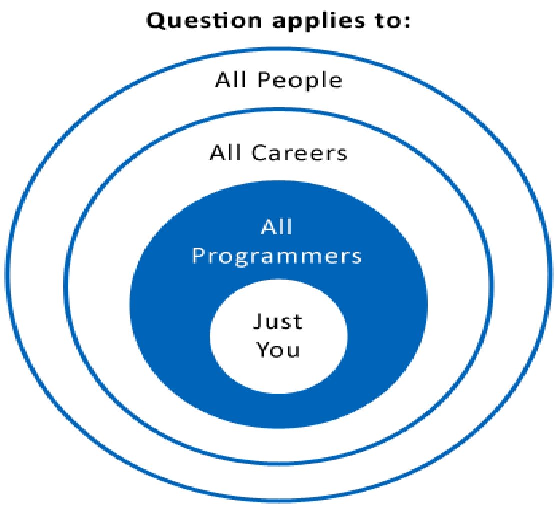
H. Data Visualization:

- Show expense trends via **bar charts, pie charts, or line graphs**.

Useful for analysing where money is spent most

3G. SOFTWARE ENGINEERING PARADIGM APPLIED

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in software engineering these paradigms stand. These can be combined into various categories, though each of them is contained in one another.



The programming paradigm is a subset of Software design paradigm which is further a subset of the Software development paradigm.

There are two levels of reliability. The first is meeting the right requirements. A careful and thorough systems study is needed to satisfy this aspect of reliability. The second level of systems reliability involves the actual work delivered to the user. At this level, the system's reliability is interwoven with software engineering and development.

There are three approaches to reliability.

❖ **Requirement Analysis:**

- Functional and non-functional requirements are gathered (e.g., user login, record expense, view report).

❖ **System Design:**

- Architecture is planned (Frontend in React, Backend in Node.js/Express, Database in MongoDB).
- UI wireframes and database schema are created.

❖ **Implementation:**

- Each module (e.g., login, add expense) is implemented and tested.

- ❖ **Testing:**
    - Unit testing and integration testing are done after each iteration.
    - Bugs and logic errors are resolved.
- ❖ **Deployment:**
    - The final version is hosted locally or on a platform like **Render**, **Netlify**, or **Vercel**.
- ❖ **Maintenance:**
    - Bugs are fixed and small features can be added based on user feedback.

4. SYSTEM DESIGN

4A. DATA FLOW DIAGRAM

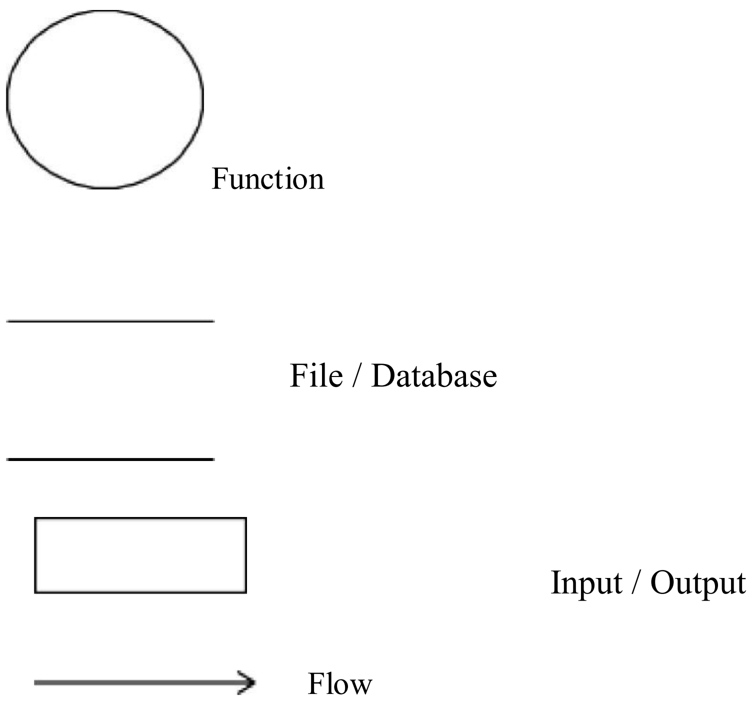
A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated.

DFD can also be used for the visualization of data processing (structured design). A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of the process or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).

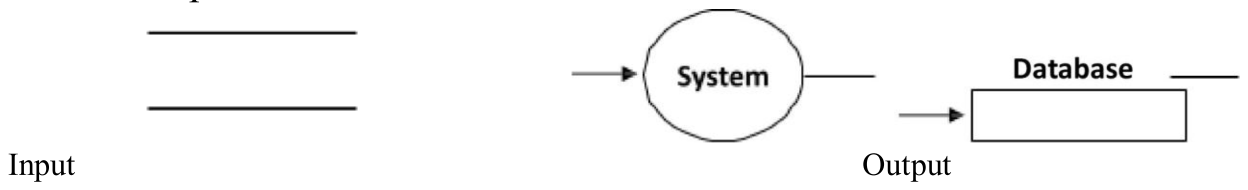
This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modeled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present for the system to do its job and shows the flow of data between the various parts of the system.

Data flow diagrams are one of the three essential perspectives of the structured-systems analysis and design method SSADM. The sponsor of a project and the end users will need to be briefed and consulted throughout all stages of a system's evolution. With a data flow diagram, users can visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's data-flow diagrams can be drawn up and compared with.

DFD Notation:



DFD Example:



Steps to Construct Data Flow Diagram:

Four Steps are generally used to construct a DFD.

Process should be named and referred for easy reference. Each name should be representative of the reference.

The destination of flow is from top to bottom and from left to right.

When a process is distributed into lower-level details they are numbered.

The names of data stores, sources, and destinations are written in capital letters.

Rules for constructing a Data Flow Diagram:

Arrows should not cross each other.

Squares, Circles, and Files must bear a name.

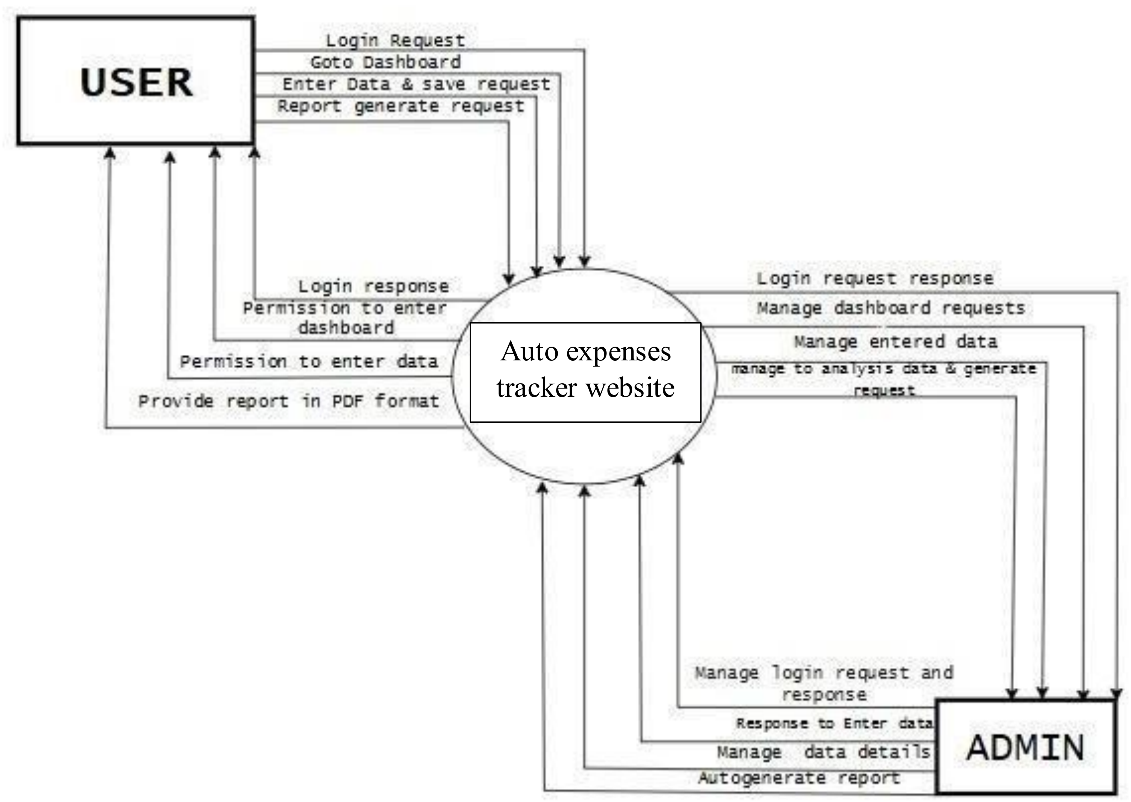
Decomposed data flow squares and circles can have the same names.

Draw all data flow around the outside of the diagram.

DATA FLOW DIAGRAM (Level 0 and Level 1):

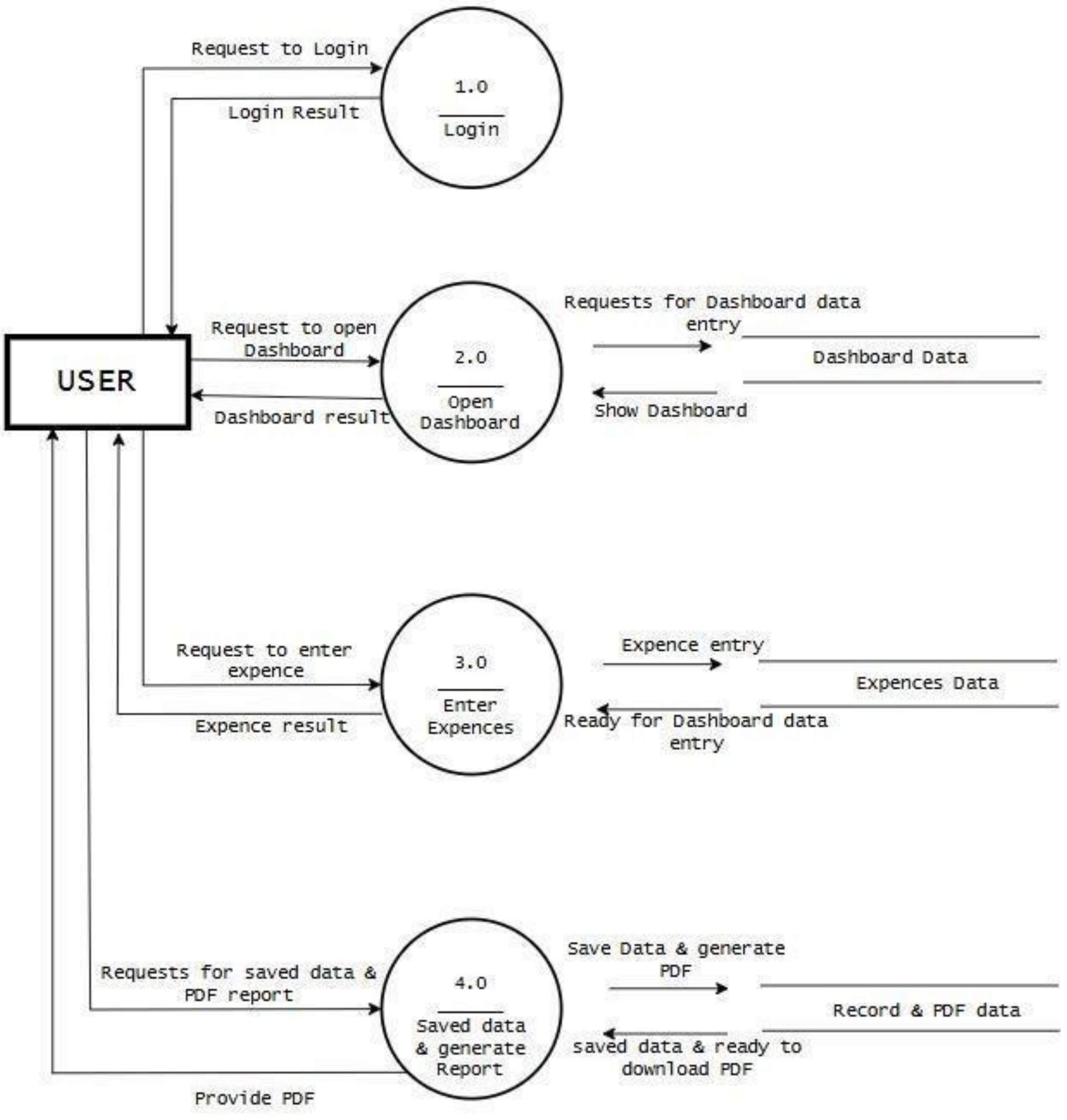
1. Level 0 (Context Level DFD):

This shows the overall system and how external entities interact with it.

