IMARTICUS

Java Full Stack Development

Team Members

Name: K.Krithika
Name:A.Asritha
Register Number: 23BCE20160 **Register Number:**23BCE8404

Project Name: Expense Tracker

Webiste Name: ExpenseVault

PROJECT OVERVIEW

Brief Description:

ExpenseVault is a web-based application designed to help users manage their personal finances efficiently. It enables users to set an initial balance, record daily expenses with categories and notes, and monitor their remaining balance in real-time. The application also includes a transaction history and sorting options for better analysis and budgeting.

Objective:

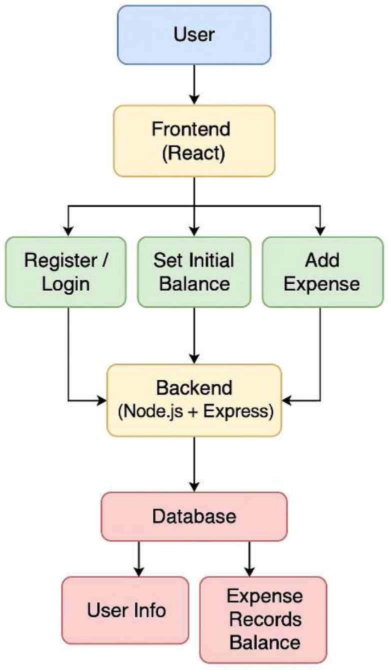
To provide a simple and intuitive platform for individuals to track and control their spending habits and make smarter financial decisions.

Key Functionalities:

- User registration and login with secure authentication
- Set an initial balance after signup
- Add expenses with title, amount, date, and category
- Auto-deduction of expenses from the current balance
- View and manage transaction history
- Sort expenses by date or amount

Technology Overview:

- Frontend: React.js, Axios
- Backend: Node.js, Express.js, MongoDB
- Deployment: Vercel (Frontend), Render (Backend), JWT



Frontend Functionality

1. User Authentication Interface

- Provides login and signup forms
- Includes password visibility toggles
- Handles form validation and user input

2. Initial Balance Input

- After successful signup/login, users are prompted to enter their initial bank balance
- This value is sent to the backend and stored for further expense calculations

3. Add Expense Form

- Users can add expenses by specifying:
- Title/Note
- Amount
- Date
- Category (via dropdown)
- Upon submission, the expense is sent to the backend and deducted from the current balance

4. Balance and Transaction Display

- Displays the current available balance dynamically
- Shows transaction history in reverse chronological order (latest first)

5. Sorting and Organization

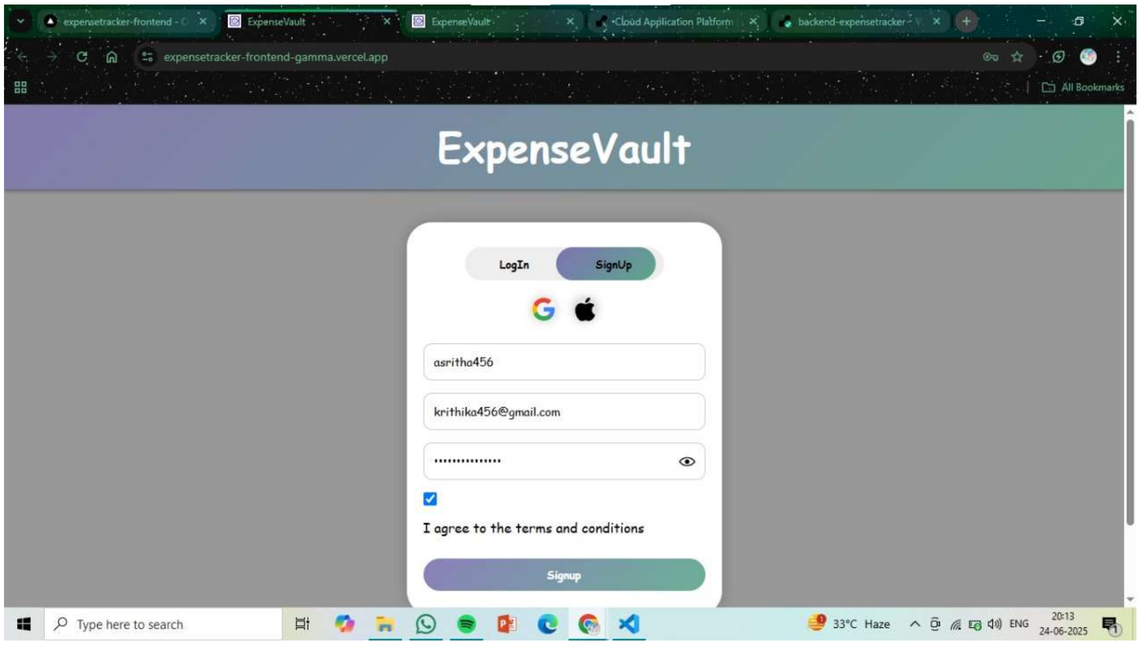
- Allows sorting of expenses by **amount** or **date**
- Ensures a clear and organized view of financial records

