

## **PERSONAL EXPENSE TRACKER APPLICATION**

IBM-Project-50636-1660919764-(PNT2022TMID24728)

## **PERSONAL EXPENSE TRACKER APPLICATION**

**NALAIYA THIRAN PROJECT BASED LEARNING ON  
PROFESSIONAL READINESS FOR INNOVATION, EMPLOYMENT  
AND ENTREPRENEURSHIP.**

### **PROJECT REPORT**

- Varun Raj V R (210419104177)
- Sakthi Kumar S (210419104166)
- Vishnuram V(210419104185)
- Sudharsan S (210419104167)

**BACHELOR OF ENGINEERING IN  
COMPUTER SCIENCE AND ENGINEERING**

**Chennai Institute Of Technology**

**Chennai-600 069**

## INDEX

### **1. INTRODUCTION**

1. Project Overview
2. Purpose

### **2. LITERATURE SURVEY**

1. Existing problem
2. References
3. Problem Statement Definition

### **3. IDEATION & PROPOSED SOLUTION**

1. Empathy Map Canvas
2. Ideation & Brainstorming
3. Proposed Solution
4. Problem Solution fit

### **4. REQUIREMENT ANALYSIS**

1. Functional requirement
2. Non-Functional requirements

### **5. PROJECT DESIGN**

1. Data Flow Diagrams
2. Solution & Technical Architecture
3. User Stories

### **6. PROJECT PLANNING & SCHEDULING**

1. Sprint Planning & Estimation
2. Sprint Delivery Schedule
3. Reports from JIRA

### **7. CODING & SOLUTIONING (Explain the features added in the project along with code)**

1. Feature 1
2. Feature 2

3. Database Schema (if Applicable)

## 8. TESTING

1. Test Cases
2. User Acceptance Testing

## 9. RESULTS

1. Performance Metrics

## 10. ADVANTAGES & DISADVANTAGES

## 11. CONCLUSION

## 12. FUTURE SCOPE

## 13. APPENDIX

Source Code

GitHub & Project Demo Link

## **1. Introduction:**

### **1.1. Project overview:**

**Category: Cloud App Development**

**Team ID : PNT2022TMID24728**

**Skills Required: IBM Cloud,HTML,Javascript,IBM Cloud Object Storage,Python,Flask,Kubernetes,Docker,IBM DB2,IBM Container Registry**

In simple words, personal finance entails all the financial decisions and activities that a Finance app makes your life easier by helping you to manage your finances efficiently. A personal finance app will not only help you with budgeting and accounting but also give you helpful insights about money management. Personal finance applications will ask users to add their expenses and based on their expenses wallet balance will be updated which will be visible to the user. Also, users can get an analysis of their expenditure in graphical forms. They have an option to set a limit for the amount to be used for that particular month if the limit is exceeded the user will be notified with an email alert.

### **1.2. Purpose:**

It's easy to make this part of your everyday routine thanks to expense tracker apps that help you manage your money on the go. These apps certainly overlap with budgeting apps, but while the latter provides a big-picture view of your finances, expense tracker apps put more of an emphasis on your spending. These apps usually categorize your expenses and help you get a good idea of your purchasing behavior.

The database connectivity is planned using the "SQL Connection" methodology. The standards of security and data protective mechanisms have been given a big choice for proper usage. The application takes care of different modules and their associated reports" which are produced as

per the applicable strategies and standards that are put forward by the administrative staff.

The entire project has been developed keeping in view of the distributed client server computing technology" in mind. The specification has been normalized up to 3NF to eliminate all the anomalies that may arise due to the database transactions that are executed by the general users and the organizational administration. The user interfaces are browser specific to give distributed accessibility for the overall system. The internal database has been selected as MS-SQL server 2000.

The basic constructs of table spaces" clusters and inde0es have been exploited to provide higher consistency and reliability for the data storage. The MS-SQL server 2000 was a choice as it provides the constructs of high-level reliability and security. The total front end was dominated using the A%(.)et technologies. At all proper levels high care was taken to check that the system manages the data consistency with proper business rules or validations.

The database connectivity was planned using the latest "SQL Connection" technology provided by Microsoft corporation. The authentication and authorization was cross checked at all the relevant stages. The user level accessibility has been restricted into two zones namely.

## **2. Literature Survey:**

### **2.1. Existing problem:**

Students write their expenses in a notebook and find it difficult in finding the major expenses category so it tends them to expend more. Students may miss out on writing their expenses in the notebook, which makes them frustrated. Family guardians take the monthly grocery list on a piece of paper so they lose that they get frustrated. Computer users track their expenses in an excel sheet by using it and get bored with the rows and columns. Mobile Users manage their expenses in a mobile app they spend more if it doesn't have daily limit remainders so they get disappointment easily.

### **2.2. References:**

- [1] Velmurugan; Richard Francis; *Expense manager application*. Dec 2020
- [2] Saumya Dubey; Rigved Rishabh Kumar; *Student expense tracking application*. Apr 2014
- [3] Joseph Jofish; Rebecca; *Tracking personal finances* Apr 2014
- [4] Shahed Anzarus Sabab; Sadman Saumik Islam, Md. Jewel Rana; Monir Hossian; *e-Expense: A Smart Approach to Track Everyday Expense*
- [5] Yash veer Singh; Dhirendra Yadav; Kunal Singh; *Group expense tracker* Jun 2020
- [6] Daily Expense Tracker Mobile Application; Nuura Najati Binti Mustafa

[7] Expenditure management system; Dr.V.Geetha, G. Nikhitha.