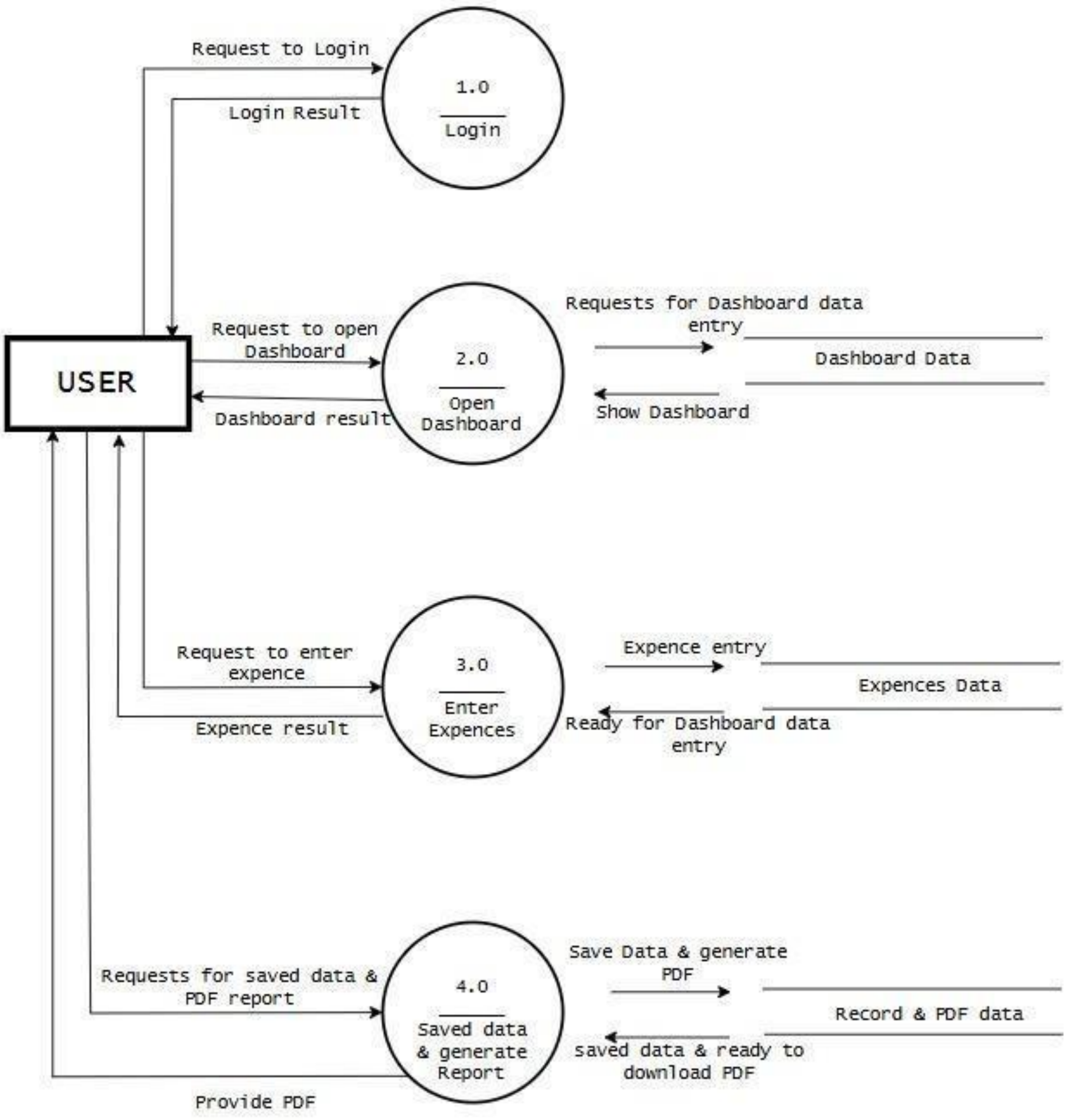

USER
Login Request
Goto Dashboard
Enter Data & save request
Report generate request
Login response
Permission to enter dashboard
Permission to enter data
Provide report in PDF format
Auto expenses tracker website
Login request response
Manage dashboard requests
Manage entered data
manage to analysis data & generate request
Manage login request and response
Response to Enter data
Manage data details
Autogenerate report
ADMIN

2. Level 1 DFD

This breaks down the main system into functional components.



- **LEVEL 1 DFD:**



Request to Login
Login Result
1.0
Login
USER
Request to open Dashboard
Dashboard result
2.0
Open Dashboard
Requests for Dashboard data entry
Show Dashboard
Dashboard Data
Request to enter expence
Expence result
3.0
Enter Expences
Expence entry
Ready for Dashboard data entry
Expences Data
Requests for saved data & PDF report
Provide PDF
4.0
Saved data & generate Report
Save Data & generate PDF
saved data & ready to download PDF
Record & PDF data

4B.SEQUENCE DIAGRAM

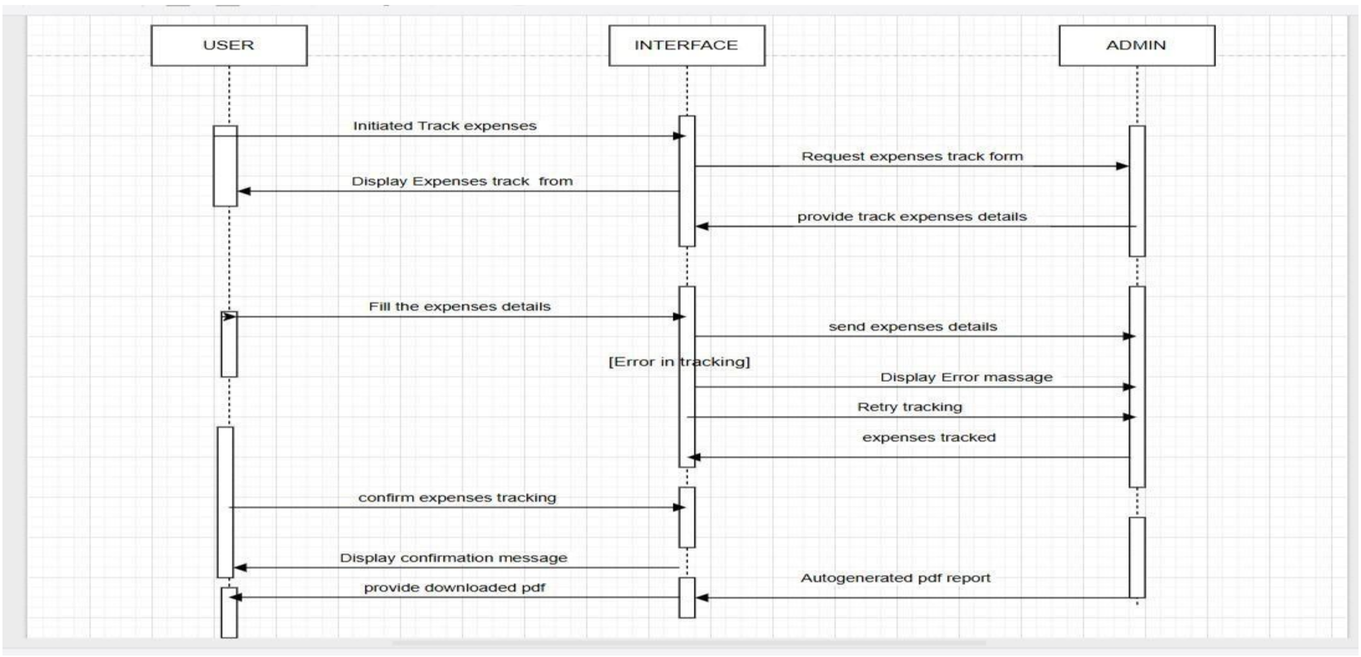
A Sequence diagram is an interaction diagram that shows how processes operate with one another and what is their order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in a time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development.

Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

A sequence diagram is the most common kind of interaction diagram, which focuses on the message interchange between several life lines .A sequence diagram describes an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding occurrence specifications on the lifelines.

The following nodes and edges are typically drawn in a UML sequence diagram: lifeline, execution specification, message, fragment, interaction, state invariant, continuation, and destruction occurrence.



A Use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the use rand the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

So only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML there are five diagrams available to model dynamic nature and a use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. So, use case diagrams consist of actors, use cases, and their relationships. The diagram is used to model the system/subsystem of an application. A single-use case diagram captures a particular functionality of a system.

So, to model the entire system numbers of use case diagrams are used. The purpose of a use case diagram is to capture the dynamic aspect of a system. But this definition is too generic to describe the purpose.

Because the other four diagrams (activity, sequence, collaboration, and State chart) are also having the same purpose. So, we will look into some specific purpose that will distinguish it from the other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So, when a system is analyzed to gather its functionalities use cases are prepared and actors are identified.

Now when the initial task is complete use case diagrams are modeled to present the outside view. So, in brief, the purposes of use case diagrams can be as follows:

- ❑ Used to gather requirements of a system.
- ❑ Used to get an outside view of a system.
- ❑ Identify external and internal factors influencing the system.

.

How to draw Use Case Diagram?

Use case diagrams are considered for high level requirement analysis of a system. So, when the requirements of a system are analysed, the functionalities are captured in use cases.

So, we can say that uses cases are nothing but the system functionalities written in an organized manner. Now the second things which are relevant to the use cases are the actors. Actors can be defined as something that interacts with the system.

The actors can be human user, some internal applications or may be some external applications. So, in a brief when we are planning to draw use case diagram, we should have the following items identified.

Functionalities to be represented as a use case

Actors

Relationships among the use cases and actors.

Use case diagrams are drawn to capture the functional requirements of a system. So, after identifying the above items we have to follow the following guidelines to draw an efficient use case diagram.

The name of a use case is very important. So, the name should be chosen in such a way so that it can identify the functionalities performed.

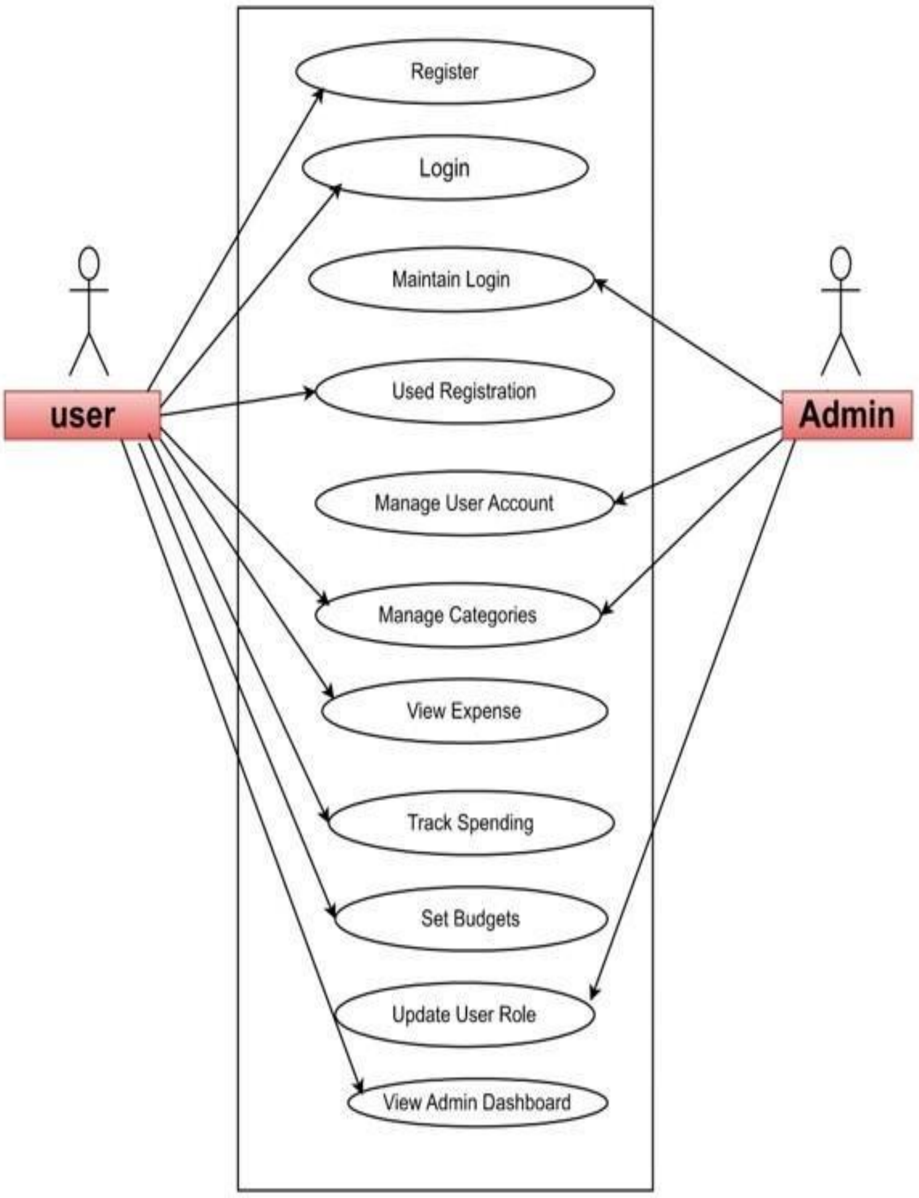
Give a suitable name for actors.

Show relationships and dependencies clearly in the diagram.

Do not try to include all types of relationships. Because the main purpose of the diagram is to identify requirements.

