PERSONAL EXPENSE TRACKER APPLICATION

A PROJECT REPORT

Submitted by

R.BALAMURUGAN 210519106006

R.DEEPAK 210519106015

S.LINGESWARAN 210519106042

M.SASIKUMAR 210519106059

For the course

HX8001 – Professional Readiness for innovation, Employability and Entrepreneurship

in

ELECTRONICS AND COMMUNICATION ENGINEERING

DMI COLLEGE OF ENGINEERING

ANNA UNIVERSITY:CHENNAI 600 025

NOVEMBER- 2022

CONTENTS

1.INTRODUCTION

1.1 Project overview

Mobile applications are top in user convenience and have over passed the web applications in terms of popularity and usability. There are various mobile applications that provide solutions to manage personal and group expense but not many of them provide a comprehensive view of both cases. In this paper, we develop a mobile application developed for the android platform that keeps record of user personal expenses, his/her contribution in group expenditures, top investment options, view of the current stock market, read authenticated financial news and grab the best ongoing offers in the market in popular categories. The proposed application would eliminate messy sticky notes, spreadsheets confusion and data handling inconsistency problems while offering the best overview of your expenses. With our application can manage their expenses and decide on their budget more effectively.

1.2 Purpose

It also known as expense manager and money manager, an expense tracker is a software or application that helps to keep an accurate record of your money inflow and outflow. Many people in India live on a fixed income, and they find that towards the end of the month they don't have sufficient money to meet their needs.

2.LITERATURE SURVEY

2.1 Existing problem

The problem of current generation population is that they can’t remember where all of the money it is earned have gone and ultimately have to live while sustaining the little money they have left for their essential needs. In this time there is no such perfect solution which helps a person to track their daily expenditure easily and efficiently and notify them about the money shortage they have. For doing so they have to maintain long ledger’s or computer logs to maintain such data and the calculation is done manually by the user, which may generate error leading to losses. Not having a complete tracking

2.2 Reference

- https://nevonprojects.com/daily-expense-tracker-system/
- https://data-flair.training/blogs/expense-tracker-python/
- https://phpgurukul.com/daily-expense-tracker-using-php-and-mysql/
- https://ijarsct.co.in/Paper391.pdf
- https://kandi.openweaver.com/?landingpage=python_all_projects&utm_source=google&utm_medium=cpc&utm_campaign=promo_kandi_ie&utm_content=kandi_ie_search&utm_term=python_devs&gclid=Cj0KCQiAgribBhDkARIsAASA5bukrZgbI9UZxzpoyf0P-ofB1mZNxzc-okUP-3TchpYMclHTYFYiqP8aAmmwEALw_wcB

2.3 Problem Statement Definition

This Expense Tracker is a web application that facilitates the users to keep track and manage their personal as well as business expenses. This application helps the users to keep a digital diary. It will keep track of a user's income and expenses on a daily basis. The user will be able to add his/her expenditures instantly and can review them anywhere and anytime with the help of the internet. He/she can easily import transactions from his/her mobile wallets without risking his/her information and efficiently protecting his/her privacy. It is common to delete files accidentally or misplace files. This expense tracker provides a complete digital solution to this problem. Excel sheets do very little to help in tracking Furthermore, they don't have the advanced functionality of preparing graphical visuals automatically. Not only it will save the time of the people but also it will assure error free calculations. The user just has to enter the income and expenditures and everything else will be performed by the system. Keywords: Expense Tracker, budget, planning, savings, graphical visualization of expenditure.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map canvas