6. API Communication

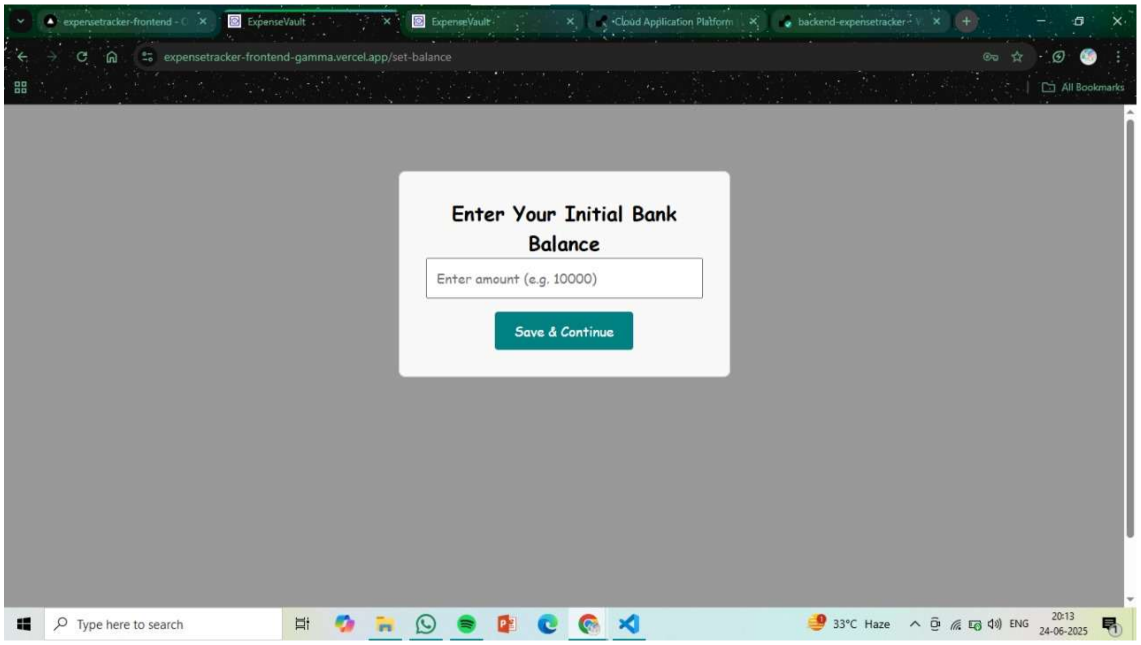
- Uses **Axios** to communicate with the backend server
- Handles all API requests such as:
- User login/register
- Adding expenses
- Fetching balance and transaction data

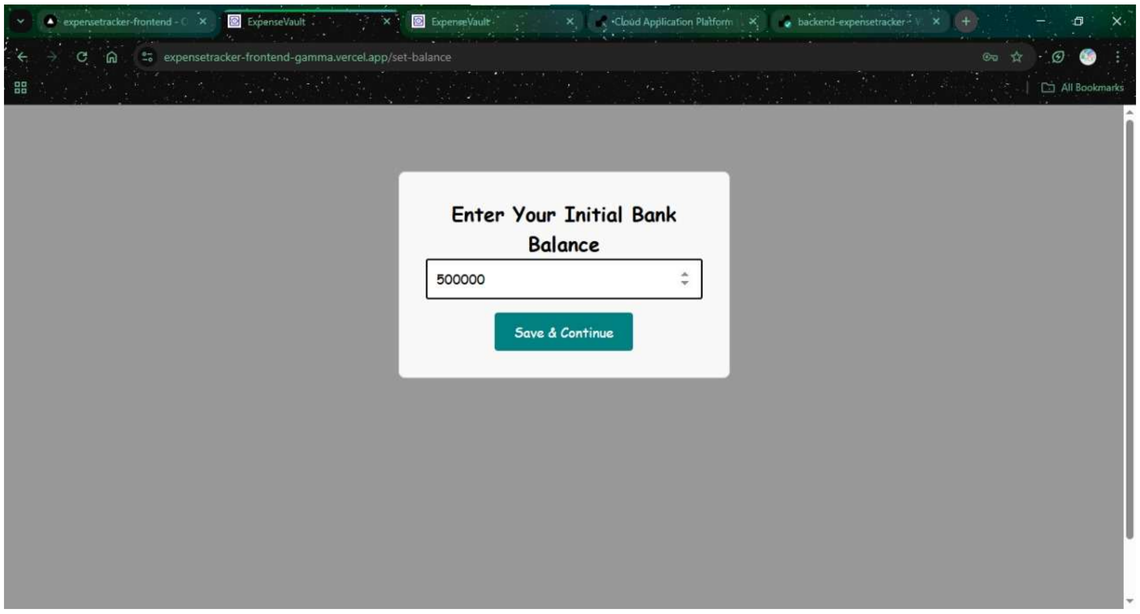
How the user interface looks like:

1. **First the user will signup with the username, email and password:**

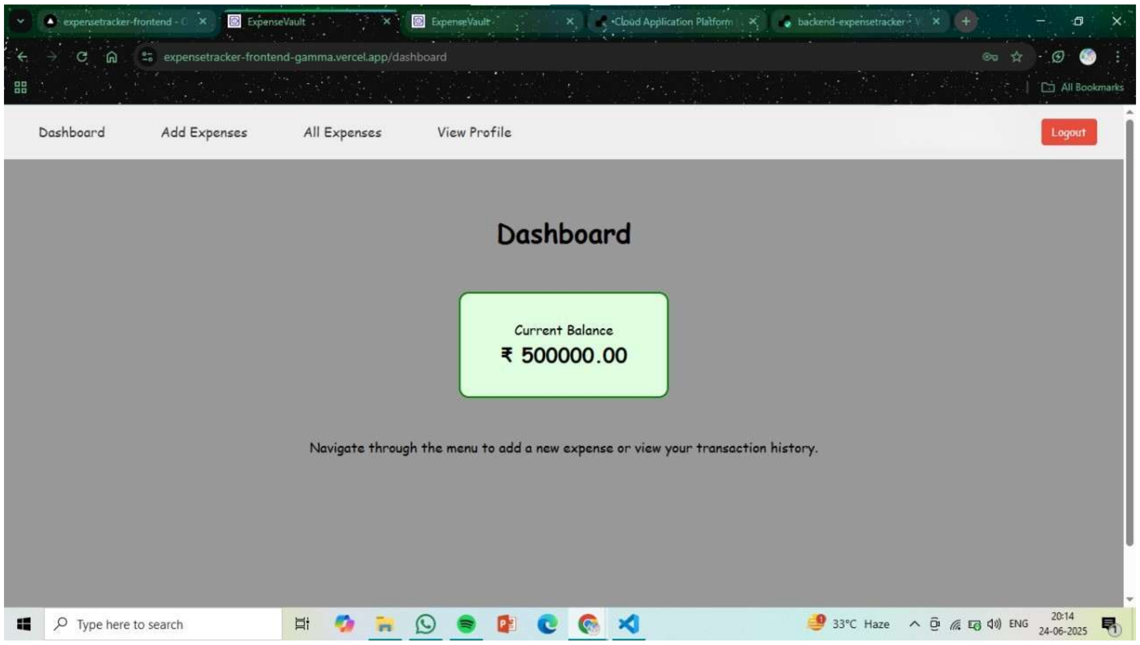


2. **Second it will redirect to set-initial bank balance page where user can set initial balance**

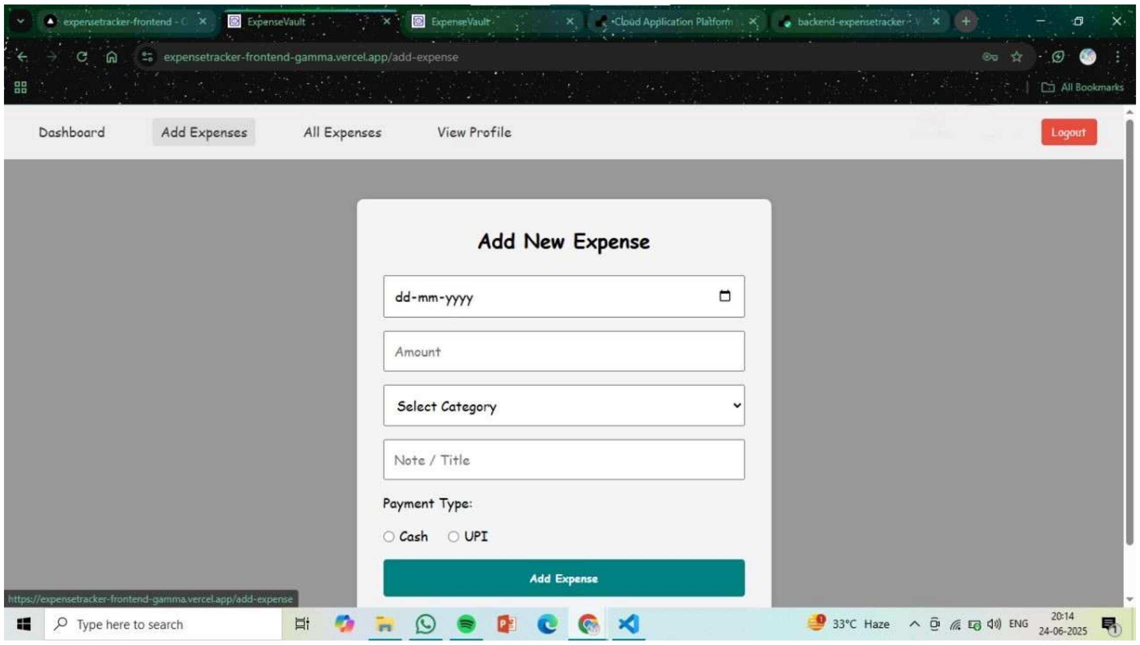



expensetracker-frontend-gamma.vercel.app/set-balance
Enter Your Initial Bank Balance
500000
Save & Continue

3. **Then the user will be redirected to dashboard where the user can find their initial balance**


expensetracker-frontend-gamma.vercel.app/dashboard
Dashboard
Add Expenses
All Expenses
View Profile
Logout
Dashboard
Current Balance
₹ 500000.00
Navigate through the menu to add a new expense or view your transaction history.

4. **User can add expense – by adding – date , amount, categories, note and their payment type**