Use note whenever required to clarify some important point

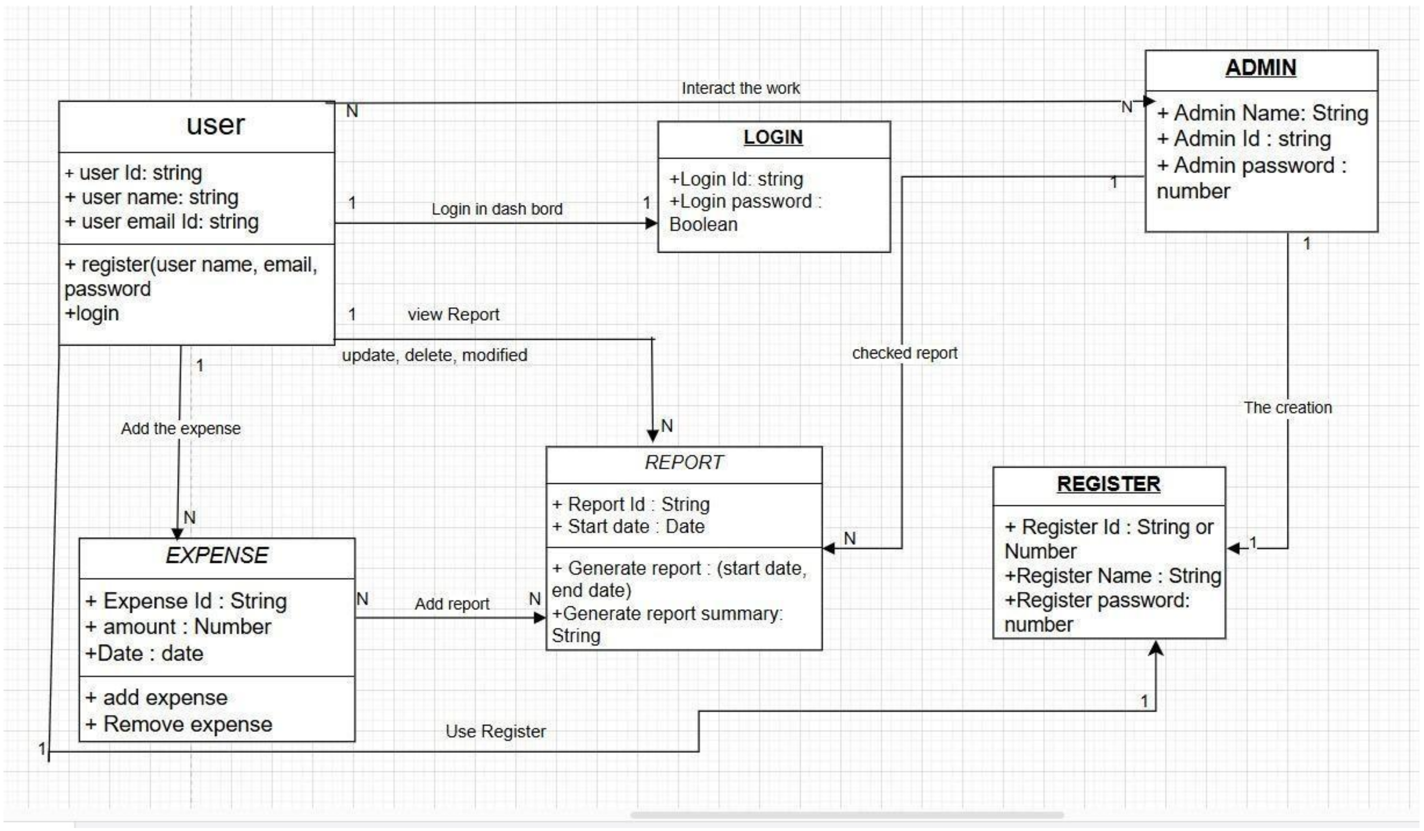
● USE CASE DIAGRAM:



4C. SCHEMA DIAGRAM

The schema is an abstract structure or outline representing the logical view of the database as a whole. Defining categories of data and relationships between those categories, database schema design makes data much easier to retrieve, consume, manipulate, and interpret.

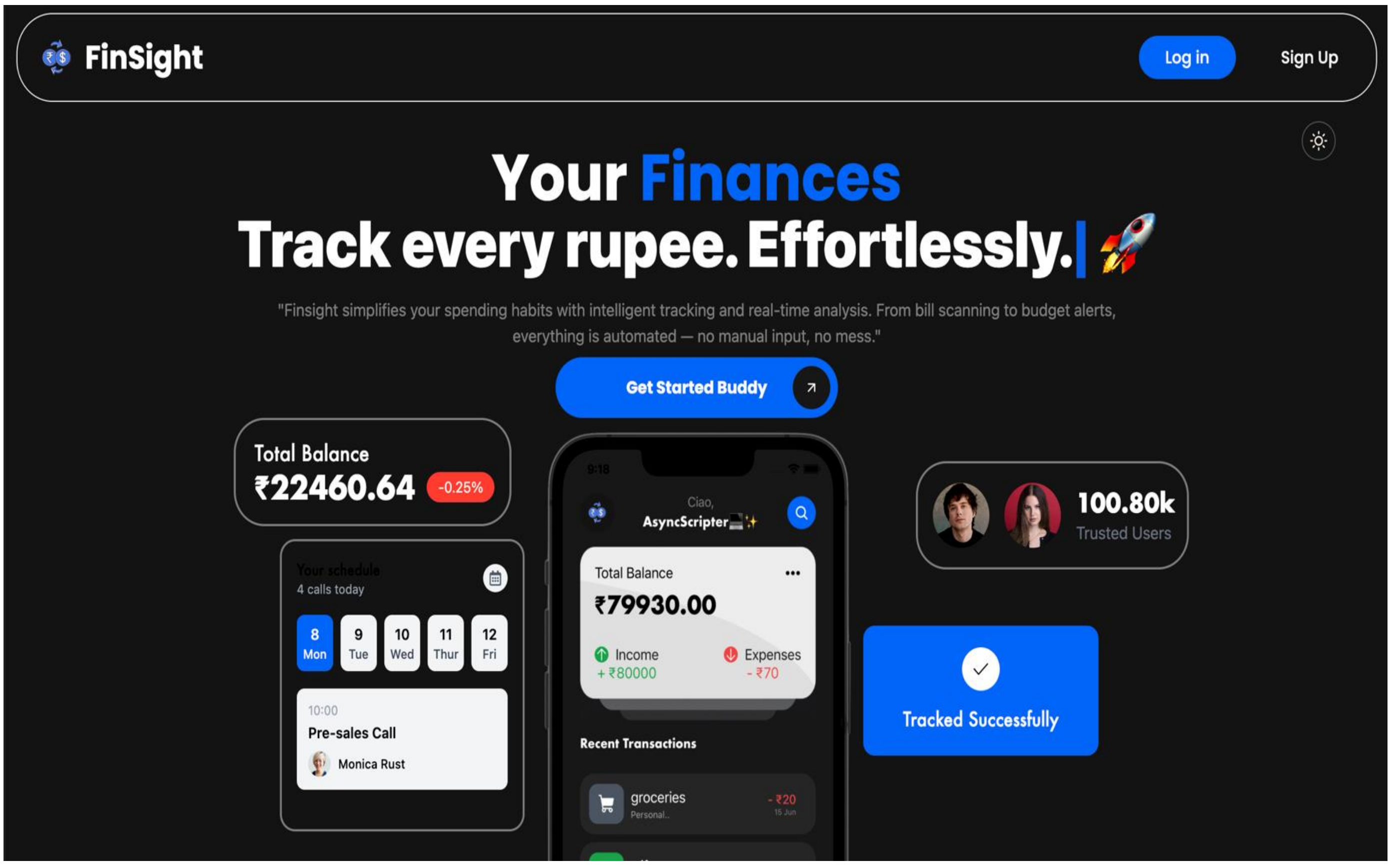
DB schema design organizes data into separate entities, determines how to create relationships between organized entities, and influences the applications of constraints on data. Designers create database schema to give other database users, such as programmers and analysts, a logical understanding of data.



4. UI SNAPSHOT

1) FRONTEND : -

➢ **Welcome page:**



✓ **CODE:**

import React, { useState } from 'react'; import Navbar from '../../compents/Navbar'; import Hero from '../../compents/Hero'; import { FiSun, FiMoon } from 'react-icons/fi';

const Welcome = () => { const [darkMode, setDarkMode] = useState(false);

return (<div className={${darkMode ? 'bg-[#131313] text-[#FAF1E6]' : 'bg-[#FAF1E6] text-black'} px-3 py-3 h-[100vh] transition-all duration-300}> {/* Toggle Button */}

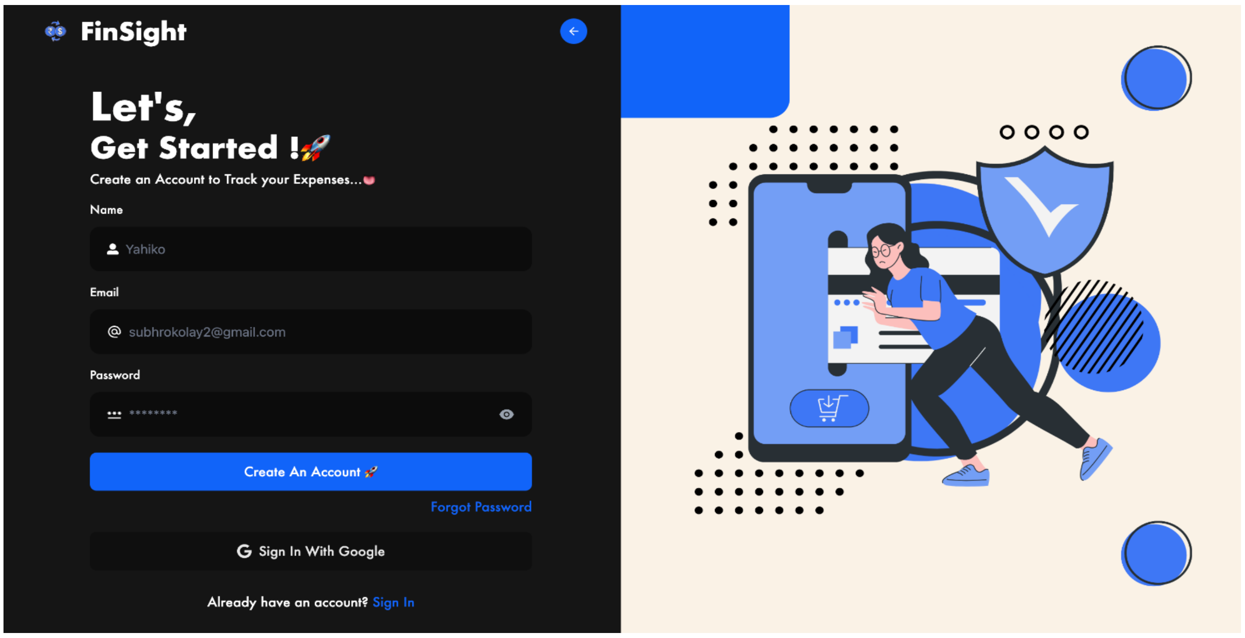
<button onClick={() => setDarkMode(prev => !prev)} className="p-2 rounded-full border mb-2" style={{ borderColor: darkMode ? '#555' : '#ccc' }} > {darkMode ? : }

```
          <Navbar title={"FinSight"} darkMode={darkMode} />
                <Hero darkMode={darkMode} />
              </div>


          ); };
```

export default Welcome;

---

1. **Sign-up Page:**



FinSight

Let's,
Get Started !🚀
Create an Account to Track your Expenses...❤️

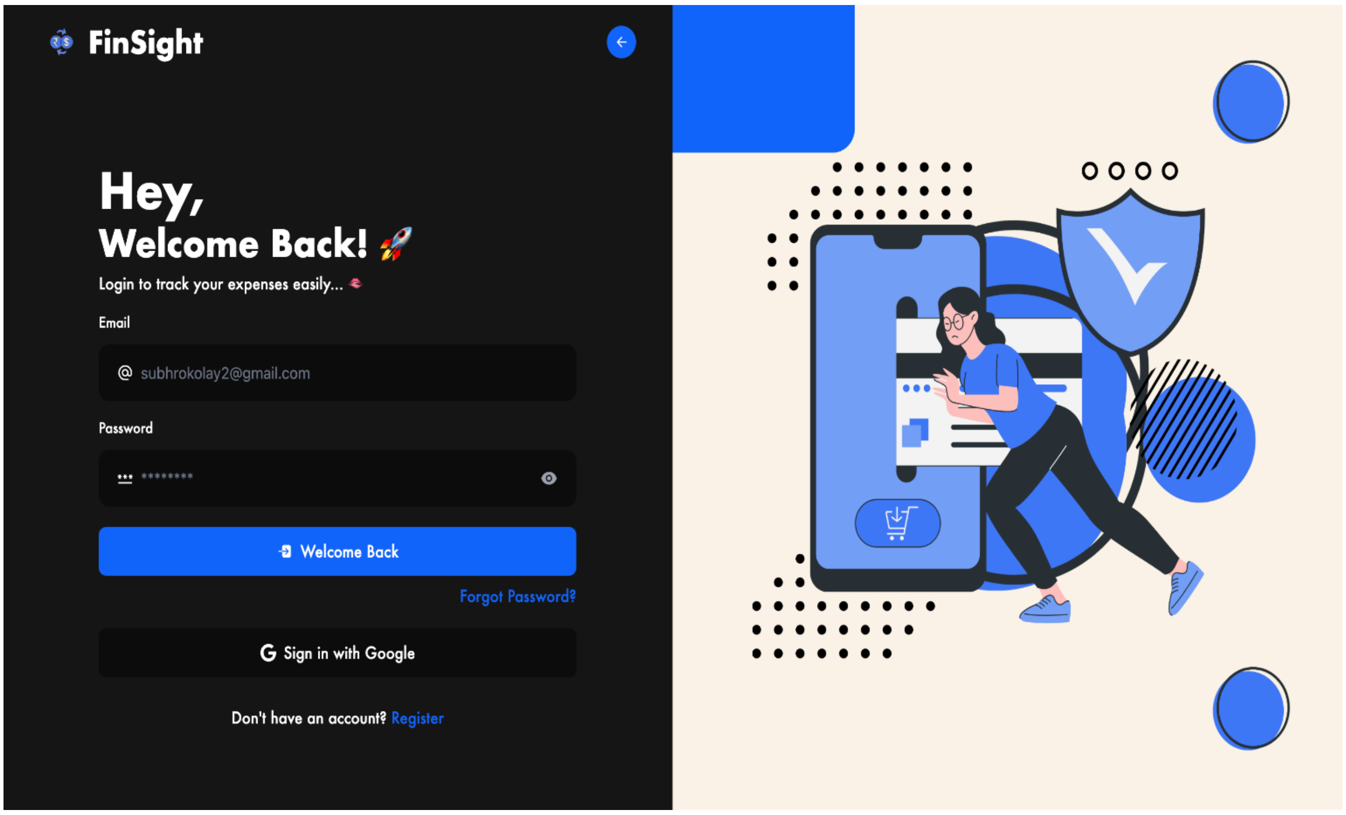
Name: Yahiko
Email: subhrokolay2@gmail.com
Password: ••••••••