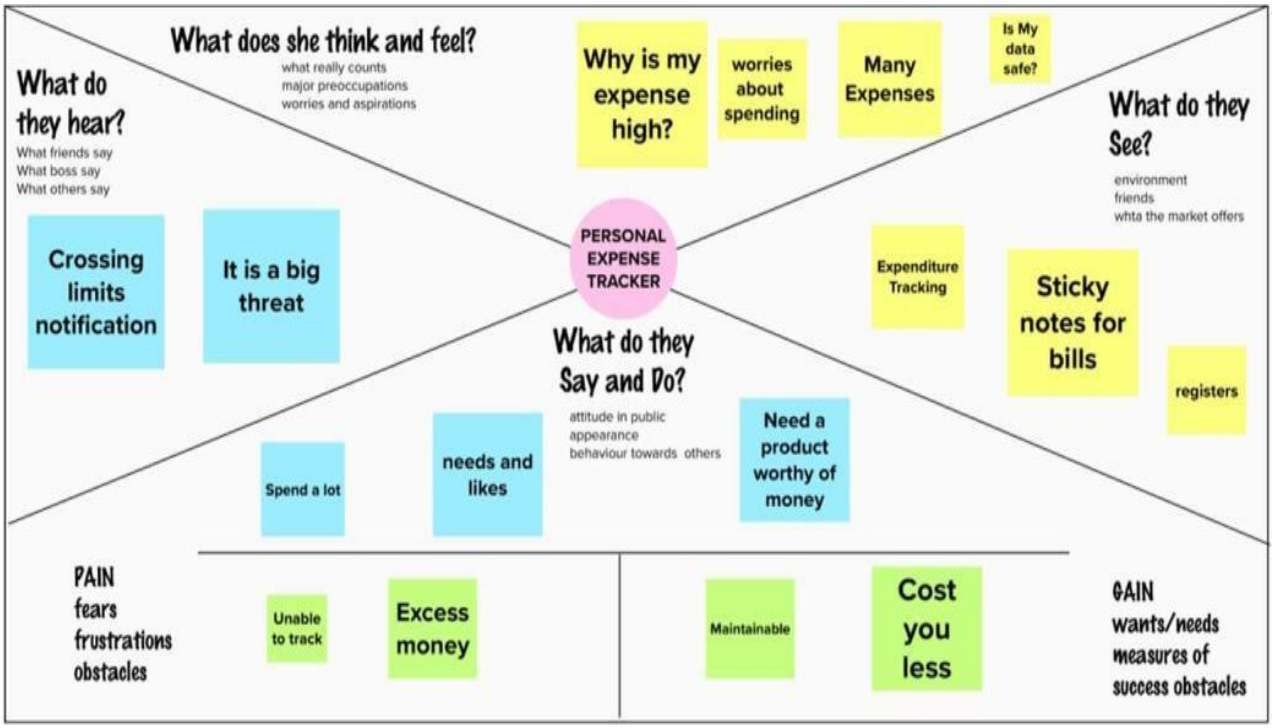


FIGURE 3.1

3.2 Ideation & Brainstorming

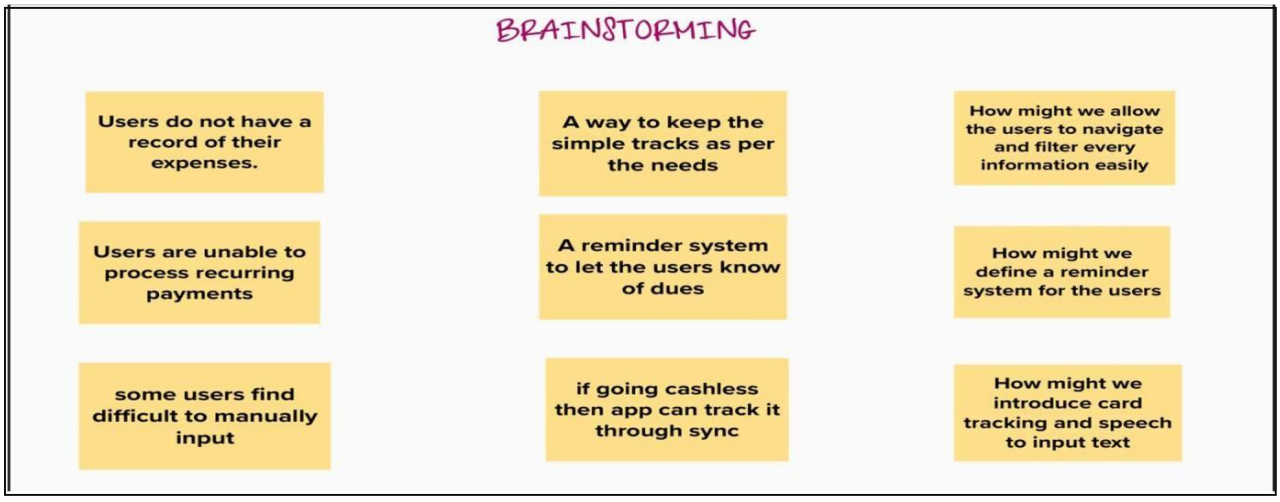


FIGURE 3.2

3.3 Proposed Solution

All people in the earning sector needs a way to manage their financial resources and track their expenditure, so that they can improve and monitor their spending habits. This makes them understand the importance of financial management and makes them better decisions in the future .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. The solution to this problem is, the people who gets regular payments can able to track their payments and void unwanted expenses. If the limit is exceeded the user will be notified with an email alert.

3.4 Proposed Solution Fit

The solution to this problem is, the people who gets regular payments can able to track their payments and avoid unwanted expenses. If the limit is exceeded the user will be notified with an email alert.

•Novelty / Uniqueness Notification can be receive through email.

• Social Impact / Customer Satisfaction Using this application one can track their personal expenses and frame a monthly/annual budget. If your expense exceeded than specified limit, the application will show you an alert message .This will make a impact on Mobile Banking for Customers' Satisfaction.

• Business Model (Revenue Model) Business people can use subscription/premium feature of this application to gain revenue.

• Scalability of the Solution The scalability of the application depends on security, the working of the application even during when the network gets down etc...

4.REQUIREMENT ANALYSIS

4.1 Functional requirement

Following are the functional requirements of the proposed solution.

- FR-1 User Registration ,Registration through Form Registration through Gmail Registration through LinkedIN
- FR-2 User Confirmation ,Confirmation via Email Confirmation via OTP
- FR-3 Tracking Expense Helpful insights about money management
- FR-4 Alert Message Give alert mail if the amount exceeds the budget limit FR-5 Category This application shall allow users to add categories of their expenses

4.2 Non Functional requirement

Following are the non-functional requirements of the proposed solution.

- NFR-1 Usability You will able to allocate money to different priorities and also help you to cut down on unnecessary spending
- NFR-2 Security More security of the customer data and bank account details.
- NFR-3 Reliability Used to manage his/her expense so that the user is the path of financial stability. It is categorized by week, month, and year and also helps to see more expenses made. Helps to define their own categories.
- NFR-4 Performance The types of expense are categories along with an option .Throughput of the system is increased due to light weight database support.
- NFR-5 Availability Able to track business expense and monitor important for maintaining healthy cash flow. NFR-6 Scalability The ability to appropriately handle increasing demands.