[8]. Daily Expense Tracker; Shivam Mehra, Prabhat Parashar

[9]. Expense tracker application; Velmurugan.R ,Mrs.P.Usha

[10]. Intelligent Online Budget Tracker; Girish Bekaroo and Sameer Sunhaloo

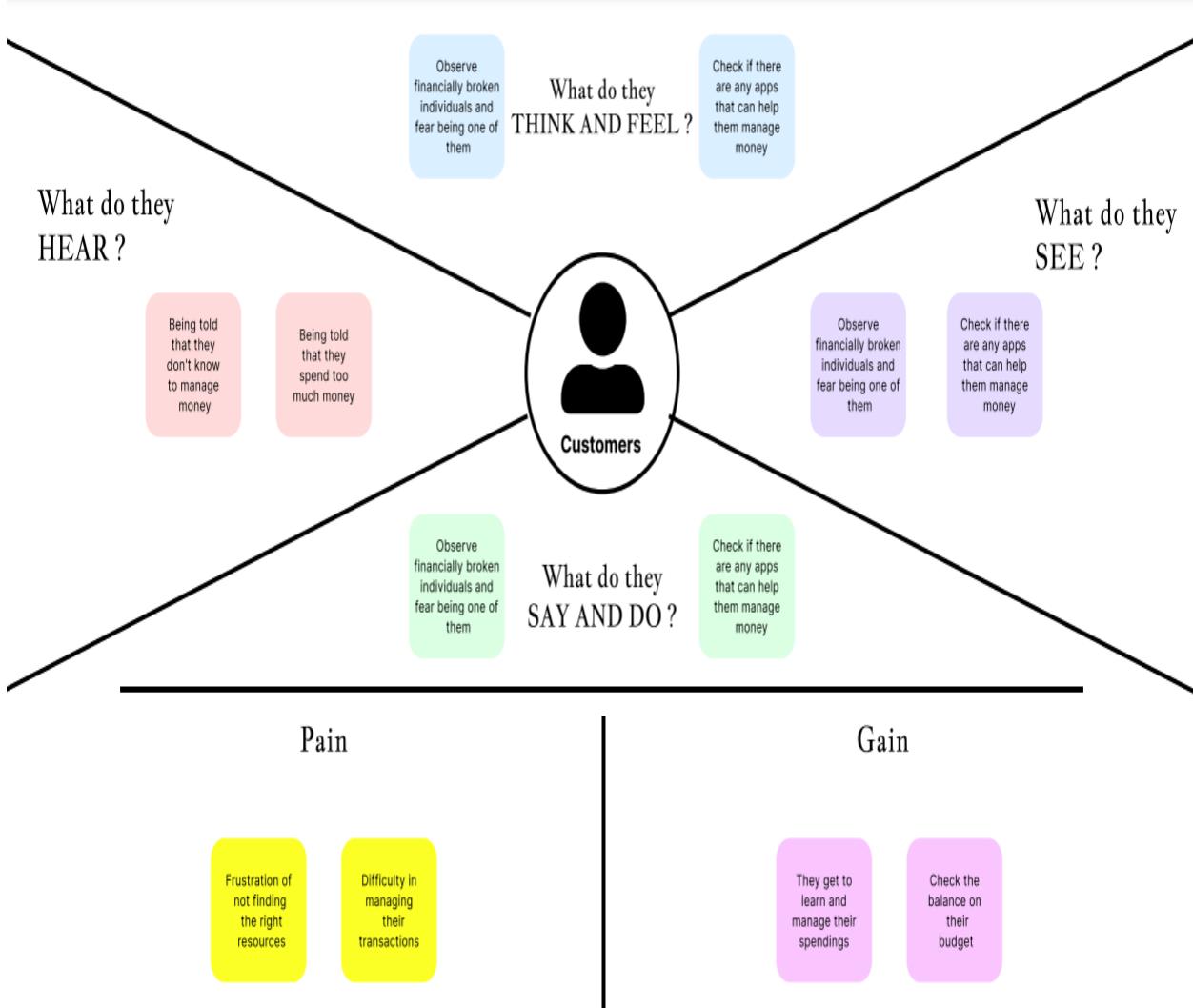
### **2.3. Problem statement definition:**

Many organizations have their own system to record their income and expenses, which they feel is the main key point of their business progress. It is good habit for a person to

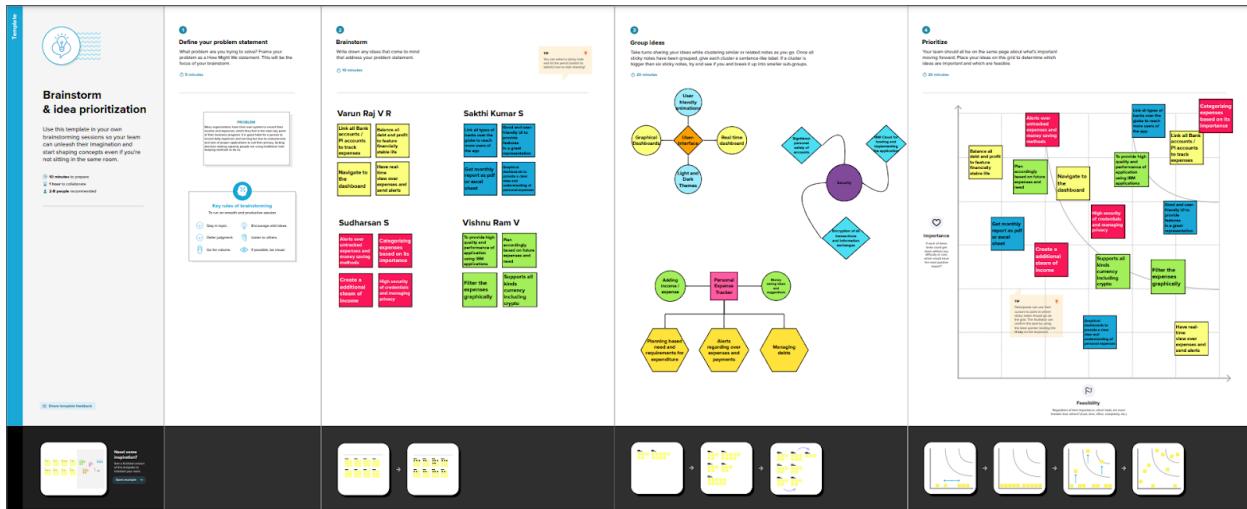
record daily expenses and earning but due to unawareness and lack of proper applications to suit their privacy, lacking decision making capacity people are using traditional note keeping methods to do so.

### 3. IDEATION & PROPOSED SOLUTION

#### 3.1. Empathy map canvas:



### 3.2. Ideation & Brainstorming:



### 3.3. Proposed Solution:

- Our application requests users to add their expenses and based on their expenses wallet balance will be updated which will be visible to the user. Users can avail an option to set a limit for the amount to be used for that particular cycle and when the user exceeds the limit, he receives an alert.
- The application also helps to keep track of bills the user has to pay, so the user gets regular reminders of due dates of bill payment. We also might send the user occasional notifications on how much limit he has left and his due dates.
- The user will be able to Stick to their Spending Limits. They can able to scan their bills any time thus data loss is avoided.
- Users can keep track of credit card bills and make payments on time so as to not get any unwanted interests.

### 3.4. Problem Solution Fit:

Define CS, fit into CC	<p><b>1. CUSTOMER SEGMENT(S)</b></p> <p>CS</p> <p>Our customers are people who have trouble tracking their expenses and would like to have an interactive application that shows them constant reminders about spending limits and due dates of bill payments.</p>	<p><b>6. CUSTOMER CONSTRAINTS</b></p> <p>CC</p> <p>Customers may find trouble regarding adding their income and expenses, as they have to enter every record manually. Sometimes the reminders and alerts about their budgets and goals might become overwhelming and annoying. Users also must have certain financial knowledge for using the application.</p>	<p><b>5. AVAILABLE SOLUTIONS</b></p> <p>AS</p> <p>While tracking expenses manually, customers have to track it and do calculations manually. So our application aims at removing that problems.</p> <p>Pros:</p> <ol style="list-style-type: none"> <li>1. Users can add their income and expenses and the app calculates the remaining balance, total amount spent by category and much more.</li> <li>2. Users can add their payment dues and also their last dates so they will get reminders and alerts.</li> </ol> <p>Cons:</p> <ol style="list-style-type: none"> <li>1. Manually enter every single transaction.</li> </ol>
Identify strong TR & EM	<p><b>2. JOBS-TO-BE-DONE / PROBLEMS</b></p> <p>J&amp;P</p> <p>In paper-based expense tracker system it is difficult to track our monthly expenses manually. The paper-based expense records may get lost very easily or even destroyed. Also people tend to forget their payment dates sometimes and get charged with late payment fees and interests.</p>	<p><b>9. PROBLEM ROOT CAUSE</b></p> <p>RC</p> <p>Written records are volatile and can be sometimes illegible as handwriting differs according to each person. So it can be hard to understand. Bill payment should be remained constantly as people forget about it during their daily life hustles.</p>	<p><b>7. BEHAVIOUR</b></p> <p>BE</p> <p>They may keep a temporary note on their mobile. People should know their budgets for each month and spend appropriately according to their goals.</p> <p>Manually track bills and payment dates maybe using sticky notes.</p>
Identify strong TR & EM	<p><b>3. TRIGGERS</b></p> <p>TR</p> <ul style="list-style-type: none"> <li>• Realizing that excessive spending leading to lack of money in case of emergencies</li> <li>• Lack of Budgeting knowledge.</li> </ul>	<p><b>10. YOUR SOLUTION</b></p> <p>SL</p> <ul style="list-style-type: none"> <li>• A cloud-based web application which keeps track of user's personal expenses. This system attempts to free the user with much of the burden of manual calculation and to keep track of the expenditure.</li> <li>• Users just need to enter their day-to-day expenses. They also have an option to set their limit. If the expenditure exceeds that limit, notification will be sent through mail.</li> </ul>	<p><b>8. CHANNELS of BEHAVIOR</b></p> <p>CH</p> <p><b>8.1 ONLINE</b> Less security and customer support. Real-time notification for un-tracked expenses is not available</p> <p><b>8.2 OFFLINE</b> Email alerts are disabled in offline mode as users are not connected to the internet.</p>
Identify strong TR & EM	<p><b>4. EMOTIONS: BEFORE / AFTER</b></p> <p>EM</p> <p>Before: Frustration, Confusion, Inadequate After: Boost , Feeling smart , Be an example for others</p>	<ul style="list-style-type: none"> <li>• Application enables users to add and track their bills and receive regular alerts on their due dates</li> </ul>	