✓ **Code:**

```
import React, { useState } from 'react';
import { colors } from '../../constant/color';
import Authlayout from '../../compents/layout/Authlayout';
import CustomInput from '../../compents/Inputs/CustomInput';
import { MdAlternateEmail, MdPassword } from "react-icons/md";
import GeneralCusBtn from '../../compents/Inputs/GeneralCusBtn';
import { FaGoogle, FaUserAlt } from 'react-icons/fa';
import { useNavigate } from 'react-router-dom';
import axiosInstance from '../../utils/axiosInstance';
import { API_PATHS } from '../../utils/apiPaths';
import { useContext } from 'react';
import { UserContext } from '../../context/useContext';
import { validateEmail } from '../../utils/validate';
const SignUp = () => {
  const [form, setForm] = useState({
   name: "",
   email: "",
   password: ""
  });
  const [error, setError] = useState(null);
  const router = useNavigate();
  const {updateUserData} = useContext(UserContext)
  const handelSub = async (e) => {
   e.preventDefault();

   if (!form.name || !form.email || !form.password) {
    alert("Please fill in all the fields before submitting.");
    return;
   }
  if (!validateEmail (form.email)) {
   alert("Please enter a valid email address.");
   return;
  }
/*  if (!validatePassword(form.password)) {
   alert("Password must be at least 6 characters long and include both letters and
numbers.");
   return;
  } */
   setError("")
   try {
    const response = await axiosInstance.post(API_PATHS.AUTH.REGISTER, {...form});
    const {token,user} = response.data;
    if(token) {
     localStorage.setItem("token",token)
     updateUserData(user);
     router("/home");
    }
   } catch (error) {
    if(error.response?.data?.message) {
      setError(error.response && error.response.data.message)
    }else {
       setError("Something wents wrong , Server failed")
    }
   }

  };
  const toggle = () => {
   router("/signin");
  };
  return (
   <Authlayout>
    <div className='w-full h-full flex items-center justify-center'>
    <div className='lg:w-[80%] h-auto md:h-full flex flex-col justify-center'>
     <div className='flex flex-col gap-2'>
     <h1 className='text-5xl text-white font-[700]' style={{ fontFamily: 'futura'
}}>Let's,</h1>
<h1 className='text-4xl font-[700] text-white' style={{ fontFamily: 'futura' }}></h1>
          <p className='text-white' style={{ fontFamily: 'futura' }}>
Create an Account to Track your Expenses...</p>
        </div>
        <form onSubmit={handelSub}>
          <CustomInput
            label={"Name"}
            placeHolder={"DaniDaniYessPapa"}
            type={"text"}
            value={form.name}
            onChange={(e) => setForm({ ...form, name: e.target.value })}
            icons={<FaUserAlt size={14} />}
          />
          <CustomInput
            placeHolder={"danidaniels@gmail.com"}
            label={"Email"}
            type={"email"}
            value={form.email}
            onChange={(e) => setForm({ ...form, email: e.target.value })}
            icons={<MdAlternateEmail />}
          />
          <CustomInput
            placeHolder={""}
            type={"password"}
            label={"Password"}
            value={form.password}
            onChange={(e) => setForm({ ...form, password: e.target.value })}
            icons={<MdPassword />}
          />
          <GeneralCusBtn type='sumbit' name={"Create An Account "} />
          <p className='text-right mt-2' style={{ color: colors.primary, fontFamily: 'futura'
}}>
            Forgot Password </p>
          <GeneralCusBtn
            name={"Sign In With Google"}
            backgroundColors={"#101010"}
            lefttIcon={<FaGoogle />}
          />
        </form>
        <h1 className='text-white text-center mt-7' style={{ fontFamily: 'futura' }}>
          Already have an account?{" "}
          <span
            onClick={toggle}
            style={{ color: colors.primary, fontFamily: 'futura', cursor: 'pointer' }}>
            Sign In
          </span>
        </h1>
```

```
                                </div>
                              </div>
                            </Authlayout>
                          );};
export default SignUp;
```

2) Login Page:



FinSight

Hey,
Welcome Back! 🚀

Login to track your expenses easily...

Email

subhrokolay2@gmail.com

Password

****

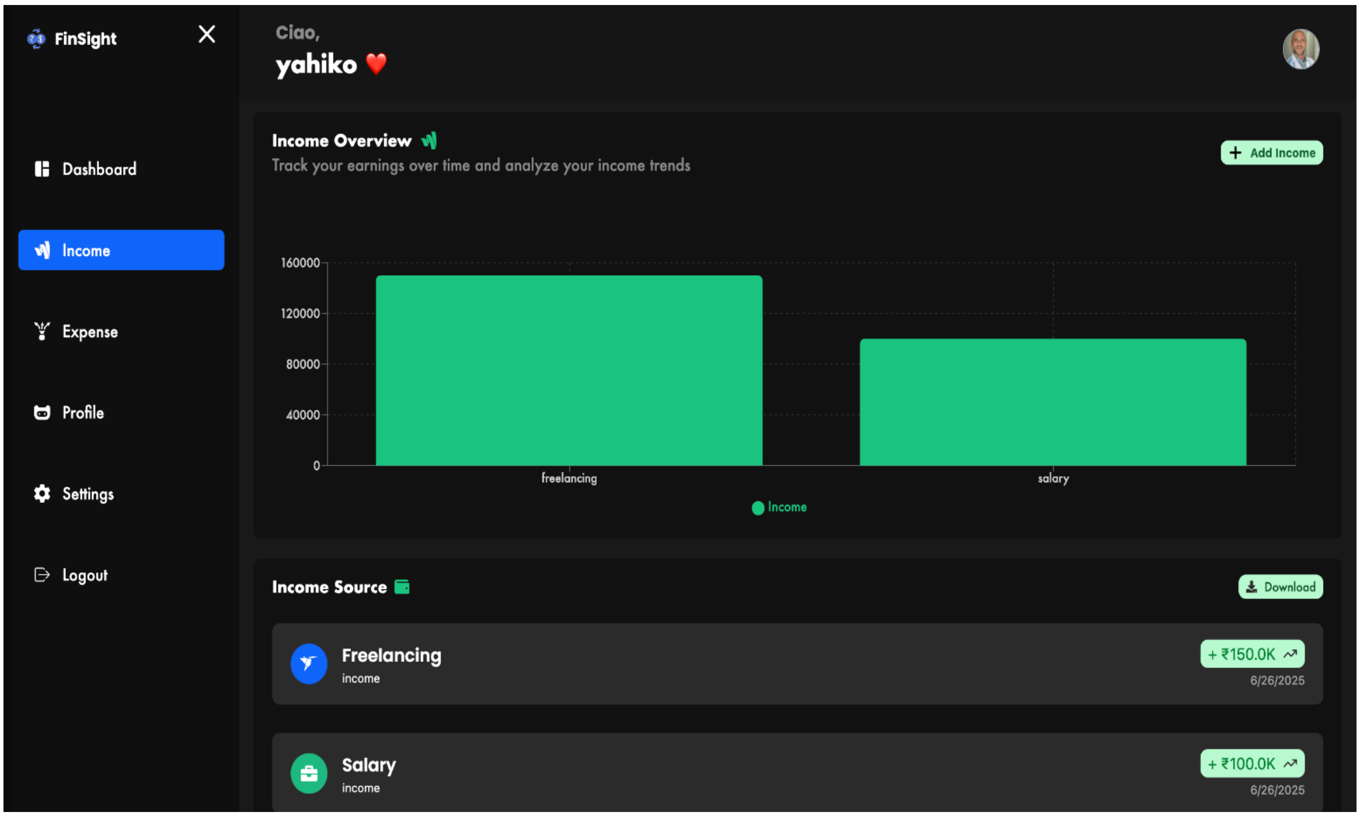
✓ **CODE**

- **Signin.jsx:**

```
import React, { useContext, useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { MdAlternateEmail, MdPassword } from 'react-icons/md';
import { FaGoogle } from 'react-icons/fa';
import { IoLogIn } from 'react-icons/io5';
import Authlayout from '../../compents/layout/Authlayout';
import CustomInput from '../../compents/Inputs/CustomInput';
import GeneralCusBtn from '../../compents/Inputs/GeneralCusBtn';
import { colors } from '../../constant/color';
import axiosInstance from '../../utils/axiosInstance';
import { API_PATHS } from '../../utils/apiPaths';
import { validateEmail, validatePassword } from '../../utils/validate';
import UserProvier, { UserContext } from '../../context/useContext';
import toast from 'react-hot-toast';
const SignIn = () => {
 const navigate = useNavigate();
 const [form, setForm] = useState({
  email: "",
  password: ""
 });
 const [error, setError] = useState(null);
  const { user, updateUserData, clearUserData } = useContext(UserContext)
 const handleSubmit = async (e) => {
  e.preventDefault();
  if (!form.email || !form.password) {
   return alert("Please fill all the fields.");
  }
  if (!validateEmail(form.email)) {
   return alert("Please enter a valid email.");
  }
  // Uncomment this if password validation is needed
  // if (!validatePassword(form.password)) {
    //   return alert("Please enter a strong password.");
    // }
    setError("");
    try {
      const response = await axiosInstance.post(API_PATHS.AUTH.LOGIN, { ...form });
      const { token, user } = response.data;
      if (token) {
        localStorage.setItem("token", token);
        console.log("✅Token saved:", localStorage.getItem("token"));
        updateUserData(user)
        navigate("/home");
        toast.success("Login succesfully")
      }
    } catch (error) {
      if (error.response?.data?.message) {
        setError(error.response.data.message);
      } else {
        setError("Something went wrong. Please try again later.");
      }
    }
  };
  const handleToggle = () => {
    navigate("/signup");
  };
  return (
    <Authlayout>
      <div className='w-full h-full flex items-center justify-center'>
        <div className='lg:w-[80%] h-auto md:h-full flex flex-col justify-center'>
          <div className='flex flex-col gap-2'>
            <h1 className='text-5xl text-white font-[700]' style={{ fontFamily: 'futura' }}>Hey,</h1>
            <h1 className='text-4xl font-[700] text-white' style={{ fontFamily: 'futura' }}>Welcome Back!
�</h1>
            <p className='text-white' style={{ fontFamily: 'futura' }}>Login to track your expenses easily...
�</p>
          </div>
          <form onSubmit={handleSubmit}>
      <CustomInput
       placeHolder="danidaniels@gmail.com"
       label="Email"
       type="email"
       value={form.email}
       onChange={(e) => setForm({ ...form, email: e.target.value })}
       icons={<MdAlternateEmail />}
      />
      <CustomInput
       placeHolder=""
       type="password"
       label="Password"
       value={form.password}
       onChange={(e) => setForm({ ...form, password: e.target.value })}
       icons={<MdPassword />}
      />
      {error && <p className="text-red-500 text-sm mb-2">{error}</p>}
      <GeneralCusBtn
       type="submit"
       lefttIcon={<IoLogIn />}
       name="Welcome Back"
      />
      <p className="text-right mt-2" style={{ color: colors.primary, fontFamily: 'futura' }}>
       Forgot Password?
      </p>
      <GeneralCusBtn
       name="Sign in with Google"
       backgroundColors="#101010"
       lefttIcon={<FaGoogle />}
      />
</form>
     <h1 className="text-white text-center mt-7" style={{ fontFamily: 'futura' }}>
      Don't have an account?{" "}
      <span onClick={handleToggle} style={{ color: colors.primary, fontFamily: 'futura', cursor: 'pointer'
}}>
       Register
```

```
            </span>
          </h1>
        </div>
      </div>
    </Authlayout>
  );
};
export default SignIn;
```