5.PROJECT DESIGN

5.1 Data Flow Diagrams

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is store

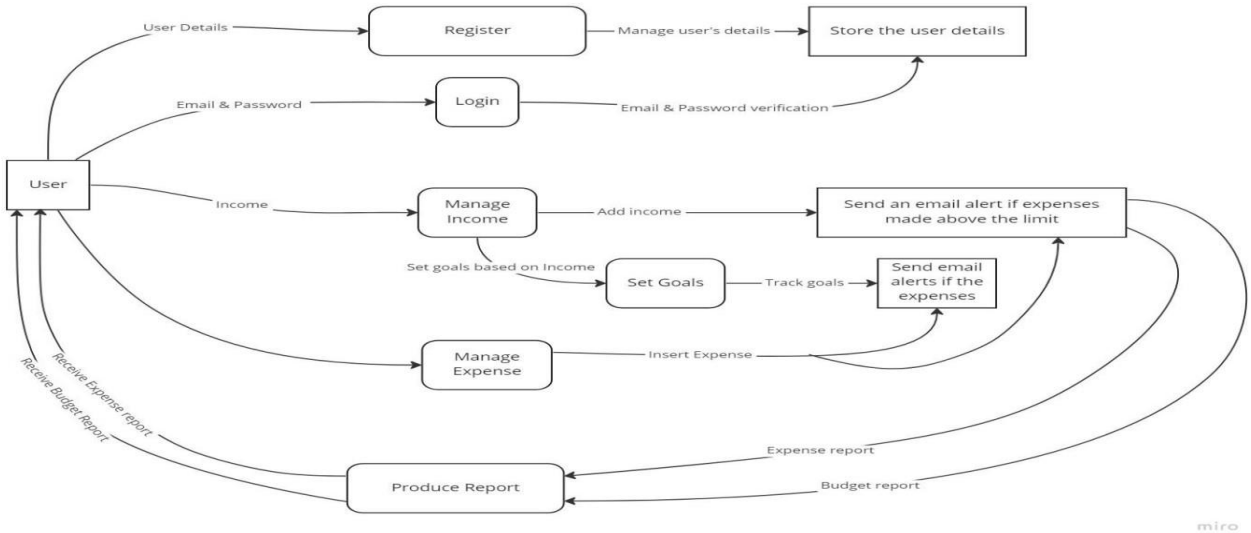


FIGURE 5.1

5.2 Solution &Technical Architecture

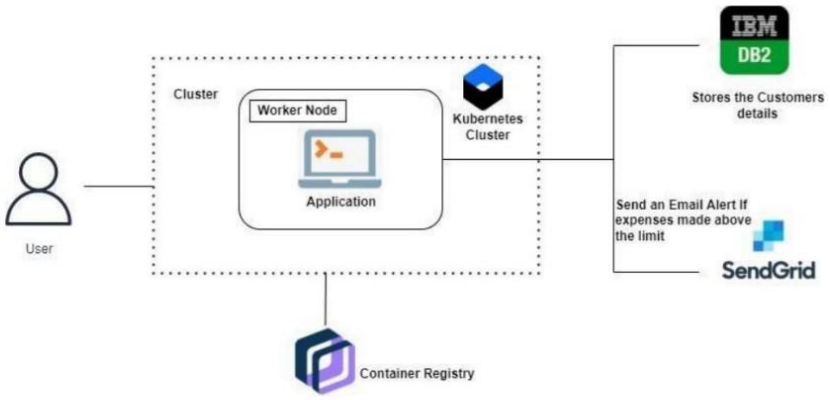


FIGURE 5.2

5.3 User Stories

Use the below template to list all the user stories for the product.

TABLE 5.3

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

		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email &click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail	I can register by entering the details	Medium	Sprint 1
	Login	USN-5	As a user, I can log into the application by entering email & password	I can access my dashboard	High	Sprint 1
	Dashboard	USN-6	As a user ,I can log into the dashboard and manage income	I can access my account / dashboard	High	Sprint 1
Customer (Web user)		USN-7	As a user, I can register for the application by Bank account.	I can access my account / dashboard	High	Sprint 1

Customer Care Executive		USN-8		I can Manage my money by viewing this report	Medium	Sprint 1
		USN-9	As a user, I can get an email if the money level is above the limit	I can receive alert email	High	Sprint 1
Admin	Responsibility	USN-10	As a system administrator ,trackthe user expenses anytime	I can track expense	High	Sprint 1