Add New Expense

dd-mm-yyyy

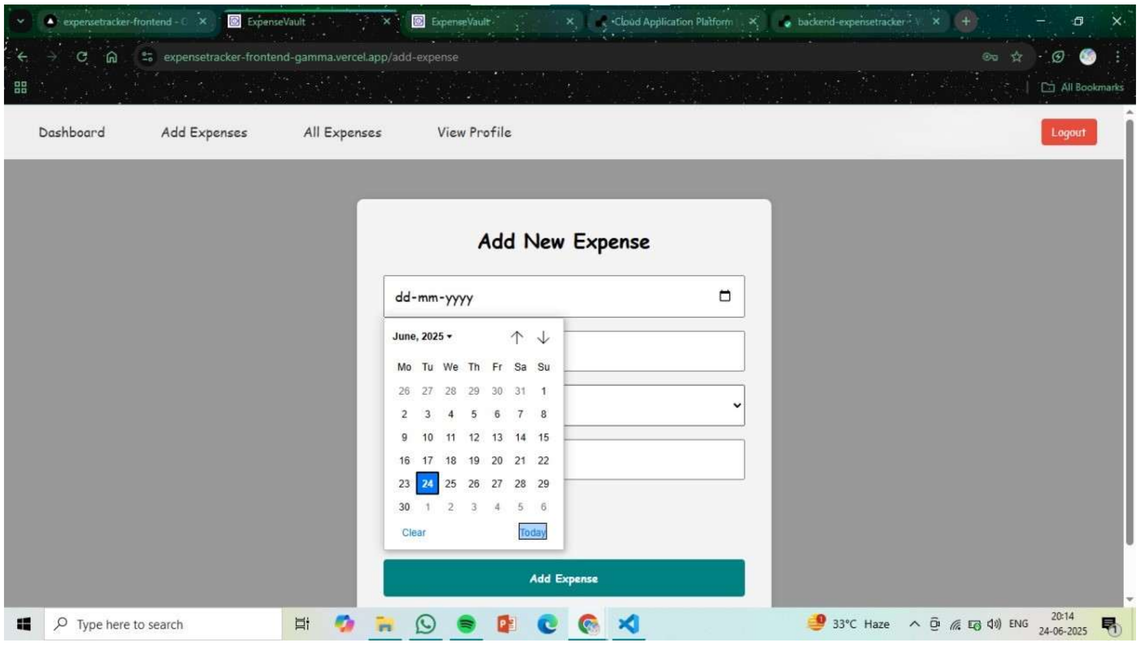
Amount

Select Category

Note / Title

Payment Type:

Cash UPI

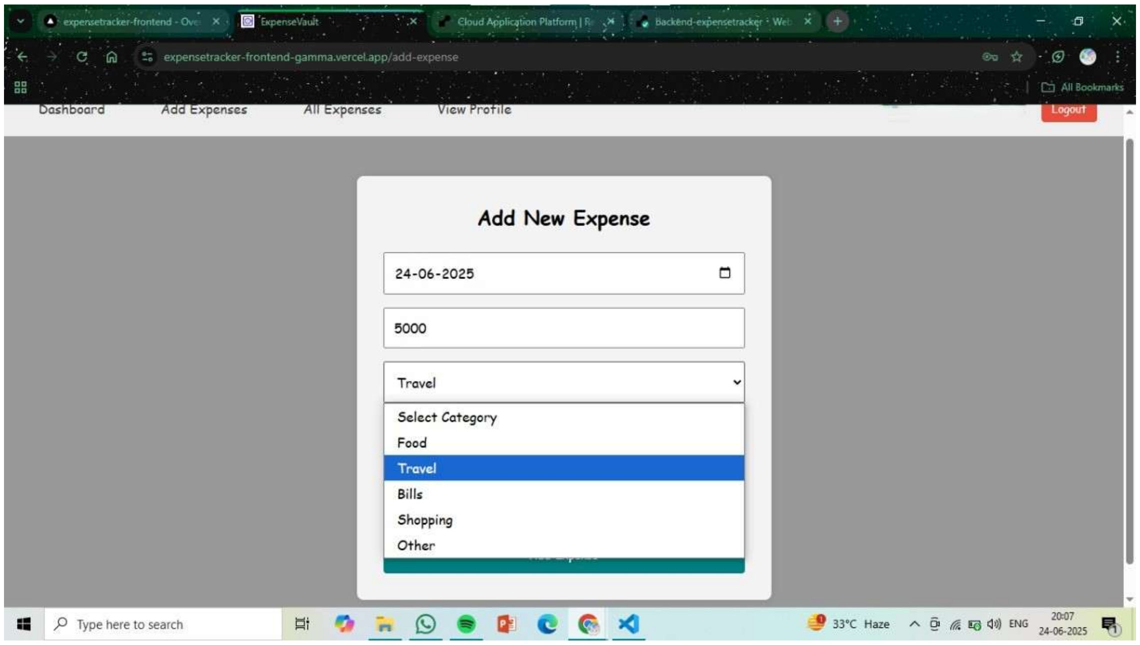


Add New Expense

dd-mm-yyyy

June, 2025

Mo	Tu	We	Th	Fr	Sa	Su
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6



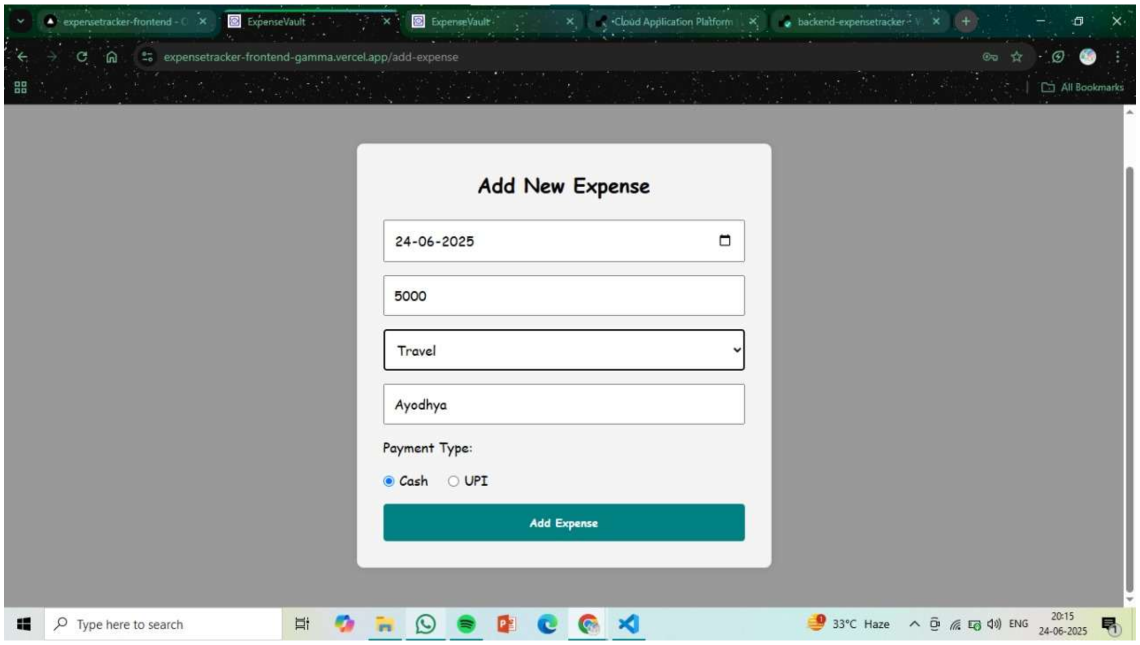
Add New Expense

24-06-2025

5000

Travel

And at last can add the expenses



Add New Expense

24-06-2025

5000

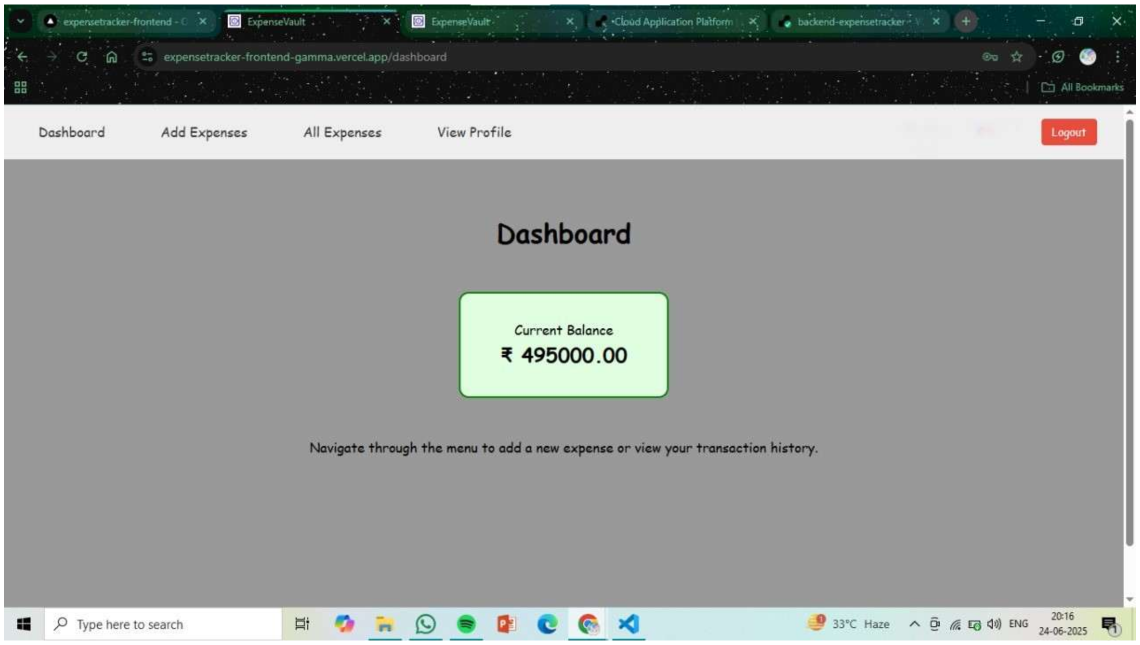
Travel

Ayodhya

Payment Type:

[✓ Cash] UPI

5. **Then user will be redirected to dashboard where his balance get automatically deducted by the backend**

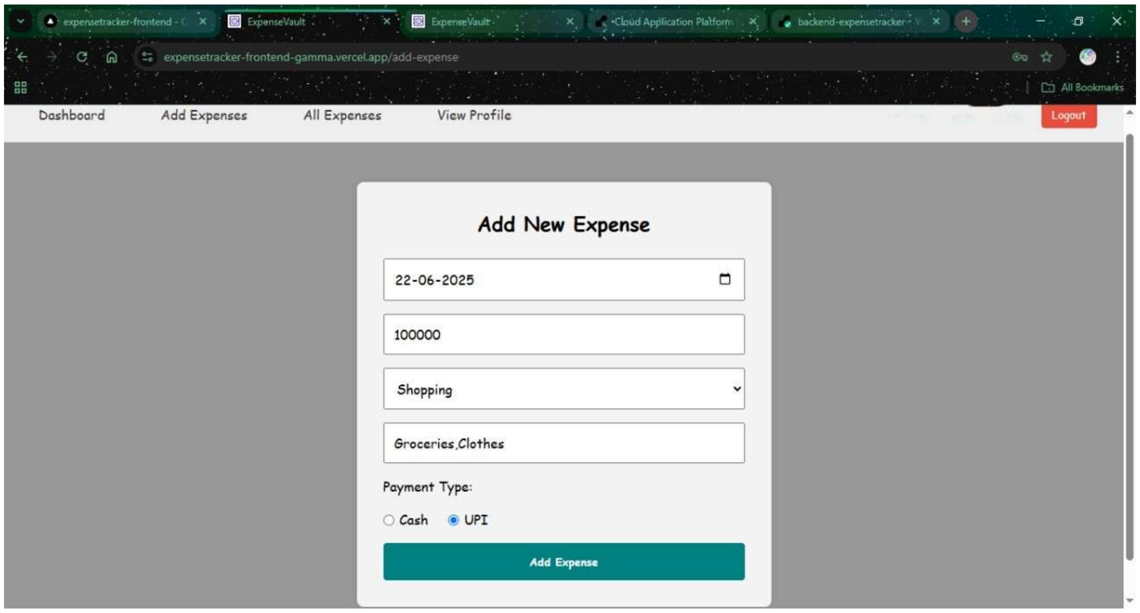


Dashboard

Current Balance
₹ 495000.00

Navigate through the menu to add a new expense or view your transaction history.