4. Income page:



Income Overview

Track your earnings over time and analyze your income trends

Income Source

Freelancing
income
+ ₹150.0K
6/26/2025

Salary
income
+ ₹100.0K
6/26/2025

- **Income.jsx**

```
import React, { useState, useEffect } from 'react'; import DashboardLayout from
        '../../compents/layout/DashboardLayout'; import IncomeOverview from
        '../../compents/Income_components/IncomeOverview'; import axiosInstance from '../../utils/axiosInstance';
        import { API_PATHS } from '../../utils/apiPaths'; import Modal from '../../compents/modals/Modal'; import
        AddIncomeForm from '../../compents/Income_components/AddIncomeForm'; import toast from 'react-hot-
        toast'; import IncomeList from '../../compents/Income_components/IncomeList'; import { colors } from
        '../../constant/color'; import GeneralCusBtn from '../../compents/Inputs/GeneralCusBtn'; import { LuTrash2
        } from 'react-icons/lu';

const Income = () => { const [incomeData, setIncomeData] = useState([]); const [loading, setLoading] =
        useState(false); const [openIncomeModal, setOpenIncomeModal] = useState(false); const
        [openDeleteAlert, setOpenDeleteAlert] = useState({ show: false, data: null });

const fetchIncomeData = async () => { if (loading) return; setLoading(true); try { const response = await
        axiosInstance.get(API_PATHS.INCOME.GET_ALL_INCOME); if (response.data) { const sorted =
        [...response.data].sort((a, b) => new Date(b.date) - new Date(a.date)); setIncomeData(sorted); } } catch
        (error) { console.error("□ Failed to fetch income data:", error?.response?.data || error.message); } finally {
        setLoading(false); } };

useEffect(() => { fetchIncomeData(); }, []);

const handleAddIncome = async (income) => { const { catagory, amount, date } = income; if (!catagory) return
        toast.error("Category is needed"); if (!amount || isNaN(amount) || Number(amount) <= 0) return
        toast.error("Amount must be a valid number > 0"); if (!date) return toast.error("Date is required");

try {
          await axiosInstance.post(API_PATHS.INCOME.ADDINCOME, {
            catagory,
            amount: Number(amount),
            date,
          });
          toast.success("Income added successfully □");
          setOpenIncomeModal(false);
          fetchIncomeData();
        } catch (error) {
          console.error("□ Error adding income:", error?.response?.data || error.message);
          toast.error("Failed to add income. Please try again.");
        }

};

const handleDownloadData = () => { toast.success("Download clicked"); };

const handleDeleteClick = (item) => { setOpenDeleteAlert({ show: true, data: item }); };
const confirmDelete = async () => { const id = openDeleteAlert.data?._id; if (!id) return;

try {
          await axiosInstance.delete(API_PATHS.INCOME.DELETE_INCOME(id));
          toast.success("Transaction deleted successfully");
          setOpenDeleteAlert({ show: false, data: null });
          fetchIncomeData();
        } catch (error) {
          console.error("Failed to delete:", error?.response?.data?.message || error?.message);
          toast.error("Failed to delete transaction");
        }


};

return (

<IncomeOverview addIncome={() => setOpenIncomeModal(true)} data={incomeData} loading={loading} />

  <Modal
              shorty="Seamless Tracking with FinSight"
          isOpen={openIncomeModal}
        onClose={() => setOpenIncomeModal(false)}
              title="Add Income">
              <AddIncomeForm onSubmit={handleAddIncome} />
            </Modal>
          </div>
          <div className="mt-5 w-full grid md:grid-cols-1 gap-3">
            <div style={{ backgroundColor: "#131313" }} className="flex rounded-lg py-4 px-5 flex-col gap-4">
              <IncomeList
                onDownload={handleDownloadData}
                onDelete={handleDeleteClick}
                transaction={incomeData}
              />
            </div>
          </div>
          <Modal
            isOpen={openDeleteAlert.show}
            onClose={() => setOpenDeleteAlert({ show: false, data: null })}
            title="Delete Income"
            titleColor="text-red-400"
            shorty={
              openDeleteAlert.data
                ? Are you sure you want to delete "${openDeleteAlert.data.catagory}" transaction?
                : "Are you sure you want to delete this transaction?"
            }
          >
            <GeneralCusBtn
              lefttIcon={<LuTrash2 size={20} color={colors.white} />}
              name={"Delete Transaction"}
              onClick={confirmDelete}
              backgroundColors={colors.rose}
            />
          </Modal>
        </DashboardLayout>

); };

export default Income;
```

- **IncomeController.js:**

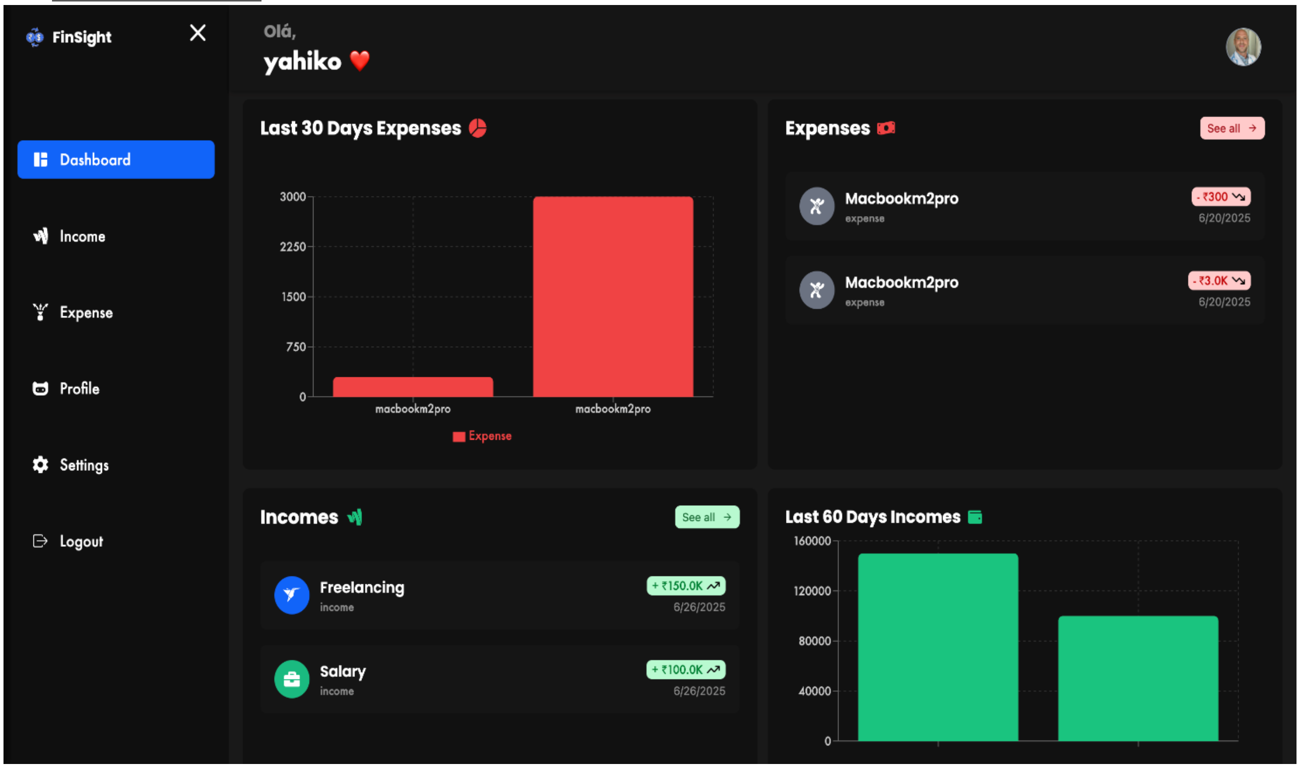
```
import jwt from "jsonwebtoken" import User from "../models/User.js" import Income from "../models/Income.js"
    import path from 'path'; import fs from 'fs'; import ExcelJS from 'exceljs'; import { fileURLToPath } from
    "url"; const __filename = fileURLToPath(import.meta.url); const _dirname = path.dirname(_filename);
    export const addIncome = async (req, res) => { const userId = req.user.id;

    try { const { icon, catagory, amount, date } = req.body;

}if (!catagory || !amount || !date) {
     return res.status(400).json({ message: "All fields requiredd" });

    const newIncome = new Income({
     userId,
     icon,
     catagory,
     amount,
     date: new Date(date),
    });
    await newIncome.save();
    res.status(200).json(newIncome);

} catch (error) { console.error("Error adding income:", error); // □ log the actual error res.status(500).json({
    message: "Server Error" }); } };

export const getAllIncome = async (req,res) => { const userId = req.user.id;

try {
      const income = await Income.find({userId}).sort({date: -1});
      res.json(income);
    }catch(error) {
      console.log("Error",error);
      return res.status(500).json({message: "Server Errorssss mother fucker..."});
    }


}
```

```
export const deleteIncome = async (req, res) => { try {

const deleted = await Income.findByIdAndDelete(req.params.id);
        if (!deleted) {
          return res.status(404).json({ message: "Income not found" });
        }
        return res.json({ message: "Successfully deleted the income" });

} catch (error) { console.log("Error in delete route:", error); return res.status(500).json({ message: "Server error
        while deleting income" }); } };

export const downloadIncomeXL = async (req, res) => { const userId = req.user.id;

try { const income = await Income.find({ userId }).sort({ date: -1 });

const workbook = new ExcelJS.Workbook();
        const worksheet = workbook.addWorksheet('Income Report');
        worksheet.columns = [
          { header: 'Category', key: 'catagory', width: 25 },
          { header: 'Amount', key: 'amount', width: 15 },
          { header: 'Date', key: 'date', width: 20 }
        ];
        income.forEach((item) => {
          worksheet.addRow({
            catagory: item.catagory,
            amount: item.amount,
            date: item.date.toISOString().split('T')[0]
          });
        });
        const downloadsDir = path.join(__dirname, '../IncomeDownload');
        if (!fs.existsSync(downloadsDir)) {
          fs.mkdirSync(downloadsDir, { recursive: true });
        }
        const filePath = path.join(downloadsDir, income-report-${userId}.xlsx);
        await workbook.xlsx.writeFile(filePath);

        } catch (error) { console.error('Error saving Excel file:', error); res.status(500).json({ message: 'Server
        error: download failed' });

        }

 };
```

❖ **INCOME.js (income model):**

```
import mongoose from "mongoose";

const IncomeSchema = new mongoose.Schema({

    userId: {type: mongoose.Schema.ObjectId, ref: "User", required: true},
    icon: {type: String},
    catagory: {type: String, required: true},
    amount: {type: Number, required: true},
    date: {type: Date, default: Date.now()}
    },
    {timestamps: true}
)
const Income = mongoose.model("Income",IncomeSchema);
export default Income;
```

5. Dashboard :-



✓ Code

import React, { useState } from 'react'; import NavDash from '../dashboardComp/NavDash'; import SideMenu from '../dashboardComp/SideMenu';

const DashboardLayout = ({ children }) => { const [isSidebarOpen, setIsSidebarOpen] = useState(true);

return (

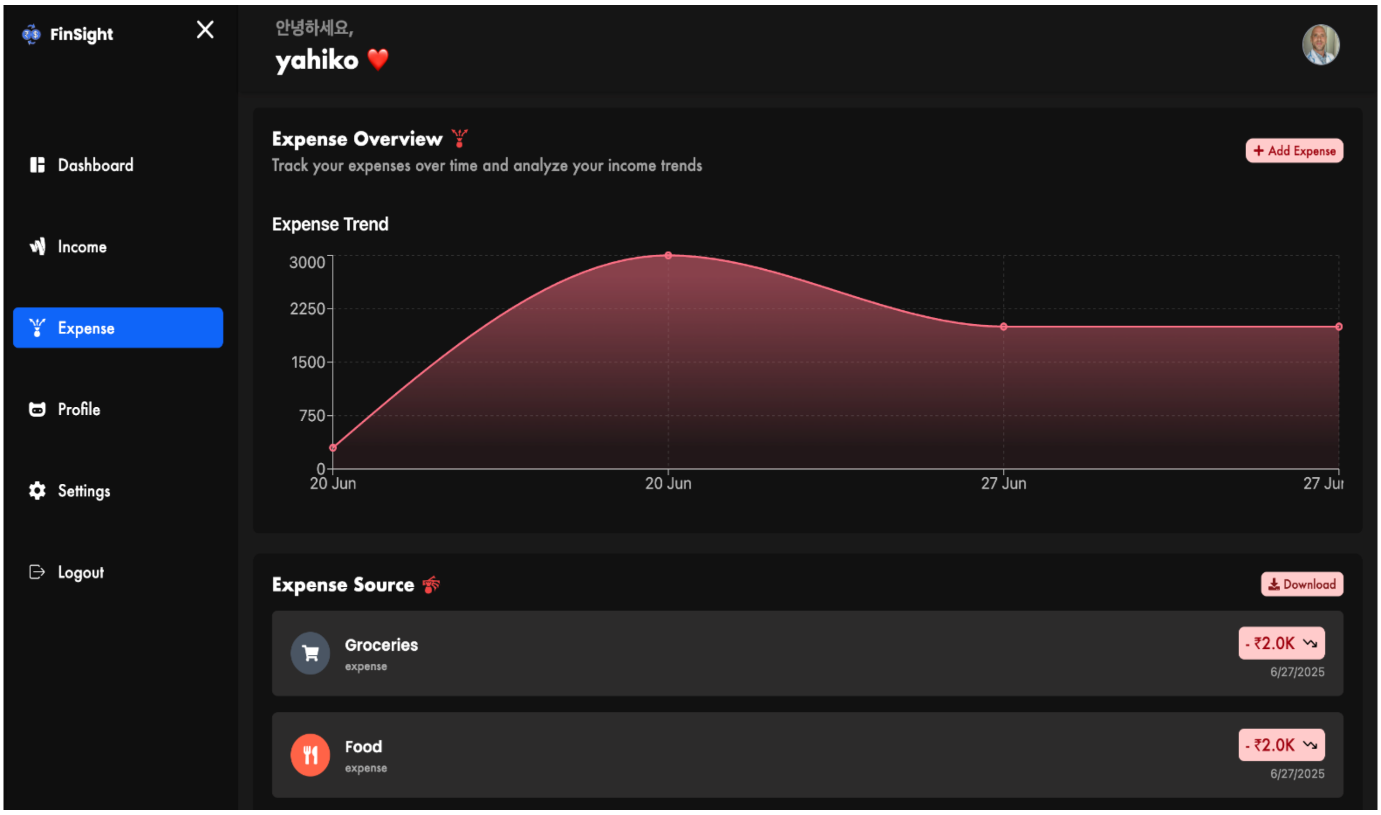
```
{/* Sidebar */} <div className={transition-all duration-300 ease-in-out ${ isSidebarOpen ? 'w-64' : 'w-16' } h-
        full} style={{ backgroundColor: '#111111' }} > <SideMenu isOpen={isSidebarOpen} toggle={() =>
        setIsSidebarOpen(!isSidebarOpen)} />

 {/* Main Content Area */}
          <div className="flex flex-col flex-1 h-full">
            {/* Navbar */}
            <NavDash />
            {/* Scrollable Content with hidden scrollbar */}
            <div className="flex-1 overflow-y-auto p-4 bg-[#1a1a1a] hide-scrollbar">
              {children}
            </div>
          </div>
        </div>

); };

export default DashboardLayout;
```