6.PROJECT PLANNING &SCHEDULING

6.1 Sprint Planning & Estimation

TABLE 6.1

Sprint	Functional Requirement(Epic)	User Story Number	User Story/Task	Story Points	Priority	Team Members
Sprint- 1	Registration, Login, Dashboard	USN-1 USN-2 USN-3	As a user, I can register for the application by entering my email, password, and confirming my password. Logging in takes to the	2	High	Deepak.R

			dashboard for the logged user.			
Sprint- 2	Workspace, Charts, Connect to IBM DB.	USN-3 USN-4	Workspace for personal expense tracking, Creating various graphs and statistics of customer's data, Connecting to IBM DB2	1	Medium	Balamurugan.R
Sprint- 3	Sendgrid	USN-3 USN-4 USN-6	Workspace for personal expense tracking, Usisg SendGrid to send mail to the user about their expenses	2	Medium	Lingeswaran.S
Sprint- 4	Docker, Cloud registry, kubernetes , Exposing	USN-7 USN-8, USN-9	CreateImage in Docker,Collabe with Kubernetes, cloud registry	2	High	Sasikumar M

6.2 Sprint Delivery Schedule

TABLE 6.2

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	6Days	24Oct2022	29Oct2022	20	29Oct2022
Sprint-2	20	6Days	31Oct2022	05Nov2022	18	06Nov2022
Sprint-3	20	6Days	07Nov2022	12Nov2022	15	14Nov2022
Sprint-4	20	6Days	14Nov2022	19Nov2022	19	21Nov2022

6.3 Reports From JIRA

Burndown Chart

Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.