#### **4. Requirement analysis:**

##### **4.1 Functional requirement:**

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form and Google OAuth
FR-2	User Confirmation	Email OTP verification
FR-3	User Financial Accounts	User data entry
FR-4	User Dashboard	Expense Data, set goals, add incomes and add bills
FR-5	User Notifications	System Access

		Real time Alerting
FR-6	Security of User Data	Secured Database  Data Security Algorithms

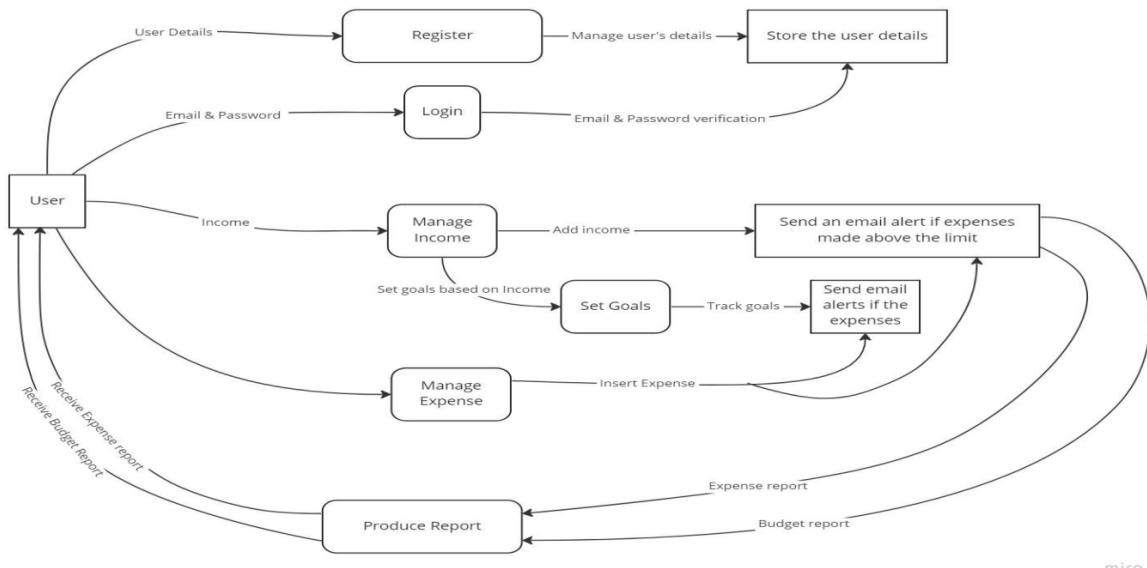
#### 4.2. Non- Functional requirements:

FR No.	Non-Functional Requirement	Description
NFR-1	<b>Usability</b>	By using this application, the user can keep track of  their expenses and can ensure that user's money is  used wisely.
NFR-2	<b>Security</b>	Maintain user personal details in a encrypted manner by using data security algorithms .

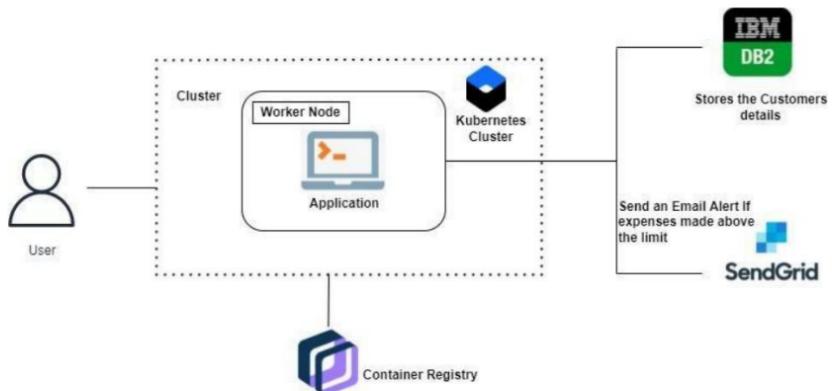
NFR-3	<b>Reliability</b>	It will maintain a proper tracking of day-to-day expenses in an efficient manner.
NFR-4	<b>Performance</b>	By enter our incoming and departing cash, and the software can help you keep and monitor it with at-most quality and security with high performance.
NFR-5	<b>Availability</b>	Using charts and graphs may help you monitor your budgeting and assets.
NFR-6	<b>Scalability</b>	Rely on your budgeting app to track, streamline, and automate all the recurrent expenses and remind you on a timely basis.

## 5. PROJECT DESIGN:

### 5.1. Data Flow Diagrams:



### 5.2. Solution & Technical Architecture:



### 5.3. User Stories:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer	Registration	USN-1	As a user, I can register for the application by	I can access my account /	High	Sprint-1

			<p>entering my email, password, and confirming my password.</p>	dashboard		
		USN-2	<p>As a user, I can log into the application by entering email &amp; password</p>	I can access The application	High	Sprint-1
	Dashboard	USN-3	<p>As a user I can enter my income And expenditure details under</p>	I can view my Daily expenses and track	High	Sprint-2

			<p>different categories.I can save my bills and set goals.</p>	my goals		
		USN-4	<p>Receive email alert if the goal is achieved or the expense exceed the budget.</p>	Mail received successfully	High	Sprint-3
		USN-5	<p>Monthly alerts can be set to pay the bills</p>	Alerts received successfully	Medium	Sprint-3

		USN-6	User can interact with watson if they have any queries.	Interaction implemented successfully	Medium	Sprint-4
Customer Care Executive		USN-7	As a customer care executive, I can  Solve the log in issues and other issues  of the application	I can provide support or solution at any time  24*7	Medium	Sprint-1
Administrator	Application	USN-8	As an administrator I can upgrade  or update the application.	I can fix the bug which arises for the	Medium	Sprint-1

				<b>Customers and users</b>		
				<b>application</b>		

## 6. PROJECT PLANNING & SCHEDULING:

### 6.1. Sprint Planning & Estimation:

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint 1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	Varun Raj V R
		USN-1	As a user, I will receive confirmation email once I have registered for the application	1	High	Sakthi Kumar S
	Login	USN-2	As a user, I can log into the application by entering email & password	1	High	Sudharsan S
	Dashboard	USN-3	Logging in takes to the dashboard for the logged user.	2	High	Vishnu Ram V

Bug fixes, routine checks and improvisation by everyone in the team \*Intended bugs only

Sprint 2	Workspace	USN-3	Workspace for personal expense tracking	2	High	Varun Raj V R
	Charts	USN-4	Creating various graphs and statistics of customer's data	1	Medium	Sudharsan S
	Connecting to IBM DB2	USN-3	Linking database with dashboard	2	High	Vishnu Ram V
		USN-3	Making dashboard interactive with JS	2	High	Sakthi Kumar S
Sprint-3		USN-3	Wrapping up the server side works of frontend	1	Medium	Sudharsan S
	SendGrid	USN-4	Using SendGrid to send mail to the user about their expenses	1	Low	Varun Raj V R
		USN-4	Integrating both frontend and backend	2	High	Vishnu Ram V