- ✓ **<u>DashboardController:</u>**

```
import Expense from "../models/Expense.js"; import Income from "../models/Income.js";

import { isValidObjectId, Types } from "mongoose";

export const getDashBoardData = async (req, res) => { try { const userID = req.user.id;

// □ Validate ObjectId
        if (!isValidObjectId(userID)) {
          return res.status(400).json({ message: "Invalid user ID" });
        }
        const userObjectId = new Types.ObjectId(String(userID));
        // □ Total Income
        const totalIncomeResult = await Income.aggregate([
          { $match: { userId: userObjectId } },
          { $group: { _id: null, total: { $sum: "$amount" } } }
        ]);
        const totalIncome = totalIncomeResult[0]?.total || 0;
        // □ Total Expenses
        const totalExpensesResult = await Expense.aggregate([
          { $match: { userId: userObjectId } },
          { $group: { _id: null, total: { $sum: "$amount" } } }
        ]);
        const totalExpenses = totalExpensesResult[0]?.total || 0;

        /* // □ Check: Expenses > Income if (totalExpenses > totalIncome) { return res.status(400).json({
        message: "You don’t have enough balance to spend.
        Your expenses exceed your income.", totalIncome, totalExpenses, totalBalance: totalIncome -
        totalExpenses }); } */ // □ Last 60 Days Income const last60DaysIncomeTransaction = await
        Income.find({ userId: userObjectId, date:

        { $gte: new Date(Date.now() - 60 * 24 * 60 * 60 * 1000) } }).sort({ date: -1 });

const incomeLast60Days = last60DaysIncomeTransaction.reduce(
          (sum, txn) => sum + txn.amount,
          0
        );

        // □ Last 30 Days Expenses
        const last30DaysExpenseTransaction = await Expense.find({  userId: userObjectId,
          date: { $gte: new Date(Date.now() - 30 * 24 * 60 * 60 * 1000) }
        }).sort({ date: -1 });

        const expenseLast30days = last30DaysExpenseTransaction.reduce(
          (sum, txn) => sum + txn.amount,
          0
        );
        // □ Last 5 Combined Transactions
        const lastTransaction = [
          ...(await Income.find({ userId: userObjectId }).sort({ date: -1 }).limit(5)).map(
            (txn) => ({
              ...txn.toObject(),
              type: "income"
            })
          ),
          ...(await Expense.find({ userId: userObjectId }).sort({ date: -1 }).limit(5)).map(
            (txn) => ({
              ...txn.toObject(),
              type: "expense"
            })
          )
        ].sort((a, b) => new Date(b.date) - new Date(a.date));

        // □ Final Response
        return res.json({
          totalBalance: totalIncome - totalExpenses,
          totalIncome,
          totalExpenses,
          last30DaysExpenses: {
            total: expenseLast30days,
            transaction: last30DaysExpenseTransaction
          },
          last60DaysIncomes: {
            total: incomeLast60Days,
            transaction: last60DaysIncomeTransaction
          },
            recentTransaction: lastTransaction
          });

} catch (error) { console.error("Dashboard Error:", error);

return res.status(500).json({ message: "Server Error: Failed to fetch dashboard data" });

 }

};
```

---

6 . **Expenses:**



✓ **CODE**

```
import React, { useState, useEffect } from 'react'; import DashboardLayout from
        '../../compents/layout/DashboardLayout'; import ExpenseOverView from
        '../../compents/expense_comp/ExpenseOverView'; import axiosInstance from
        '../../utils/axiosInstance'; import { API_PATHS } from '../../utils/apiPaths'; import Modal from
        '../../compents/modals/Modal'; import AddExpenseForm from
        '../../compents/expense_comp/AddExpenseForm'; import toast from 'react-hot-toast'; import
        ExpenseList from '../../compents/expense_comp/ExpenseList'; import GeneralCusBtn from
        '../../compents/Inputs/GeneralCusBtn'; import { colors } from '../../constant/color'; import {
        LuTrash2 } from 'react-icons/lu';
```

```
const Expense = () => { const [expenseData, setExpenseData] = useState([]); const [loading, setLoading] =
          useState(false); const [openExpenseModal, setOpenExpenseModal] = useState(false); const
          [openDeleteAlert, setOpenDeleteAlert] = useState({ show: false, data: null });

// Fetch all expenses const fetchExpenseData = async () => { if (loading) return; setLoading(true); try {
          const response = await axiosInstance.get(API_PATHS.EXPENSE.GET_ALL_EXPENSE); if
          (response.data) { const sorted = [...response.data] .sort((a, b) => new Date(b.date) - new
          Date(a.date)) .map(item => ({ ...item, id: item._id })); // Ensure consistent id
          setExpenseData(sorted); } } catch (error) { console.error('Failed to fetch expenses:', error); }
          finally { setLoading(false); } };

useEffect(() => { fetchExpenseData(); }, []);

// Handle new expense addition const handleAddExpense = async (expense) => { const { category,
          amount, date } = expense;

if (!category) return toast.error('Category is required');
          if (!amount || isNaN(amount) || Number(amount) <= 0)
            return toast.error('Enter a valid amount greater than 0');
          if (!date) return toast.error('Date is required');
          try {
            await axiosInstance.post(API_PATHS.EXPENSE.ADD_EXPENSE, {
              category,
              amount: Number(amount),
              date,
            });
            toast.success('Expense successfully added');
            setOpenExpenseModal(false);
            fetchExpenseData();
          } catch (error) {
            console.error('Error adding expense:', error?.response?.data || error?.message);
            toast.error('Failed to add expense');
          }

};

// Trigger delete confirmation modal const handleDelete = (item) => { setOpenDeleteAlert({ show: true,
          data: item }); };

// Confirm delete action const confirmDelete = async () => { const id = openDeleteAlert.data?.id; if (!id)
          return;

try {
            await axiosInstance.delete(API_PATHS.EXPENSE.DELETE_EXPENSE(id));
            toast.success('Deleted the transaction successfully');
            setOpenDeleteAlert({ show: false, data: null });
            fetchExpenseData();
          } catch (error) {
            console.error('Failed to delete the data:', error?.response?.data?.message || error?.message);
```

```
              toast.error('Failed to delete');
            }


};

// Placeholder for download const handleDownload = async () => { // implement if needed };

return ( {/* Overview and Add Expense Modal */}

<ExpenseOverView addExpense={() => setOpenExpenseModal(true)} data={expenseData} />

<Modal title="Add Expense" shorty="Seamless track with FinSight" isOpen={openExpenseModal}
        onClose={() => setOpenExpenseModal(false)} >

 {/* Expense List */}
            <div className="mt-5 w-full grid md:grid-cols-1 gap-3">
              <ExpenseList
                transaction={expenseData}
              onDelete={handleDelete}
                onDownload={handleDownload}
              />
            </div>

            {/* Delete Confirmation Modal */}
            <Modal
              isOpen={openDeleteAlert.show}
              onClose={() => setOpenDeleteAlert({ show: false, data: null })}
              title="Delete Expense"
              shorty="Are you sure you want to delete this transaction?"
              titleColor="text-red-400"
            >
              <GeneralCusBtn
                backgroundColors={colors.rose}
                lefttIcon={<LuTrash2 color={colors.white} />}
                name="Delete Transaction"
                onClick={confirmDelete}
              />
            </Modal>
          </DashboardLayout>


);

};

export default Expense;
```

✓ **ExpenseController:**

```
import Income from "../models/Income.js"; import Expense from "../models/Expense.js"; import User from
        "../models/User.js"; import path from 'path'; import fs from 'fs'; import ExcelJS from 'exceljs'; import {
        fileURLToPath } from "url"; const __filename = fileURLToPath(import.meta.url); const _dirname =
        path.dirname(_filename); export const addExpense = async (req,res) => { const userId = req.user.id; try {
        const { icon, category, amount, date } = req.body; if(!category || !amount || !date) { return
        res.status(400).json({message: "are you dumb fucker all fields is requird"}); } const expense = new
        Expense({ userId, icon, category, amount, date }); await expense.save(); res.status(200).json({message:
        "new Expense is added"});

        }catch(error) {
                console.log("Server Error",error);
                return res.status(500).json({message: "server failed expense is not added"})

}

}

export const getAllExpense = async (req,res) => { const userId = req.user.id;

try {
          const allExpensse = await Expense.find({userId}).sort({date: -1})
          res.json(allExpensse)

         } catch (error) {
          console.log("Server Error failed to Show all expense",error)
         }

}

export const delectExpense = async (req,res) => { try { const deleteEx = await
        Expense.findByIdAndDelete(req.params.id).sort({date: -1}); if(!deleteEx) return
        res.status(401).json({message: "expense is not exxist"}); res.json({message: "Expense is deleted
        successfully"}); } catch (error) { return res.status(500).json({message: "expense is not deleted, server
        failed"}); } }

export const downloadExpenseXL = async (req, res) => { const userId = req.user.id;

try { const expense = await Expense.find({ userId }).sort({ date: -1 });

const workbook = new ExcelJS.Workbook();
        const worksheet = workbook.addWorksheet('Expense Report');

        worksheet.columns = [
          { header: 'Category', key: 'category', width: 25 },
          { header: 'Amount', key: 'amount', width: 15 },
          { header: 'Date', key: 'date', width: 20 },
        ];
        expense.forEach((item) => {
          worksheet.addRow({
            category: item.category,
            amount: item.amount,
            date: item.date.toISOString().split('T')[0],
          });
        });

        const downloadsDir = path.join(__dirname, '../ExpenseDownload');
        if (!fs.existsSync(downloadsDir)) {
          fs.mkdirSync(downloadsDir, { recursive: true });
        }

        const filePath = path.join(downloadsDir, expense-report-${userId}.xlsx);
        await workbook.xlsx.writeFile(filePath);

        res.download(filePath);

} catch (error) { console.error('Error saving Excel file:', error);

 res.status(500).json({ message: 'Server error: download failed' });

         }

};
```

- ✓ **Expense.js(expenseModel):**

```
import mongoose from "mongoose";


const ExpenseSchema = new mongoose.Schema({

    userId: {type: mongoose.Schema.ObjectId,ref: "User",required: true},

    icon: {type: String},

    category: {type: String, required: true},

    amount: {type: Number, required: true},

    date: {type: Date, default: Date.now()}

},

   {timestamps: true}

)

const Expense = mongoose.model("Expense",ExpenseSchema);

export default Expense;
```

✓ **<u>authController.js auth:</u>**

```
import jwt from "jsonwebtoken";
import User from "../models/User.js";
import { use } from "react";
import { token } from "morgan";
const generateToken = (id) => {
  return jwt.sign({ id }, process.env.JWT_SECRET, { expiresIn: "1h" });
};

export const registerUser = async (req, res) => {
  // Debugging logs`
  console.log("Headers:", req.headers);
  console.log("Raw body:", req.body);

  if (!req.body || Object.keys(req.body).length === 0) {
    return res.status(400).json({
      message: "Request body is empty",
      receivedHeaders: req.headers,
      expected: "Content-Type: application/json with valid JSON body"
    });
  }
  const { name, email, password, profilePic } = req.body;
  if (!name || !email || !password) {
    return res.status(400).json({
      message: "All fields are required",
      received: { name, email, password }
    });
  }
  try {
    const existUser = await User.findOne({ email });
    if (existUser) {
      return res.status(400).json({ message: "Email already exists" });
    }
    const user = await User.create({ name, email, password, profilePic });
    res.status(201).json({
      id: user._id,
      user,
      token: generateToken(user._id),
    });
  } catch (error) {
    console.error("Registration error:", error);
    res.status(500).json({
      message: "Registration failed",
      error: error.message
    });
  }
};
export const loginUser = async (req, res) => {

  if (!req.body || Object.keys(req.body).length === 0) {
    return res.status(400).json({
      message: "Request body is empty",
      receivedHeaders: req.headers,
      expected: "Content-Type: application/json with valid JSON body"
    });
  }
   const {email,password} = req.body;
   if(!email || !password) {
    return res.status(400).json({message: "Fill all the fields fucker"})
   }

   try {
    const user = await User.findOne({email});
   if(!user || !(await user.comparePassword(password))) {
     return res.status(400).json({message: "Password not matched"})
   }

   res.status(201).json({
      id: user._id,
      user,
      token: generateToken(user._id)
   })
   } catch (error) {
      console.log("Login Error",error);
      res.status(500).json({
        message: "Login Failed",
        error: error.message
      })
   }
};

export const getUser = async (req, res) => {
    try {
      const user = await User.findById(req.user.id).select("-password");
      if(!user) {
         res.status(404).json({message: "No created yet"})
      }
      res.status(200).json(user)
    } catch (error) {
      console.log("User error", error)
      res.status(500).json({message: "No user", error: error.message})
    }
};
```