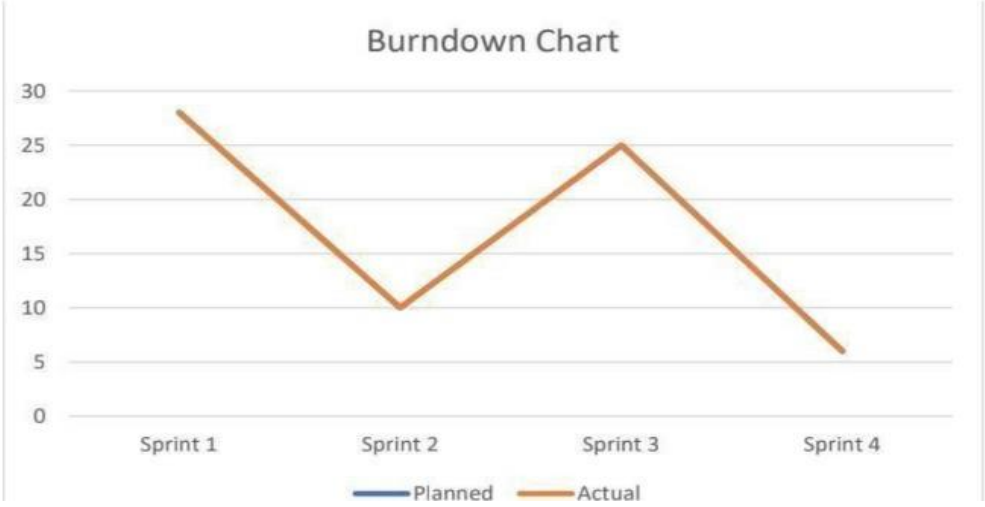
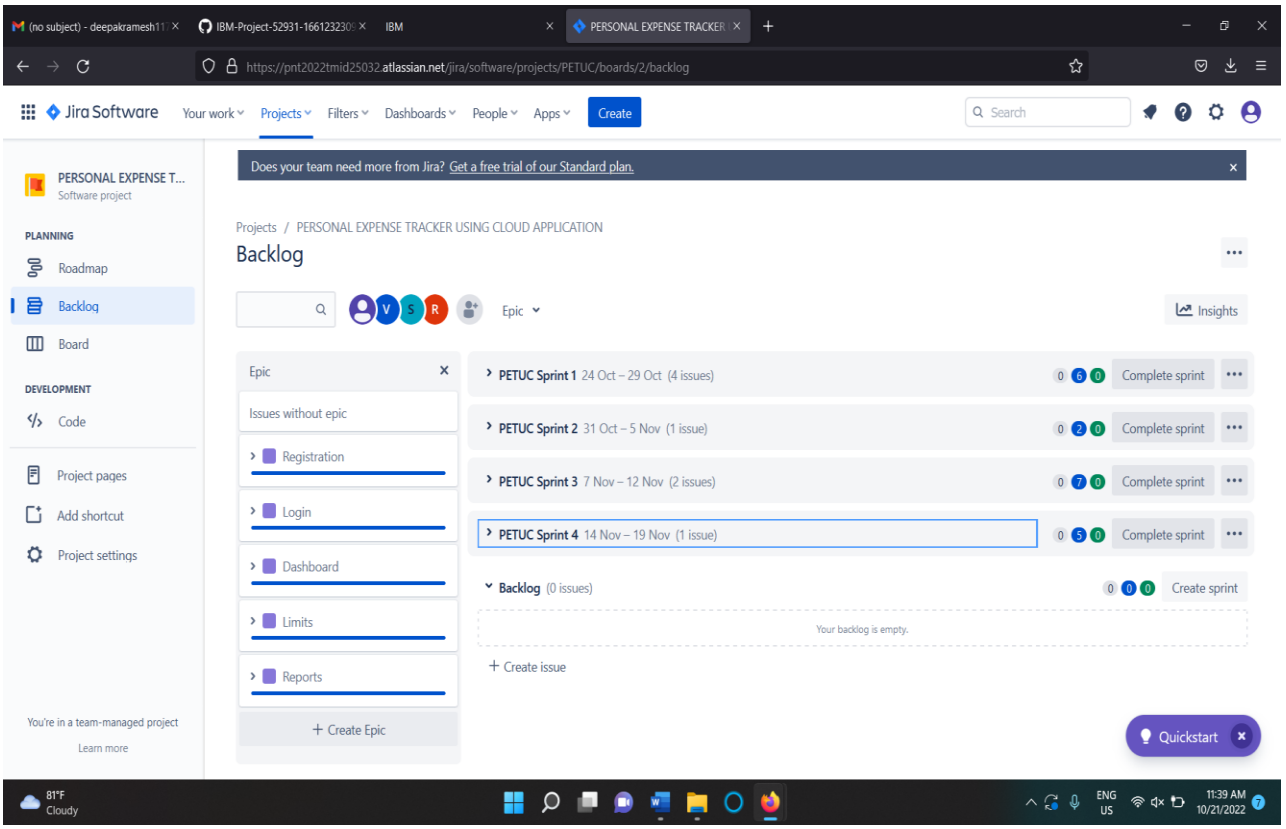