6. **Users can add as many expenses they need**



Add New Expense

22-06-2025

100000

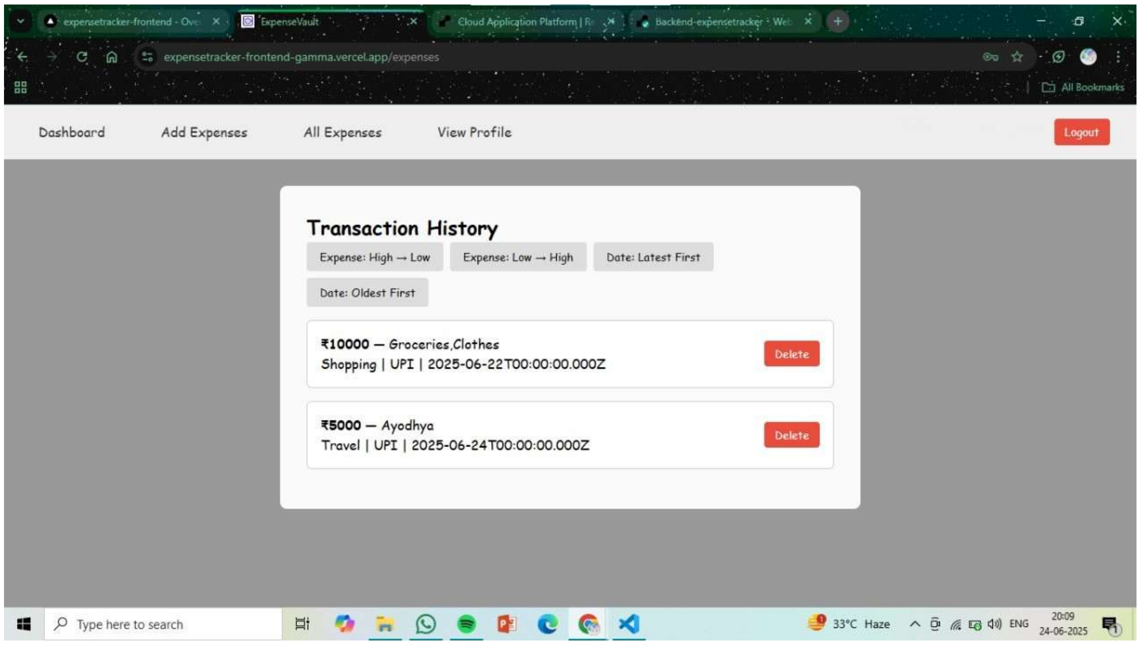
Shopping

Groceries,Clothes

Payment Type:

Cash [✓ UPI]

7. **Users can also view their Transaction**



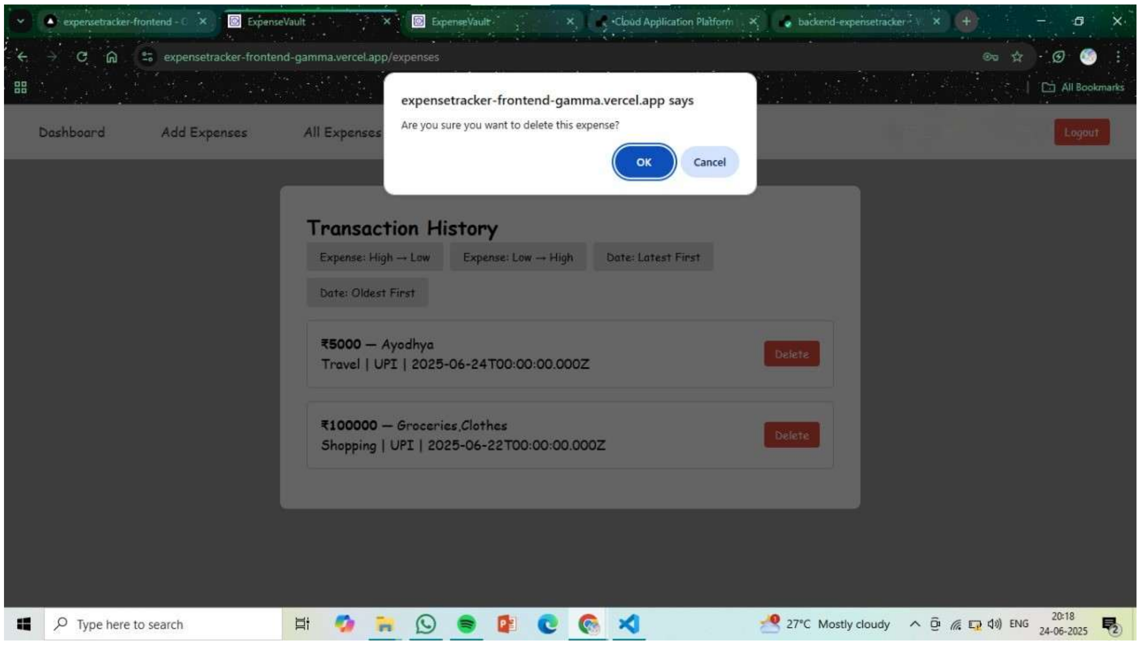
Transaction History

Expense: High → Low | Expense: Low → High | Date: Latest First | Date: Oldest First