- ✓ **AuthMiddleware**

```
import jwt from "jsonwebtoken"
        import User from "../models/User.js"

        export const protect = async (req,res,next) => {
          let token = req.headers.authorization?.split(" ")[1];
          if(!token) {
            return res.status(401).json({message: "Not authorizeddd"})
          }
          try{
            const decode = jwt.verify(token,process.env.JWT_SECRET);
            req.user = await User.findById(decode.id).select("-password")
            next()
          } catch(error) {
             res.status(500).json({message: "not auth failed"})
          }
        };
```

- ✓ **USER.js:**

```
import mongoose from "mongoose";
          import bcrypt from "bcryptjs";

          // Define the schema
          const UserSchema = new mongoose.Schema(
            {
              name: {
                type: String,
                required: [true, "Name is required"],
                trim: true,
              },
    email: {
      type: String,
      required: [true, "Email is required"],
      unique: true,
      lowercase: true,
    },
    password: {
      type: String,
      required: [true, "Password is required"],
      minlength: 6, // Optional: basic password validation
    },
    profilePic: {
      type: String,
      default: null,
    },
  },
  {
    timestamps: true,
  }
);

// □ Hash password before saving to DB
UserSchema.pre("save", async function (next) {
  if (!this.isModified("password")) return next();
  try {
    const salt = await bcrypt.genSalt(10);
    this.password = await bcrypt.hash(this.password, salt);
    next();
  } catch (err) {
    next(err);
  }
```

```
});

// □ Add password comparison method
UserSchema.methods.comparePassword = async function (enteredPassword) {
  return await bcrypt.compare(enteredPassword, this.password);
};

// Export the model
const User = mongoose.model("User", UserSchema);
```

```
export default User;
```

✓ **Server.js:**

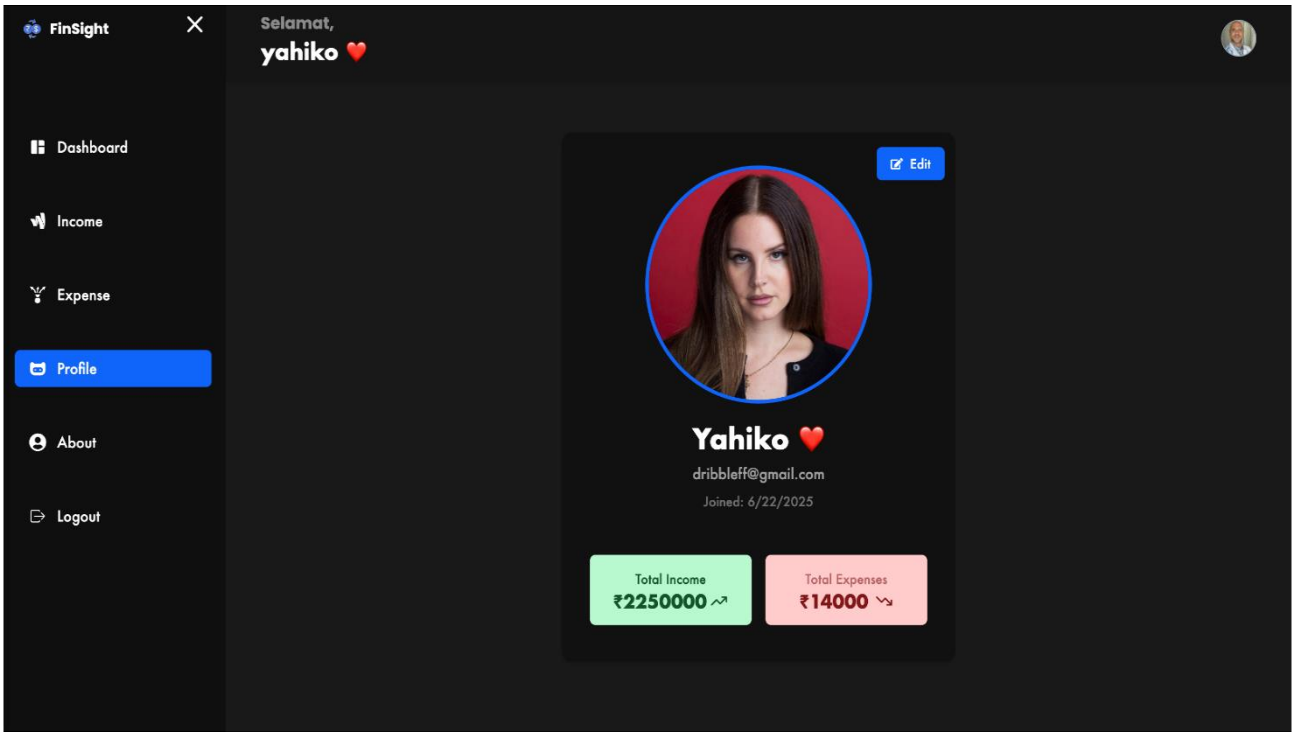
```
import express from "express"; import dotenv from "dotenv"; import cors from "cors"; import path
        from "path"; import { fileURLToPath } from "url";

import cookieParser from "cookie-parser"; import morgan from "morgan";

import authRoutes from "./routes/authRoutes.js"; import incomeRoute from
        "./routes/incomeRoutes.js"; import expenseRoute from "./routes/expenseRoute.js"; import
        dashBoardRoute from "./routes/dashboardRoute.js"; import { connectDb } from
        "./config/db.js";

dotenv.config();

const app = express(); const __filename = fileURLToPath(import.meta.url); const _dirname =
        path.dirname(_filename);

// Connect Database connectDb();

// Enable CORS before routes app.use(cors({ origin: process.env.FRONTEND_URL ||
        "http://localhost:5173", credentials: true, }));

// Middleware app.use(express.json()); app.use(express.text()); app.use(cookieParser());
        app.use(morgan("dev"));

// Optional: Handle plain text bodies pretending to be JSON app.use((req, res, next) => { if
        (req.is('text/plain')) { try { req.body = JSON.parse(req.body); } catch (err) { return
        res.status(400).json({ error: "Invalid JSON in plain text body" }); } } next(); });

// API Routes app.use("/api/v1/auth", authRoutes); app.use("/api/v1/income", incomeRoute);
        app.use("/api/v1/expense", expenseRoute); app.use("/api/v1/dashboard", dashBoardRoute);

// Serve static files (uploads) app.use("/api/v1/auth", express.static(path.join(__dirname, "uploads")));
```

// Start Server const PORT = process.env.PORT || 5000; app.listen(PORT, () => { console.log(□ Server running on port ${PORT});

}

);

1. **Profile page:**



✓ *Code:*

import React, { useContext, useEffect, useState } from 'react'; import DashboardLayout from '../../compents/layout/DashboardLayout'; import { images } from '../../constant/images'; import { UserContext } from '../../context/useContext'; import { FaEdit } from "react-icons/fa"; import { colors } from '../../constant/color'; import axiosInstance from '../../utils/axiosInstance'; import { API_PATHS } from '../../utils/apiPaths'; import { useNavigate } from 'react-router-dom'; import { LuTrendingDown, LuTrendingUp } from 'react-icons/lu'; import Modal from '../../compents/modals/Modal'; import AvatarWithEdit from '../../compents/avatar/AvatarWithEdit'; import CustomInput from '../../compents/Inputs/CustomInput'; import GeneralCusBtn from '../../compents/Inputs/GeneralCusBtn'; import { MdTipsAndUpdates } from "react-icons/md"; import { toast } from 'react-hot-toast';

const Profile = () => { const { user, updateUserData } = useContext(UserContext); const [data, setData] = useState({}); const [loading, setLoading] = useState(false); const [openModal, setOpenModal] = useState(false);

// Editable fields const [name, setName] = useState(user?.name || "");

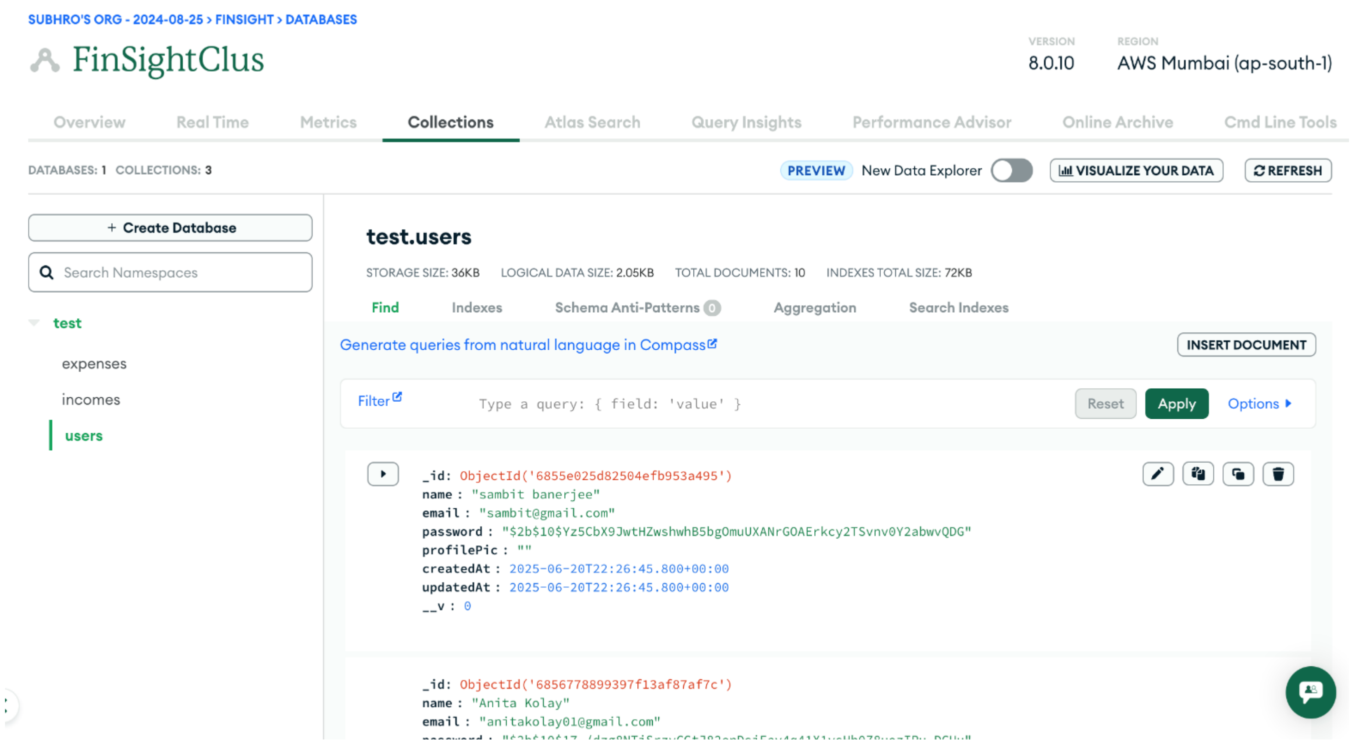
const navigate = useNavigate();

```
const fetchData = async () => { if (loading) return; setLoading(true); try { const response = await
axiosInstance.get(API_PATHS.DASHBOARD.GET_DATA); if (response.data) { setData(response.data); }
} catch (error) { console.log("Data not found", error?.response?.data?.message || error?.message); } finally
{ setLoading(false); } };

useEffect(() => { fetchData(); }, []);

const userStats = { totalExpenses: data?.totalExpenses ? ₹${data.totalExpenses} : '₹0', totalIncome:
data?.totalIncome ? ₹${data.totalIncome} : '₹0', joined: user?.createdAt ? new
Date(user.createdAt).toLocaleDateString() : 'N/A', };

const handleMoal = () => { setOpenModal(true); };

const handleUpdateProfile = async () => { try { const res = await
axiosInstance.put(API_PATHS.UPDATE.UPDATE_PROFILE, { name }); if (res.data?.user) {
updateUserData(res.data.user); toast.success("Profile updated"); setOpenModal(false); } } catch (error) {
console.log(error); toast.error("Failed to update profile"); } };

if (!user) { return (

Loading user profile...

); }

return ( <div style={{ fontFamily: "futura" }} className="w-full flex justify-center mt-10 px-4">

<AvatarWithEdit initialImage={user?.profilePic || images.logo} />

      <div className="text-center space-y-2">
        <h1 className="text-white text-3xl font-semibold capitalize">{user.name} □</h1>
        <p className="text-white text-md opacity-60">{user.email}</p>
        <p className="text-white text-sm opacity-40">Joined: {userStats.joined}

</p>
      </div>

      <div className="w-full grid grid-cols-2 gap-4 mt-6">
        <div onClick={() => navigate("/income")} className="px-4 flex flex-col

items-center justify-center py-3 bg-green-200 rounded-md text-center">
          <p className="text-sm text-green-900 mt-1">Total Income</p>
          <h2 className=" text-xl text-green-900 flex items-center gap-1

font-bold">{userStats.totalIncome} <LuTrendingUp /></h2>
        </div>
        <div onClick={() => navigate("/expense")} className="px-4 flex flex-col

items-center justify-center py-3 rounded-md bg-red-200 text-center">
                  <p className="text-sm text-red-900 opacity-60 mt-1">Total Expenses</p>
                  <h2 className="text-red-900 text-xl flex items-center gap-2 font-
            bold">{userStats.totalExpenses} <LuTrendingDown /></h2>
```

```
                    </div>
                  </div>
                </div>

                <Modal
                  isOpen={openModal}
                  onClose={() => setOpenModal(false)}
                  title="Edit Profile"
            shorty={"Update your personal details"}>
                  <CustomInput
                    value={name}
                    onChange={(e) => setName(e.target.value)}
                    label={"Name"} />
                  <GeneralCusBtn
                    onClick={handleUpdateProfile}
                    name={"Update Details"}
                    lefttIcon={<MdTipsAndUpdates color='white' />}

                         />
        </Modal>
      </div>
    </DashboardLayout>
  ); }; export default Profile
```