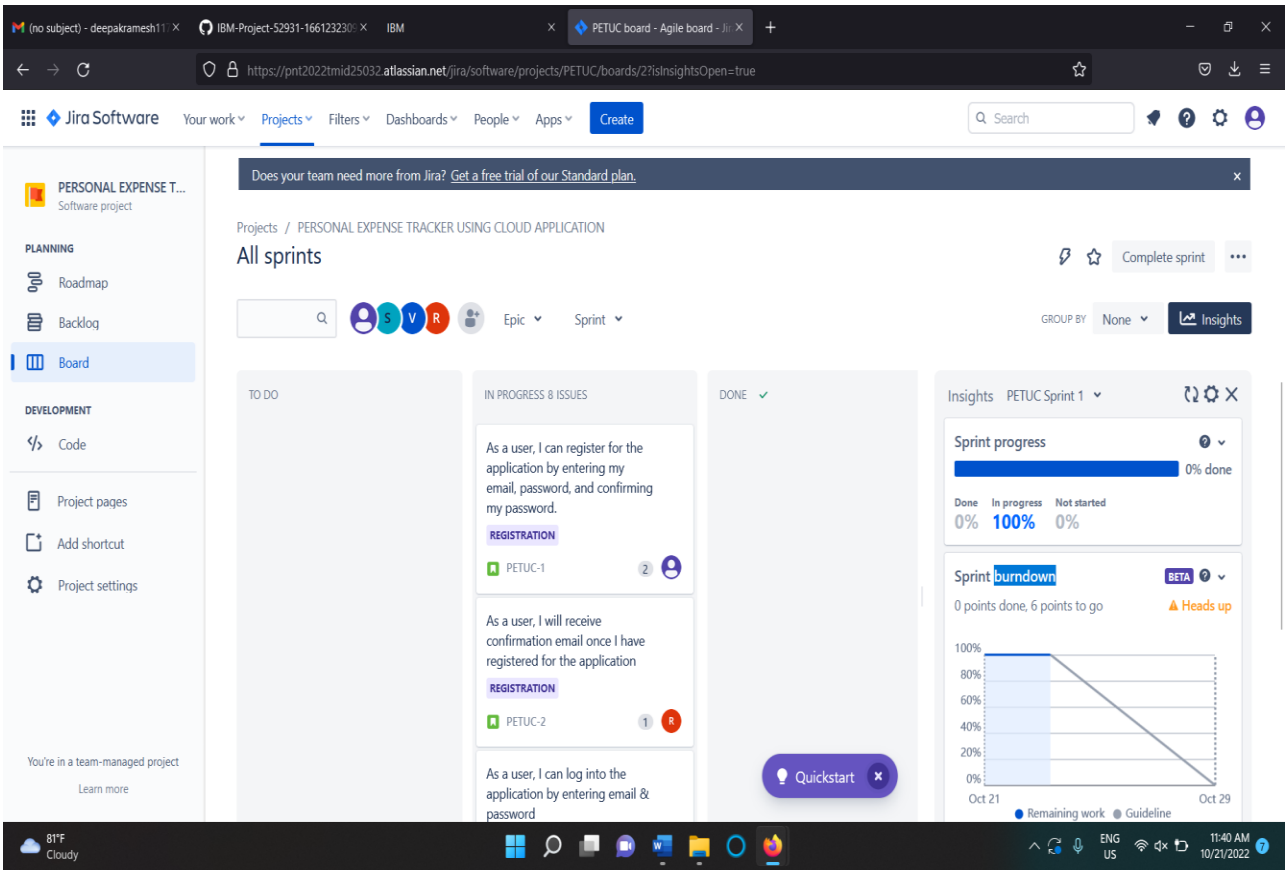