Bug fixes, routine checks and improvisation by everyone in the team \*Intended bugs only

	Watson Assistant	USN- 6	Creating Chatbot for expense tracking and for clarifying user's query	1	Medium	Sudharsan S
Sprint-4	Docker	USN-1	Creating image of website using docker/	2	High	Sakthi Kumar S
	Cloud Registry	USN-2	Uploading docker image to IBM Cloud registry	2	High	Sakthi Kumar S
	Kubernetes	USN-3	Create container using the docker image and hosting the site	2	High	Varun Raj V R
	Exposing	USN-4	Exposing IP/Ports for the site	2	High	Vishnu Ram V

### 6.2. Sprint Delivery Schedule:

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

### 6.3. Reports from JIRA:

JIRA backlog tab that shows the Sprint wise schedule and progress of tasks

Jira Software Your work Projects Filters Dashboards People Apps Create

Personal expense track... Software project

PLANNING Roadmap Backlog Board

DEVELOPMENT Code Project pages Add shortcut Project settings

You're in a team-managed project Learn more

Does your team need more from Jira? Get a free trial of our Standard plan.

Projects / Personal expense tracker Backlog

Search VR Epic Insights

PET Sprint 1 19 Oct – 29 Oct (10 issues) Complete sprint

PET Sprint 2 30 Oct – 5 Nov (10 issues) Start sprint

PET Sprint 3 6 Nov – 12 Nov (10 issues) Start sprint

PET Sprint 4 14 Nov – 19 Nov (9 issues) Start sprint

Backlog (0 issues) Create sprint

Quickstart

This screenshot shows the Jira Backlog page for a project named "Personal expense tracker". On the left, there's a sidebar with navigation links for Planning (Roadmap, Backlog, Board), Development (Code, Project pages, Add shortcut, Project settings), and a note about being in a team-managed project. The main area has a search bar, a VR button, and an "Epic" dropdown. It lists four completed sprints (PET Sprint 1 to 4) with their dates and issue counts, followed by a "Backlog" section with zero issues. Buttons for "Create sprint" and "Quickstart" are at the bottom right.

## JIRA Roadmap that shows the completion of tasks with time.

Personal expense track... Software project

PLANNING Roadmap Backlog Board

DEVELOPMENT Code Project pages Add shortcut Project settings

You're in a team-managed project Learn more

Projects / Personal expense tracker Roadmap

Give feedback Share Export View settings

Sprints

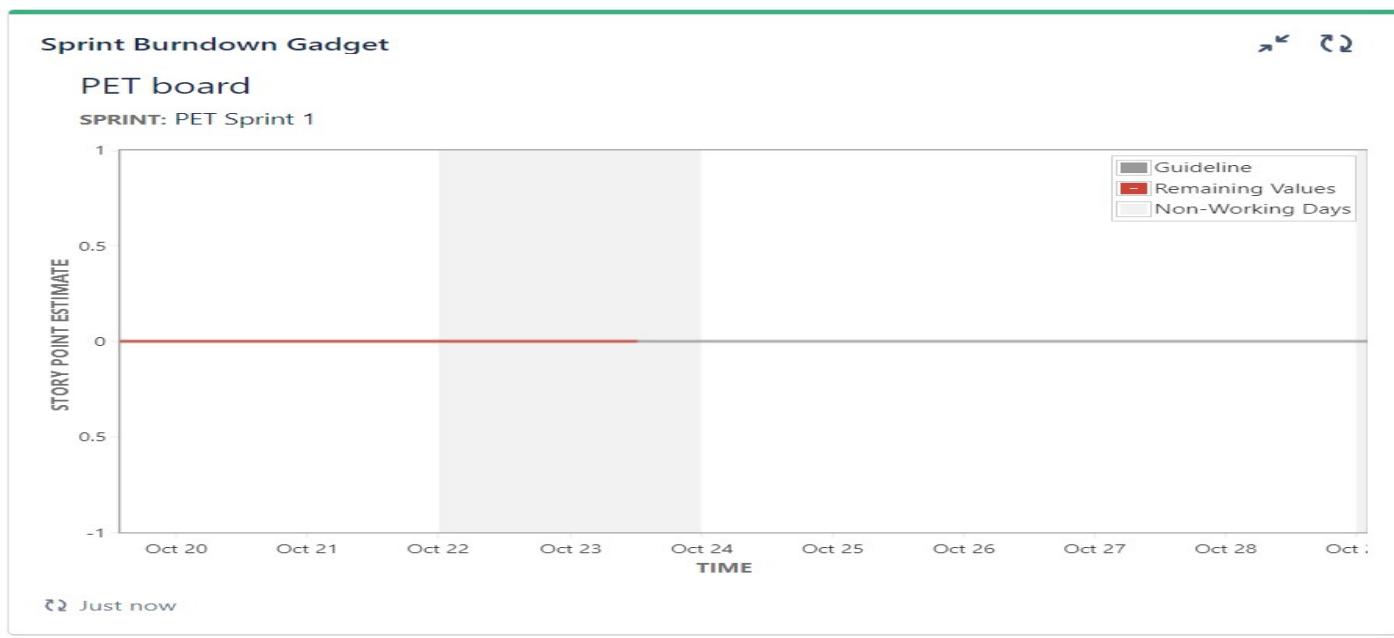
PET-10 Frontend PET Sprint 1 PET Sprint 2 PET Sprint 3

PET-11 Backend PET-12 Deployment PET-13 Frontend Testing PET-14 Backend testing

Create Epic

Today Weeks Months Quarters Quickstart

This screenshot shows the Jira Roadmap page for the same project. The sidebar is identical to the Backlog page. The main area displays a timeline from October 21 to November 15. It shows three sprints: "PET Sprint 1" (Oct 21-29), "PET Sprint 2" (Oct 30-Nov 5), and "PET Sprint 3" (Nov 6-12). Under each sprint, there are horizontal bars representing tasks like "PET-10 Frontend", "PET-11 Backend", etc. A vertical orange line marks the current date. At the bottom, there are buttons for "Today", "Weeks" (which is selected), "Months", "Quarters", and "Quickstart".



## 7. CODING & SOLUTIONING (Explain the features added in the project along with code)

### 7.1. Feature 1:

#### Adding transactions:

Users can add their transactions (expenses and income) to the records page and it will be displayed in the table. This can help them track how much they have earned and spent over time.

EXPLORER    ...    index.js

SOURCE CODE    ⌂ ⌂ ⌂

Sprint 2 > src > Pages > RecordDashboard > index.js

```
1 import React, { useEffect, useState } from 'react'
2 import Header from '../Components/Header'
3 import SideBar from '../Components/SideBar'
4 import { Box, Button, Checkbox, Container, Dialog, DialogActions, DialogContent, DialogContentText, DialogTitle, Divider, FormControlLabel, Paper, Table, TableBody, TableContainer, TableHead, TablePagination, TableRow, TextField } from '@material-ui/core'
5 import emptyImg from '../assets/empty_item.svg'
6 import './recordDashboard.css'
7 import BudgetCard from '../Components/BudgetCard'
8 import { recordStore } from '../store'
9 function RecordDialog(props) {
10     const { onClose, open } = props;
11
12     const handleClose = () => {
13         onClose();
14    };
15    const handleChange = (event) => {
16        props.setExpense({ ...props.expense, "gain": event.target.checked })
17    };
18
19    return (
20        <Dialog onClose={handleClose} open={open}>
21            <DialogTitle>Add expense</DialogTitle>
22            <DialogContent>
23                <TextField
24                    autoFocus
25                    margin="dense"
26                    required
27                    label="Amount"
28                    type="number"
29                    onChange={(e) => props.setExpense({ ...props.expense, "amount": e.target.value })}
30                    fullWidth
31                    variant="outlined"
32                />
33                <TextField
34                    autoFocus
35                    margin="dense"
36                    required
37                    label="Category"
38                    type="text"
39                    onChange={(e) => props.setExpense({ ...props.expense, "category": e.target.value })}
40                    fullWidth
41                />
42            </DialogContent>
43            <DialogActions>
44                <Button onClick={handleClose}>Cancel</Button>
45                <Button onClick={() => props.onAddExpense()}>Add</Button>
46            </DialogActions>
47        </Dialog>
48    );
49}
```