❖ BACKEND:-

1) USER DATA:


SUBHRO'S ORG - 2024-08-25 > FINSIGHT > DATABASES

FinSightClus

VERSION 8.0.10 REGION AWS Mumbai (ap-south-1)

DATABASES: 1 COLLECTIONS: 3

test
- expenses
- incomes
- users

test.users

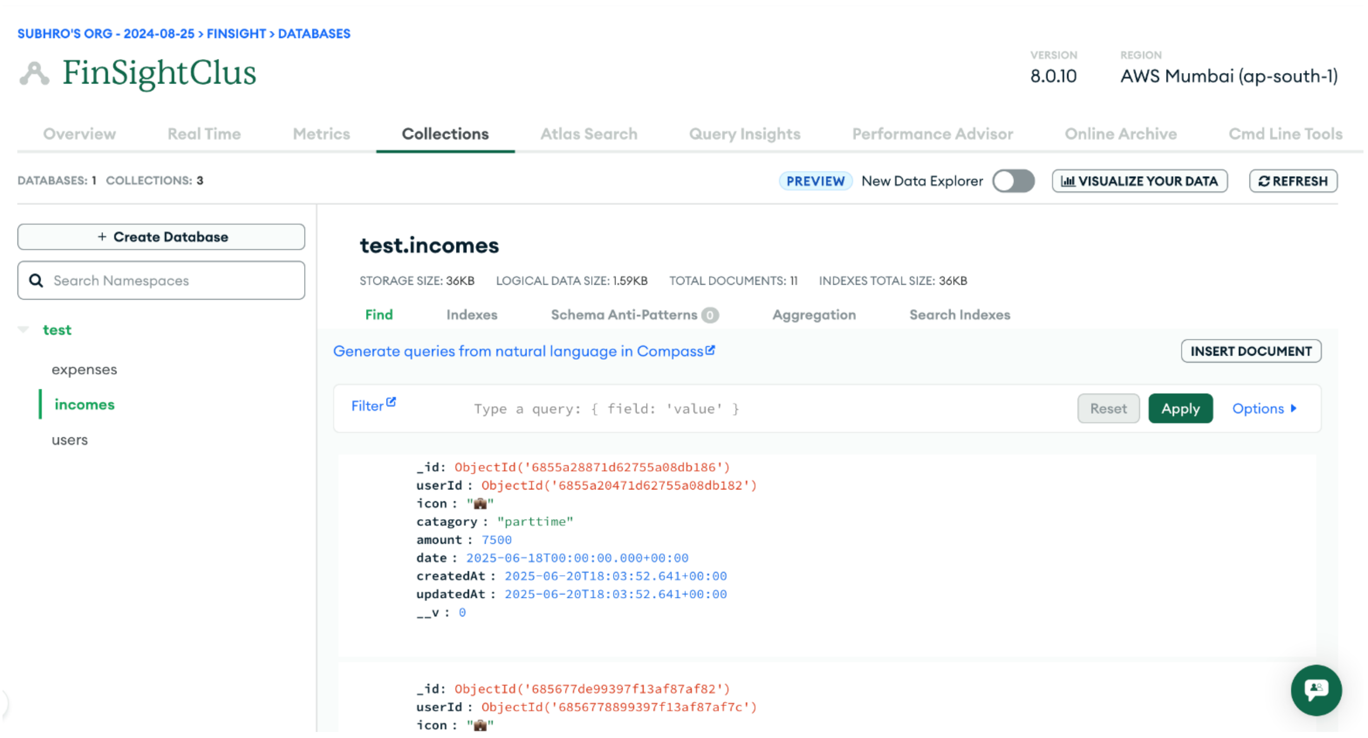
STORAGE SIZE: 36KB LOGICAL DATA SIZE: 2.05KB TOTAL DOCUMENTS: 10 INDEXES TOTAL SIZE: 72KB

```
_id: ObjectId('6855e025d82504efb953a495')
name : "sambit banerjee"
email : "sambit@gmail.com"
password : "$2b$10$Yz5CbX9JwtHZwshwhB5bgOmuUXANrGOAErkcy2TSvnv0Y2abwvQDG"
profilePic : ""
createdAt : 2025-06-20T22:26:45.800+00:00
updatedAt : 2025-06-20T22:26:45.800+00:00
__v : 0

_id: ObjectId('6856778899397f13af87af7c')
name : "Anita Kolay"
email : "anitakolay01@gmail.com"
```

- Database - db.js:

```
import mongoose from "mongoose";

export const connectDb = async () => {
    try {
      await mongoose.connect(process.env.MONGODB_URL,{})
      console.log("✅ Database is connected successfullly")
    } catch (error) {
      console.log("DB is not connected",error)
      process.exit(1)

    }
}
```

2) INCOMES DATA:



SUBHRO'S ORG - 2024-08-25 > FINSIGHT > DATABASES

FinSightClus

VERSION 8.0.10 REGION AWS Mumbai (ap-south-1)

DATABASES: 1 COLLECTIONS: 3

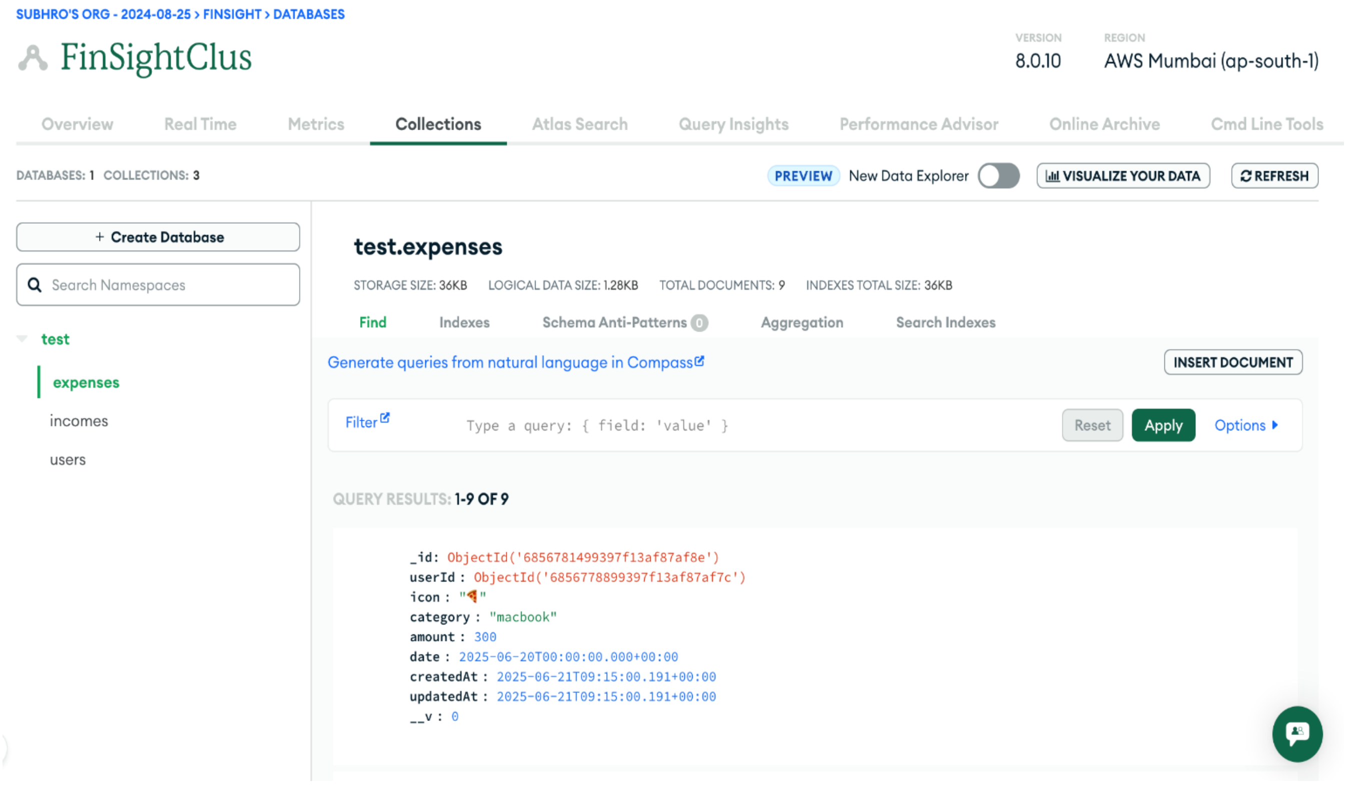
test
- expenses
- incomes
- users

test.incomes

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.59KB TOTAL DOCUMENTS: 11 INDEXES TOTAL SIZE: 36KB

```
_id: ObjectId('6855a28871d62755a08db186')
userId : ObjectId('6855a20471d62755a08db182')
icon : "💼"
catagory : "parttime"
amount : 7500
date : 2025-06-18T00:00:00.000+00:00
createdAt : 2025-06-20T18:03:52.641+00:00
updatedAt : 2025-06-20T18:03:52.641+00:00
__v : 0

_id: ObjectId('685677de99397f13af87af82')
userId : ObjectId('6856778899397f13af87af7c')
icon : "💼"
```

3) EXPENSES DATA:



```
_id: ObjectId('6856781499397f13af87af8e')
userId : ObjectId('6856778899397f13af87af7c')
icon : "🍕"
category : "macbook"
amount : 300
date : 2025-06-20T00:00:00.000+00:00
createdAt : 2025-06-21T09:15:00.191+00:00
updatedAt : 2025-06-21T09:15:00.191+00:00
__v : 0
```

CONCLUSION

The **Automated Expenses Recording and Monitoring System** successfully addresses the need for an efficient, user-friendly platform to manage personal or organizational financial records. By automating the process of recording daily transactions and providing real-time monitoring features, this system reduces manual effort, minimizes human errors, and ensures data accuracy. The integration of modern technologies like MongoDB, React, Node.js, and Express.js has enabled the system to offer scalability, security, and a responsive user experience.

This project not only streamlines expense management but also empowers users with insights into their spending habits, allowing for better financial planning and control. With potential for future enhancements such as analytics dashboards, mobile integration, and budget alerts, this system lays a strong foundation for a comprehensive digital financial assistant.

FUTURE SCOPE & FURTHER ENHANCEMENTS

❖ Future Scope

The **Automated Expenses Recording and Monitoring System** has significant potential for future development and enhancements. Some of the possible improvements and extensions include:

- **Mobile App**
  - A mobile version can be made so users can manage expenses from their phone.
- **Charts and Graphs**
  - Add simple charts to show where and how money is spent.
- **Notification System**
  - Send alerts for high spending or bill reminders.
- **Data Export**
  - Allow users to download their expense data as PDF or Excel.

❑ **Multi-User Support**

- ♦ Add feature for more than one user, useful for families or small teams.

❑ **Language Options**

- ♦ Support different languages to help more users.

❑ **Cloud Backup**

- ♦ Store data online so it's safe and always available.

❖ *Further enhancement:-*

In the future, some more useful features can be added to make the system better:

❑ **Login with OTP or Fingerprint**

- ♦ More secure ways to log in, like using fingerprint.

❑ **Monthly Budget Setting**

- ♦ Users can set a monthly budget and get alerts when they are close to the limit.

❑ **Automatic Expense Detection**

- ♦ The system can detect and record expenses from SMS or bank messages.

❑ **Dark Mode and Custom Themes**

- ♦ Add theme options to improve user experience and comfort.

❑ **Income Tracking**

- ♦ Not just expenses, but also track income sources.

❑ **Sharing Reports**

- ♦ Users can share their expense report with others (family, accountant, etc.).

❑ **Search and Filter Options**

- ♦ Easily find any expense using search or filter.

8. BIBLIOGRAPHY

https://reactjs.org

https://nodejs.org

https://expressjs.com

https://www.mongodb.com

https://www.youtube.com/watch?v=T55Kb8rrH1g&list=PLbtI3_MArDOkXRLxdMt1NOMtCS-84ibHH