FIGURE 6.3


Jira Software
Your work
Projects
Filters
Dashboards
People
Apps
Create
Search
PERSONAL EXPENSE T...
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 USING CLOUD APPLICATION
Backlog
Epic
Insights
Issues without epic
Registration
Login
Dashboard
Limits
Reports
Create Epic
PETUC Sprint 1 24 Oct – 29 Oct (4 issues)
PETUC Sprint 2 31 Oct – 5 Nov (1 issue)
PETUC Sprint 3 7 Nov – 12 Nov (2 issues)
PETUC Sprint 4 14 Nov – 19 Nov (1 issue)
Complete sprint
Backlog (0 issues)
Create sprint
Your backlog is empty.
Create issue
Quickstart
81°F
Cloudy
ENG
US
11:39 AM
10/21/2022


Jira Software
Your work
Projects
Filters
Dashboards
People
Apps
Create
Search
PERSONAL EXPENSE T...
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 USING CLOUD APPLICATION
All sprints
Complete sprint
Epic
Sprint
GROUP BY
None
Insights
TO DO
IN PROGRESS 8 ISSUES
DONE
As a user, I can register for the application by entering my email, password, and confirming my password.
REGISTRATION
PETUC-1
As a user, I will receive confirmation email once I have registered for the application
REGISTRATION
PETUC-2
As a user, I can log into the application by entering email & password
Insights
PETUC Sprint 1
Sprint progress
0% done
Done
In progress
Not started
0%
100%
0%
Sprint burndown
BETA
0 points done, 6 points to go
Heads up
100%
80%
60%
40%
20%
0%
Oct 21
Oct 29
Remaining work
Guideline
Quickstart
81°F
Cloudy
ENG
US
11:40 AM
10/21/2022