OUTLINE

TIMELINE

The screenshot shows a code editor interface with the following details:

- EXPLORER** tab is selected.
- SOURCE CODE** tab is active.
- index.js** is the currently open file.
- The file path is: Sprint 2 > src > Pages > RecordDashboard > index.js
- The code in **index.js** is as follows:

```
      1 //> <FormControlLabel control=<Checkbox defaultChecked value={props.expense?.gain} onChange={handleChange} /> label="is Expense" />
      2 <FormControlLabel control=<Input label="Category" type="text" onChange={(e) => props.setExpense({ ...props.expense, "category": e.target.value })} fullWidth variant="outlined" />
      3 <FormContent>
      4 <FormContent>
      5 <FormContent>
      6 <FormContent>
      7 <FormContent>
      8 <FormContent>
      9 <FormContent>
     10 <FormContent>
     11 <FormContent>
     12 <FormContent>
     13 <FormContent>
     14 <FormContent>
     15 <FormContent>
     16 <FormContent>
     17 <FormContent>
     18 <FormContent>
     19 <FormContent>
     20 <FormContent>
     21 <FormContent>
     22 <FormContent>
     23 <FormContent>
     24 <FormContent>
     25 <FormContent>
     26 <FormContent>
     27 <FormContent>
     28 <FormContent>
     29 <FormContent>
     30 <FormContent>
     31 <FormContent>
     32 <FormContent>
     33 <FormContent>
     34 <FormContent>
     35 <FormContent>
     36 <FormContent>
     37 <FormContent>
     38 <FormContent>
     39 <FormContent>
     40 <FormContent>
     41 <FormContent>
     42 <FormContent>
     43 <FormContent>
     44 <FormContent>
     45 <FormContent>
     46 <FormContent>
     47 <FormContent>
     48 <FormContent>
     49 <FormContent>
     50 <FormContent>
     51 <FormContent>
     52 const RecordDashboard = () => {
     53   const [hasExpense, setHasExpense] = useState(false);
     54   const [expense, setExpense] = useState(null);
     55   const records = recordStore.useState(s => s.records);
     56   const [open, setOpen] = useState(false);
     57   const handleClickOpen = () => {
     58     setOpen(true);
     59   };
     60   const getRecords = async () => {
     61     let token = localStorage.getItem('token');
     62     await fetch('http://localhost:5000/records', {
     63       method: 'GET',
     64       headers: new Headers({
     65         'x-access-token': token
     66       })
     67     }).then(res => res.json()).then(data => {
     68       console.log(data)
     69       recordStore.update(s => {
     70         s.records = data.records
     71       })
     72     })
     73   }
     74   useEffect(() => {
     75     getRecords()
     76   }, [records])
}
```

Below the code editor, there are navigation buttons: **OUTLINE** and **TIMELINE**.

The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar on the left showing project structure:
  - Sprint 1
  - Sprint 2
  - backend
  - node\_modules
  - public
  - src
    - assets
    - Components
    - Pages
      - BillDashboard
      - BudgetDashboard
      - Login
    - RecordDashboard
      - index.js (selected)
      - recordDashboard...
    - Signup
    - App.css
    - App.js
    - index.js
    - reportWebVitals.js
    - store.js
  - .gitignore
  - package-lock.json
  - package.json
  - README.md
  - Sprint 3
  - Sprint 4
- index.js** file open in the main editor area, showing code for a React component:

```
useEffect(() => {
    getRecords()
}, [records])
const formData = new FormData();
const handleClose = (value) => {
    if (expense) {
        setHasExpense(true)
        formData.append("amount", expense.amount)
        formData.append("category", expense.category)
        formData.append("gain", expense.gain)
        let token = localStorage.getItem('token')
        fetch('http://localhost:5000/records', {
            method: 'POST',
            body: formData,
            headers: new Headers({
                "x-access-token": token
            })
        }).then(res => res.json()).then(data => {
            console.log(data)
            getRecords()
        })
    }
    setOpen(false);
};
return (
    <>
        <SideBar />
        <div className='dashBoardContainer'>
            <Header />
            <div className="record_body">
                {records.length > 0 ? (
                    <div className="record_body_container">
                        /* <Paper elevation={5} className="record_chartCont">
                            <Typography variant="h6" color="text.secondary" gutterBottom>
                                Chart
                            </Typography>
                        </Paper> */
                        <Paper elevation={5} style={{ padding: "20px" }} className="record_TableCont">
                            <Box sx={{ display: "flex", justifyContent: "space-between", alignItems: "center" }} >
```

EXPLORER

SOURCE CODE

index.js

Sprint 2 > src > Pages > RecordDashboard > index.js

```
100     <SkeuBar />
101     <div className='dashBoardContainer'>
102         <Header />
103         <div className="record__body">
104             {records.length > 0 ? (
105                 <div className="record__body__container">
106                     /* <Paper elevation={5} className="record__ChartCont">
107                         <Typography variant="h6" color="text.secondary" gutterBottom>
108                             Chart
109                         </Typography>
110                     </Paper> *)
111                     <Paper elevation={5} style={{ padding: "20px" }} className="record__TableCont">
112                         <Box sx={{ display: "flex", justifyContent: "space-between", alignItems: "center" }}>
113                             <Typography variant="h6" color="text.secondary" gutterBottom>
114                                 Records
115                             </Typography>
116                             <Button onClick={handleClickOpen} variant="contained">Add expense</Button>
117                             <RecordDialog
118                                 open={open}
119                                 onClose={handleClose}
120                                 setExpense={setExpense}
121                                 expense={expense}
122                             />
123                         </Box>
124                         <Table sx={{ minWidth: 650 }} aria-label="simple table">
125                             <TableHead>
126                                 <TableRow>
127                                     <TableCell>Date</TableCell>
128                                     <TableCell align="right">Category</TableCell>
129                                     <TableCell align="right">Amount</TableCell>
130                                 </TableRow>
131                             </TableHead>
132                             <TableBody>
133                                 {records.map((row, ind) => (
134                                     <TableRow
135                                         key={ind}
136                                         sx={{ '&:last-child td, &:last-child th': { border: 0 } }}>
137                                         >
138                                         <TableCell>{new Date(row?.date_created).toLocaleDateString()}</TableCell>
139                                         <TableCell align="right">{row?.category}</TableCell>
```

OUTLINE

TIMELINE

```
index.js
  ...
  <Table>
    <TableBody>
      {records.map((row, index) => (
        <TableRow key={index} sx={{ '&:last-child td, &:last-child th': { border: 0 } }}>
          <TableCell>{new Date(row.date_created).toLocaleDateString()}</TableCell>
          <TableCell align="right">{row.category}</TableCell>
          <TableCell align="right" component="th" scope="row">
            {row.amount}
          </TableCell>
        </TableRow>
      ))}
    </TableBody>
  </Table>
</Paper>
</div>
) : (
  <div className="record_no_body_container">
    <h3>You have not created any transactions yet</h3>
    <img src={emptyImg} alt="" />
    <Button onClick={handleClickOpen} variant="contained">Add expense</Button>
    <RecordDialog
      open={open}
      onClose={handleClose}
      setExpense={setExpense}
      expense={expense}>
      </RecordDialog>
    </div>
  </div>
)
</div>
)
}

export default RecordDashboard
```