₹10000 — Groceries,Clothes
Shopping | UPI | 2025-06-22T00:00:00.000Z

₹5000 — Ayodhya
Travel | UPI | 2025-06-24T00:00:00.000Z

8. **And can delete the expenses**



expensetracker-frontend-gamma.vercel.app says

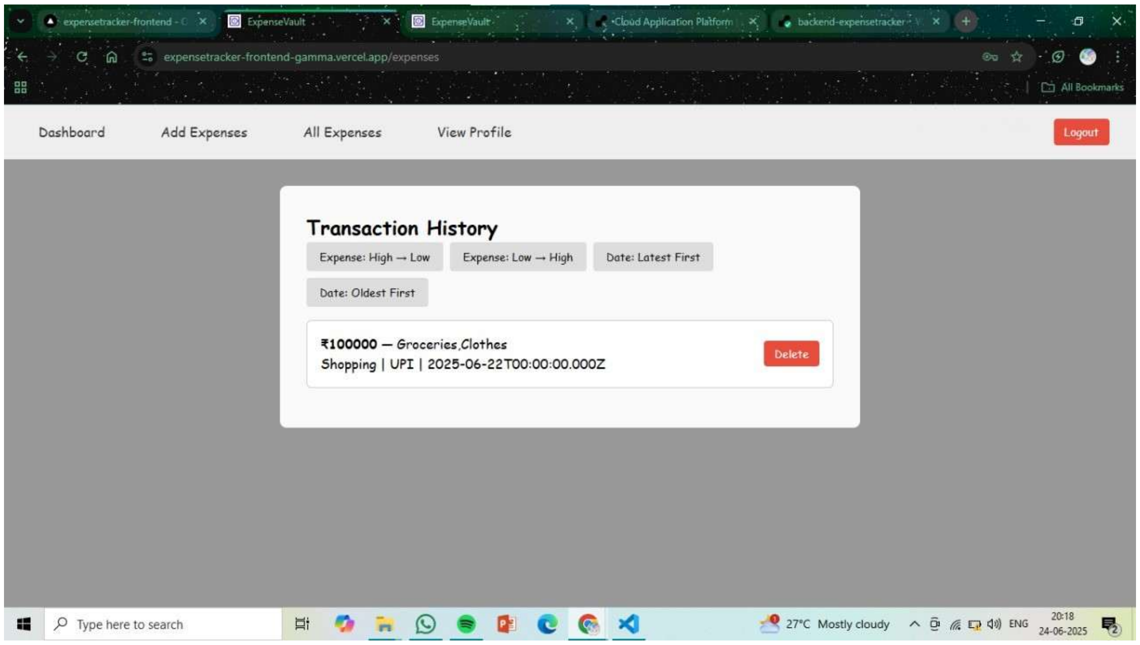
Are you sure you want to delete this expense?

Transaction History

Expense: High → Low | Expense: Low → High | Date: Latest First | Date: Oldest First

₹5000 — Ayodhya
Travel | UPI | 2025-06-24T00:00:00.000Z

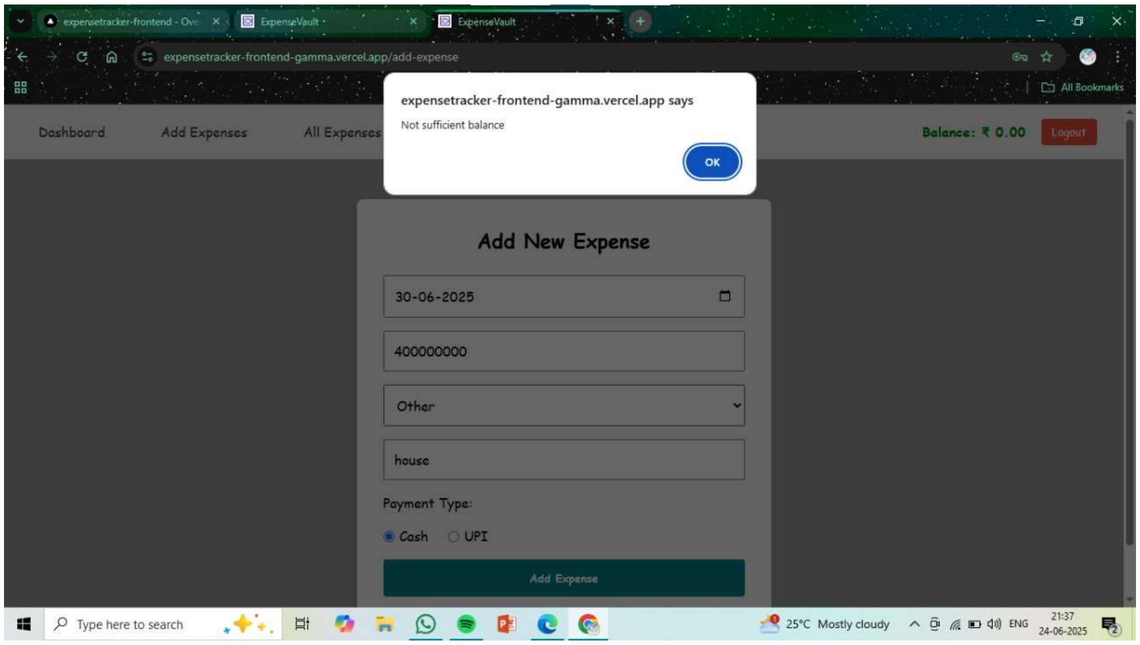
₹100000 — Groceries,Clothes
Shopping | UPI | 2025-06-22T00:00:00.000Z