## 7.1. Feature 2:

### Tracking Budgets:

**Users can create their budget limits and our app sends you an email alert when the total expenses exceed the limit.**

```
index.js
38     </Box>
39   );
40 }
41 function BudgetDialog(props) {
42   const { onClose, open } = props;
43
44   const handleClose = () => {
45     onClose();
46   };
47
48   return (
49     <Dialog onClose={handleClose} open={open}>
50       <DialogTitle>Add budget</DialogTitle>
51       <DialogContent>
52         <TextField
53           margin="dense"
54           label="Monthly Budget"
55           onChange={(e) => props.setBudget(e.target.value)}
56           type="number"
57           fullWidth
58           required
59           variant="outlined"
60         />
61       </DialogContent>
62       <DialogActions>
63         <Button onClick={handleClose}>Done</Button>
64       </DialogActions>
65     </Dialog>
66   );
67 }
```

```
app.py
44 db.create_all()
45 class User(db.Model):
46     id = db.Column(db.Integer, primary_key = True)
47     public_id = db.Column(db.String(50), unique = True)
48     name = db.Column(db.String(100))
49     email = db.Column(db.String(70), unique = True)
50     password = db.Column(db.String(80))
51     monthly_limit = db.Column(db.Float)
52     phone_number = db.Column(db.Integer)
53     income = db.Column(db.Float)
```

### 7.3. Database schema (If applicable):

3 separate schemas for Bills, records and Users were created.

EXPLORER    ...    app.py 5 ×

SOURCE CODE Sprint 4 > backend > app.py > ...

```
>/
38     with app.app_context():
39         db.create_all()
40     class User(db.Model):
41         id = db.Column(db.Integer, primary_key = True)
42         public_id = db.Column(db.String(50), unique = True)
43         name = db.Column(db.String(100))
44         email = db.Column(db.String(70), unique = True)
45         password = db.Column(db.String(80))
46         monthly_limit = db.Column(db.Float)
47         phone_number = db.Column(db.Integer)
48         income = db.Column(db.Float)
49
50     class Record(db.Model):
51         id = db.Column(db.Integer,primary_key = True)
52         user = db.Column(db.String(50))
53         category = db.Column(db.String(50))
54         date_created = db.Column(db.DateTime(timezone=True),default=datetime.utcnow)
55         amount = db.Column(db.Float)
56         gain = db.Column(db.Boolean)
57     @property
58     def serialize(self):
59         return {
60             'id' : self.id,
61             'user':self.user,
62             'category':self.category,
63             'date_created':self.date_created,
64             'amount':self.amount,
65             'gain':self.gain
66             # This is an example how to deal with Many2Many relations
67         }
68
69     class Bills(db.Model):
70         id = db.Column(db.Integer, primary_key = True)
71         user = db.Column(db.String(50))
72         name = db.Column(db.String(50))
73         due_date = db.Column(db.Date)
74         amount = db.Column(db.Float)
75         date_created = db.Column(db.DateTime(timezone=True),default=datetime.utcnow)
76     @property
```

OUTLINE    ...

TIMELINE

EXPLORER    ...    app.py 5 ×

SOURCE CODE Sprint 4 > backend > app.py > ...

```
>/
58     def serialize(self):
59         return {
60             'id' : self.id,
61             'user':self.user,
62             'category':self.category,
63             'date_created':self.date_created,
64             'amount':self.amount,
65             'gain':self.gain
66             # This is an example how to deal with Many2Many relations
67         }
68
69     class Bills(db.Model):
70         id = db.Column(db.Integer, primary_key = True)
71         user = db.Column(db.String(50))
72         name = db.Column(db.String(50))
73         due_date = db.Column(db.Date)
74         amount = db.Column(db.Float)
75         date_created = db.Column(db.DateTime(timezone=True),default=datetime.utcnow)
76     @property
77     def serialize(self):
78         return {
79             'id' : self.id,
80             'user':self.user,
81             'name':self.name,
82             'date_created':self.date_created,
83             'due_date':self.due_date,
84             'amount':self.amount,
85             # This is an example how to deal with Many2Many relations
86         }
87     # decorator for verifying the JWT
```

## 8. Testing

### 8.1. Test cases:

			Date	17-Nov-22									
			Team ID	PNT2022TMID24728									
			Project Name	Personal Expense Tracker Application									
Test case ID	Feature Type	Component	Test Scenario	Pre-Requisite	Steps To Execute	Test Data	Expected Result	Actual Result	Status	Comments	TC for Automation(Y/N)	BUG ID	Executed By
1	Functional	Login Page	Verify user is able to Login into the Application		1) Open the Personal expense tracker application. 2) Login with user Credentials 3) Verify logged in to user account	Username: Varun Password: test	Login Successful	Working as expected	Pass		N		Varun
2	Functional	Signup Page	Verify user is able to Signup in the Application		1) Open the Personal expense tracker 2) Enter the Details and Create a new User 3) Verify if user is created and inserted into DB Table	Username: Sudharshan Password: test Name: Ayshu DOB: 12/9/2001 Password: test	Account Created Successfully	Working as expected	Pass		N		Sakthi
3	Functional	Dashboard page	Verify if all the user details are stored in Database		1) Open the Personal expense tracker application. 2) Enter the Details and Create a new User 3) Verify if user is created and inserted into DB Table	Username: Sakthi@gmail.com password: Testing123	User should navigate to user account homepage	Working as expected	Pass				Sudharsan
4	Functional	Login page	Verify user is able to log into application with Invalid credentials		1.Enter URL and click go 2.Click on Sign IN button 3.Enter InValid username/email in Email text box 4.Enter valid password in password text box 5.Click on login button	Username: chalam@gmail password: Testing123	Application should show 'Incorrect email or password' validation message.	Working as expected	Pass				Vishnu
5	Functional	Login page	Verify user is able to log into application with Invalid credentials		1.Enter URL and click go 2.Click on Sign IN button 3.Enter InValid username/email in Email text box 4.Enter valid password in password text box 5.Click on login button	Username: chalam@gmail.com password: Testing12367868786876876876	Application should show 'Incorrect email or password' validation message.	Working as expected	Pass				Varun

## 8.2. User Acceptance testing:

This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	1	0	0	0	1
Duplicate	1	0	0	0	1
External	3	1	0	0	4
Fixed	4	1	0	0	5
Not Reproduced	0	0	0	0	1
Skipped	0	0	0	0	0
Won't Fix	0	0	0	0	0
Totals	9	2	0	0	11

This report shows the number of test cases that have passed, failed, and untested

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	0	0	0	0
Client Application	5	0	0	5
Security	0	0	0	0
Outsource Shipping	0	0	0	0
Exception Reporting	5	0	0	5
Final Report Output	0	0	0	0
Version Control	0	0	0	0

## 9. Results

### 9.1. Performance metrics:

Scope/feature	NFT - Risk Assessment						Risk Score	Justification
	Functional Changes	Hardware Changes	Software Changes	Impact of Downtime	Load/Volume Changes			
New	Low	No Changes	Moderate	Yes, 2hrs	>10 to 30%		GREEN	

NFT - Detailed Test Plan					
S.No	Project Overview	NFT Test approach	Assumptions/Dependencies/Risks	Approvals/SignOff	
1	Login Page	1) Open the Personal Expense Tracker Application 2) Login with user Credentials	No Risks	N/A	
2	Signup Page	1) Open the Personal Expense Tracker Application 2) Enter the Details and Create a new User	No Risks	N/A	
3	Records Page	1) Log in to Personal Expense Tracker Application 2) Enter all the pesonal details and expenses and mark it as expense or income	No Risks	N/A	
4	Dashboard	1) Log in to Personal Expense Tracker Application 2) View the Analytics	No Risks	N/A	
5	Bills Page	1) Log in to Personal Expense Tracker Application 2) Bills can be added.	No Risks	N/A	
5	Email Acknowledgement	1) Mails are Sent to the Registered user if expenses>budget	No Risks	N/A	

End Of Test Report						
NFT Test approach	NFR - Met	Test Outcome	GO/NO-GO decision	Recommendations	Identified Defects (Detected/Closed/Open)	Approvals/SignOff
1) Log in to Personal Expense Tracker Application 2) Test for all Testcases 3) Log out to Personal Expense Tracker Application	YES	Test Passed	GO/NO-GO decision	N/A	None	N/A

## **10. Advantages and Disadvantages**

### **Advantages**

- This system helps users to reduce their expenses.
- Easy to use.
- You'll have better insight into your spending habits.
- Provides a better overview and comprehensive analysis.
- Limit your spending.

### **Disadvantages**

- Need to spend specified time to enter data.
- On some occasions it is not mandatory to restrict expenses.

## **11. Conclusion**

Personal Expense Tracker is a web application. We created this application so that a user can accurately calculate his/her daily cost. After making this application we assure that this application will help its users to manage the cost of their daily expenditure. It will guide them and make them aware about their daily expenses. It will prove to be helpful for the people who are frustrated with their daily budget management, irritated because of the amount of expenses and wish to manage money and to preserve the record of their daily cost which may be useful to change their way of spending money. In short, this application will help its users to overcome the wastage of money.

## **12. Future scope**

Now in our web application we covered almost all features but in the future we will add some more futures. The features are below:

- Multiple account support.
- Giving an alert message to the user's mobile number.
- Include currency converter.
- Reports are created in categories.
- Giving users the ability to export the data in all available formats like pdf, excel etc...

## 13. Appendix

### 13.1 Source code:

#### App.py (Flask file):

```
from flask import Flask, request, jsonify, make_response, current_app
from flask_sqlalchemy import SQLAlchemy
import uuid
from sqlalchemy import extract
from flask_cors import CORS
# import ibm_db
# import ibm_db_sa
from werkzeug.security import generate_password_hash, check_password_hash
# imports for PyJWT authentication
import jwt
from datetime import datetime, timedelta
from functools import wraps
from waitress import serve
# from ibm_db_alembic.ibm_db import IbmDbImpl
import os
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail

# creates Flask object
app = Flask(__name__)
CORS(app)

# configuration
# NEVER HARDCODE YOUR CONFIGURATION IN YOUR CODE
# within this block, current_app points to app.
# INSTEAD CREATE A .env FILE AND STORE IN IT
app.config['SECRET_KEY'] = ''
# database name
app.config['CORS_HEADERS'] = 'Content-Type'
# app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///Database.db'
app.config['SQLALCHEMY_DATABASE_URI'] = ''
```

```
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
# creates SQLALCHEMY object
db = SQLAlchemy(app)
# Database ORMs

default_info = {
    'name': '',
    'limit': 5000,
    'phone_number': 0,
    'currency': '₹',
    'income': 0,
    'category': 'misc'
}

# with app.app_context():

#     db.create_all()

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    public_id = db.Column(db.String(50), unique=True)
    name = db.Column(db.String(100))
    email = db.Column(db.String(70), unique=True)
    password = db.Column(db.String(110))
    monthly_limit = db.Column(db.Float)
    phone_number = db.Column(db.String(20))
    income = db.Column(db.Float)

class Record(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user = db.Column(db.String(50))
    category = db.Column(db.String(50))
    date_created = db.Column(db.DateTime(
        timezone=True), default=datetime.utcnow)
```

```

amount = db.Column(db.Float)
gain = db.Column(db.Boolean)

@property
def serialize(self):
    return {
        'id': self.id,
        'user': self.user,
        'category': self.category,
        'date_created': self.date_created,
        'amount': self.amount,
        'gain': self.gain
    # This is an example how to deal with Many2Many relations
}

```

```

class Bills(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user = db.Column(db.String(50))
    name = db.Column(db.String(50))
    due_date = db.Column(db.Date)
    amount = db.Column(db.Float)
    date_created = db.Column(db.DateTime(
        timezone=True), default=datetime.utcnow)

@property
def serialize(self):
    return {
        'id': self.id,
        'user': self.user,
        'name': self.name,
        'date_created': self.date_created,
        'due_date': self.due_date,
        'amount': self.amount,
    # This is an example how to deal with Many2Many relations
}

```

```

# decorator for verifying the JWT


def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = None

        # jwt is passed in the request header
        if 'x-access-token' in request.headers:
            token = request.headers['x-access-token']

        # return 401 if token is not passed
        if not token:
            return jsonify({'message': 'Token is missing !!!'}), 401

        try:
            # decoding the payload to fetch the stored details
            # print("received token : ",token)
            # print(app.config['SECRET_KEY'])
            data = jwt.decode(
                token, app.config['SECRET_KEY'], algorithms=["HS256"])
            # print("data",data)
            current_user = User.query\
                .filter_by(public_id=data['public_id'])\
                .first()

        except:
            return jsonify({
                'message': 'Token is invalid !!!'
            }), 401

        # returns the current logged in users context to the routes
        return f(current_user, *args, **kwargs)

    return decorated

# User Database Route
# this route sends back list of users

```

```
@app.route('/user', methods=['GET'])
@token_required
def get_all_users(current_user):
    # querying the database
    # for all the entries in it
    users = User.query.all()
    # converting the query objects
    # to list of jsons
    output = []
    for user in users:
        # appending the user data json
        # to the response list
        output.append({
            'public_id': user.public_id,
            'name': user.name,
            'email': user.email
        })

    res = jsonify({'users': output})
    res.headers['Access-Control-Allow-Origin'] = '*'
    return res

# def user_has_exceeded_send_email(current_user):

@app.route('/getinfo', methods=['GET'])
@token_required
def get_info(current_user):
    output = {}

    output['public_id'] = current_user.public_id
    output['name'] = current_user.name
    output['email'] = current_user.email
    output['monthly_limit'] = current_user.monthly_limit
```

```
        output['phone_number'] = current_user.phone_number
        output['income'] = current_user.income

        res = jsonify({'users': output})
        res.headers['Access-Control-Allow-Origin'] = '*'
        return res
    # route for logging user in

@app.route('/login', methods=['POST'])
def login():
    # creates dictionary of form data
    auth = request.form

    if not auth or not auth.get('email') or not auth.get('password'):
        # returns 401 if any email or / and password is missing
        return make_response(
            'Could not verify',
            401,
            {'WWW-Authenticate': 'Basic realm ="Login required !!!"'}
        )

    user = User.query\
        .filter_by(email=auth.get('email'))\
        .first()

    if not user:
        # returns 401 if user does not exist
        res = make_response(
            'Could not verify',
            401,
            {'WWW-Authenticate': 'Basic realm ="User does not exist !!!"'}
        )
        res.headers['Access-Control-Allow-Origin'] = '*'
        return res
```

```

if check_password_hash(user.password, auth.get('password')):
    # generates the JWT Token
    token = jwt.encode({
        'public_id': user.public_id,
        'exp': datetime.utcnow() + timedelta(minutes=24*60*10)
    }, app.config['SECRET_KEY'], algorithm="HS256")

    res = make_response(jsonify({'token': token}), 201)
    res.headers['Access-Control-Allow-Origin'] = '*'
    return res

# returns 403 if password is wrong
res = make_response(
    'Could not verify',
    403,
    {'WWW-Authenticate': 'Basic realm ="Wrong Password !!!"'})
)
res.headers['Access-Control-Allow-Origin'] = '*'
return res

# signup route
@app.route('/signup', methods=['POST'])
def signup():
    # creates a dictionary of the form data
    data = request.form

    # gets name, email and password
    name, email = data.get('name'), data.get('email')
    password = data.get('password')
    income = data.get('income') if data.get(
        'income') else default_info['income']
    monthly_limit = data.get('monthly_limit') if data.get(
        'monthly_limit') else default_info['limit']
    phone_number = data.get('phone_number') if data.get(
        'phone_number') else default_info['phone_number']

    # checking for existing user