Transaction History

₹100000 — Groceries,Clothes
Shopping | UPI | 2025-06-22T00:00:00.000Z

9. **If the user try to enter amount greater than balance will get the error.**



expensetracker-frontend-gamma.vercel.app says
Not sufficient balance

Add New Expense

30-06-2025
400000000
Other
house
Payment Type:
(● Cash) ○ UPI

Backend Functionality:

1. User Authentication

- Manages secure user registration and login
- Hashes passwords using bcrypt
- Generates and validates JWT tokens for authenticated sessions

2. Balance Management

- Stores the user's initial balance upon signup
- Updates the balance dynamically whenever an expense is added
- Exposes endpoints to get and update the user's current balance

3. Expense Handling

- Accepts expense data from the frontend and stores it in MongoDB
- Deducts the expense amount from the user's balance automatically
- Provides APIs to:
    - Add new expenses
    - Fetch all expenses for a specific user
    - Delete specific expenses (if implemented)

4.API's: ROUTES:

In this project, routes are used in the backend to define how different HTTP requests (like GET, POST, PUT, DELETE) should be handled. Think of them as the paths that connect the frontend with the backend logic and database.

For example:

Defined API's:

app.use('/auth',authRouter);

app.use('/expenses',authenticate, expenseRouter);

app.use('/users',authenticate,userRouter);

authRouter:

```
const authRouter = require('express').Router();
const authController = require('../controllers/authController');
const {authLimiter} = require('../middlewares/rateLimiterMiddleware');

authRouter.post('/signup',authLimiter,authController.signup);
authRouter.post('/login',authLimiter,authController.login);

module.exports = authRouter;
```

expenseRouter:

```
const expenseRouter = require('express').Router();
const expenseController = require('../controllers/expensesController')

expenseRouter.get('/',expenseController.getAllExpenses);

expenseRouter.get('/:id',expenseController.getExpenseById);

expenseRouter.post('/',expenseController.createExpense);

expenseRouter.patch('/:id',expenseController.updateExpense);

expenseRouter.delete('/:id',expenseController.deleteExpense);

module.exports = expenseRouter;
```

userRouter:

```
const userRouter = require('express').Router();
const userController = require('../controllers/userController');
const {authenticate} = require('../middlewares/authMiddleware');
const {requiredRole} = require('../middlewares/verifyRoleMiddleware');

userRouter.get('/',authenticate,requiredRole(['admin']),userController.getAllUsers);
userRouter.patch('/balance',userController.modifyInitialBalance);
userRouter.get('/balance',authenticate,userController.getBalance);

userRouter.put('/set-balance',authenticate,userController.updateBalance)

module.exports = userRouter;
```

- /auth/login → POST Method → Logs in a user
- /auth/signup → POST Method → Registers a new user
- /expenses → POST Method → Adds a new expense (and deducts from user balance)
- /expenses/:id → PATCH → Updates an existing expense

- /expenses/:id → DELETE → Deletes an existing expense
- /users/balance→ PATCH → Modifies the logged-in user's initial balance
- /users/balance→ GET → Fetches the current balance of the logged-in user
- /user/set-balance → PUT → Set initial balance

Each route calls a controller function that performs the actual logic (like querying the database, calculating balance, etc.).

How Balance is Calculated – Backend Flow

When a user signs up, the backend stores their initial balance in the database via the /auth/signup route.

<u>Whenever a new expense is added through POST /expenses, the backend:</u>

- Retrieves the logged-in user's balance.
- Checks if the balance is sufficient.
- If yes, deducts the expense amount.
- Saves the updated balance back to the user document.
- The updated balance can be:
- Viewed via GET /users/balance
- Manually changed using PATCH /users/balance

Note: The balance is not auto-adjusted if an expense is deleted or updated.

5. Database Integration

- Uses Mongoose to define and interact with:
    - User schema (credentials and balance)
    - Expense schema (title, amount, date, category)
- Ensures proper validation and data consistency

6. Security & Data Integrity

- Protects routes using JWT-based middleware to restrict access to authenticated users only
- Validates incoming data to prevent malformed or unauthorized entries

Expense Tracking Mechanism

1. User Adds an Expense (Frontend)

User fills out a form with:

- Title/Note
- Amount
- Category (e.g., Food, Transport)
- Date

This data is sent to the backend via:

POST /api/expenses/add

2. Backend Handles the Request (Routes + Controller)

- The /api/expenses/add route receives the request.
- It passes the data to a controller function (e.g., addExpense()).
- This function:
    - Saves the expense to the MongoDB database
    - Deducts the expense amount from the user's current balance (in the database)
    - Returns a success response with updated balance or confirmation

3. Frontend Updates the UI

- On successful response, the new expense is added to the transaction history.
- The balance is updated and shown to the user.

Internal Workflow

- Frontend: React components manage UI and handle user input.
- Axios: Used to make API calls to backend routes.
- Backend (Node + Express):
    - Routes define API paths (like /add, /balance, /expenses)
    - Controllers handle logic (like saving data, updating balance)
- Database: MongoDB stores:
    - User info
    - Expense records
    - Balance data