```

```

user = User.query\
    .filter_by(email=email)\\
    .first()

if not user:
    # database ORM object
    user = User(
        public_id=str(uuid.uuid4()),
        name=name,
        email=email,
        password=generate_password_hash(password),
        income=income,
        monthly_limit=monthly_limit,
        phone_number=phone_number
    )
    # insert user
    db.session.add(user)
    db.session.commit()

    res = make_response('Successfully registered.', 201)
    res.headers['Access-Control-Allow-Origin'] = '*'

else:
    # returns 202 if user already exists
    res = make_response('User already exists. Please Log in.', 202)
    res.headers['Access-Control-Allow-Origin'] = '*'

return res

```

```

@app.route('/bills', methods=['GET'])
@token_required
def get_bills(current_user):
    bills = Bills.query.filter_by(user=current_user.public_id).all()
    if bills is None:
        bills = []
    res = make_response(jsonify({'bills': [i.serialize for i in bills]}),
201)
    res.headers['Access-Control-Allow-Origin'] = '*'

```

```

    return res

@app.route('/records', methods=['GET'])
@token_required
def get_record(current_user):
    records = Record.query.filter_by(user=current_user.public_id).all()
    if records is None:
        records = {}
    res = make_response(
        jsonify({'records': [i.serialize for i in records]}), 201)
    res.headers['Access-Control-Allow-Origin'] = '*'
    return res

@app.route('/records', methods=['POST'])
@token_required
def put_record(current_user):
    form = request.form
    if not form:
        res = make_response('could not add record no data received', 401)
        res.headers['Access-Control-Allow-Origin'] = '*'
        return res
    if not form.get('category') or (form.get('gain') is None) or not
form.get('amount'):
        res = make_response(
            'could not add record no enough data received', 401)
        res.headers['Access-Control-Allow-Origin'] = '*'
        return res
    record = Record(
        user=current_user.public_id,
        category=form.get('category') if form.get(
            'category') else default_info['category'],
        amount=form.get('amount'),
        gain=form.get('gain') == "True"
    )

```

```

db.session.add(record)
db.session.commit()

dt = datetime.utcnow()
record_this_month =
Record.query.filter_by(user=current_user.public_id).filter(db.extract(
    'year', Record.date_created) == dt.year, db.extract('month',
Record.date_created) == dt.month)
current_month_spending = sum(
    [-1*i.amount if i.gain else i.amount for i in
record_this_month.all()])
if current_month_spending >= current_user.monthly_limit:
    message = Mail(
        from_email='210419104166@smartinternz.com',
        to_emails=current_user.email,
        subject='Your Monthly expenses have exceeded your target
budget.',
        html_content=f'''
<strong>Hey {current_user.name}!</strong><br>

Your Monthly expenses have exceeded your target budget. <br>
Kindly visit the Expense application for more insights.<br>
Visit: expenso

Thank you! keep Tracking!<br>
Adios Amigos.!'''')
try:
    sg = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
    response = sg.send(message)
    # print(response.status_code)
    # print(response.body)
    # print(response.headers)
except Exception as e:

```

```
        print("email error", e)

    res = make_response("sucessfully added record", 201)
    res.headers['Access-Control-Allow-Origin'] = '*'
    return res

@app.route('/bills', methods=['POST'])
@token_required
def put_bills(current_user):
    form = request.form
    if not form:
        res = make_response('could not add record no data received', 401)
        res.headers['Access-Control-Allow-Origin'] = '*'
        return res
    if not form.get('amount') or not form.get('due_date') or not
form.get('amount') or not form.get('bill_name'):
        res = make_response(
            'could not add record no enough data received', 401)
        res.headers['Access-Control-Allow-Origin'] = '*'
        return res
    bills = Bills(
        user=current_user.public_id,
        amount=form.get('amount'),
        due_date=datetime.strptime(form.get('due_date'), "%Y-%m-
%d").date(),
        name=form.get('bill_name')
    )

    db.session.add(bills)
    db.session.commit()
    res = make_response("sucessfully added bill", 201)
    res.headers['Access-Control-Allow-Origin'] = '*'
    return res
```

```

@app.route('/dashboard', methods=['GET'])
@token_required
def dashboard(current_user):
    dt = datetime.utcnow()
    record_this_month =
Record.query.filter_by(user=current_user.public_id).filter(db.extract(
        'year', Record.date_created) == dt.year, db.extract('month',
Record.date_created) == dt.month)
    record_last_seven_days =
Record.query.filter_by(user=current_user.public_id).filter(
        Record.date_created > (dt-timedelta(days=7))).all()
    last_week_spending = sum(
        [-1*i.amount if i.gain else i.amount for i in
record_last_seven_days])
    current_month_spending = sum(
        [-1*i.amount if i.gain else i.amount for i in
record_this_month.all()])
    if dt.month > 1:
        last_month_spending =
Record.query.filter_by(user=current_user.public_id).filter(db.extract(
        'month', Record.date_created) == dt.month-1,
db.extract('year', Record.date_created) == dt.year).all()
    else:
        last_month_spending =
Record.query.filter_by(user=current_user.public_id).filter(db.extract(
        'month', Record.date_created) == 12,
db.extract('year', Record.date_created) == dt.year-1).all()
        income = current_user.income if current_user.income is not
None else 0
        balance = income - current_month_spending
        by_category =
record_this_month.filter_by(gain=False).with_entities(
            Record.category,
db.func.sum(Record.amount)).group_by(Record.category).all()
        category_list = {}

```

```

        for x, y in by_category:
            category_list[x] = y
        response_obj = make_response(jsonify({'last_week_spending':
last_week_spending, 'expense_by_category': category_list, 'income':
income, 'balance': balance,
                                         'monthly_limit':
current_user.monthly_limit, 'current_month_spending':
current_month_spending, 'last_month_spending': last_month_spending}), 201)
        response_obj.headers['Access-Control-Allow-Origin'] = '*'
        return response_obj

@app.route('/budget', methods=['POST'])
@token_required
def addbudget(current_user):
    form = request.form
    limit = form.get('budget')
    current_user.monthly_limit = limit
    db.session.add(current_user)
    db.session.commit()
    res = make_response("sucessfully added budget", 201)
    res.headers['Access-Control-Allow-Origin'] = '*'
    return res

@app.before_first_request
def create_tables():
    db.create_all()

if __name__ == "__main__":
    # setting debug to True enables hot reload
    # and also provides a debugger shell
    # if you hit an error while running the server
    # app.run(debug=True, host="0.0.0.0")
    serve(app, listen='*:5000')

```

## **13.2 Attachements:**

**Github link:**

<https://github.com/IBM-EPBL/IBM-Project-50636-1660919764>

**Project demo Link:**

<http://159.122.178.155:30634/>

**Video Link:**

<https://drive.google.com/file/d/1Sxr0wnTqdyXhla57Dv6Mtq7UWxZfht28/view?usp=sharing>