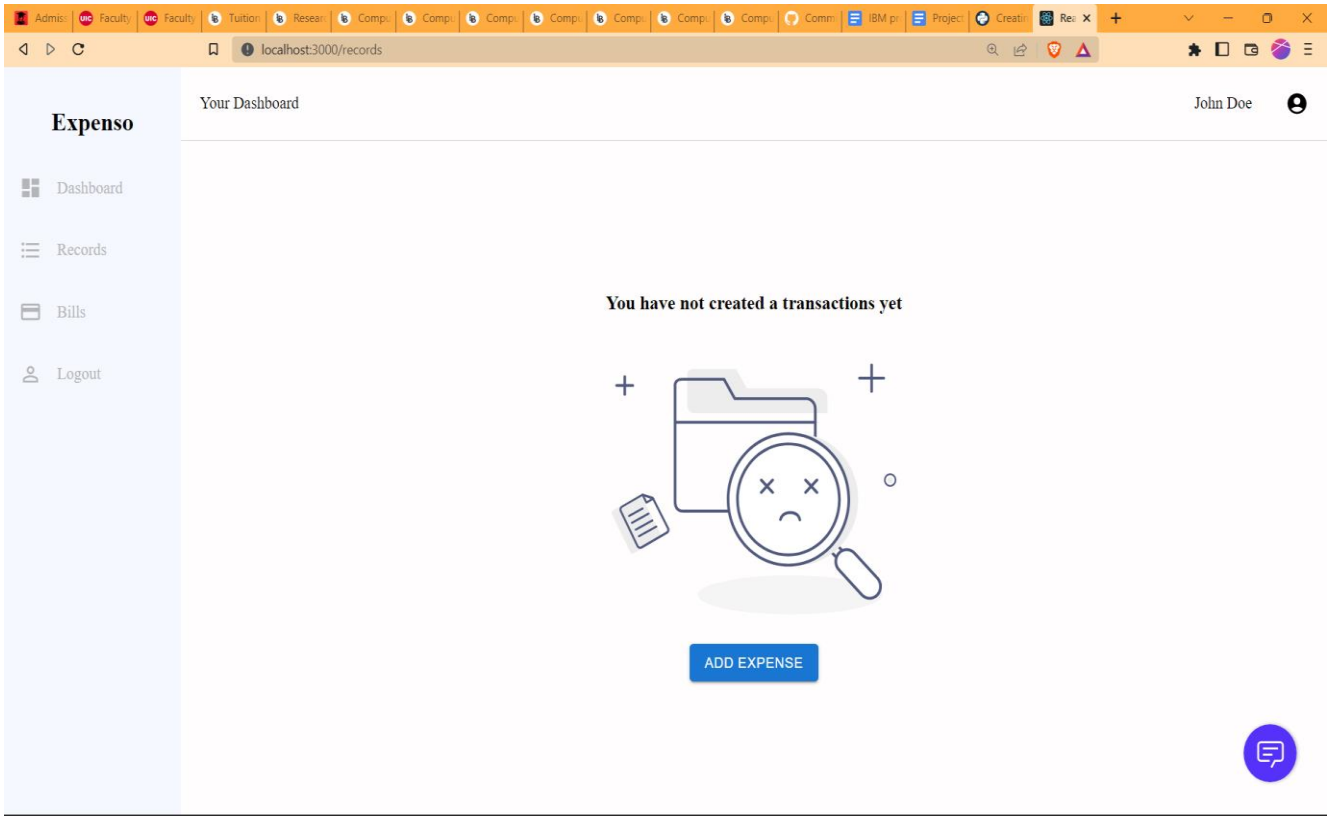
FIGURES 6.5

7.Coding And Solutioning:

7.1 Features 1 :

Adding Transactions

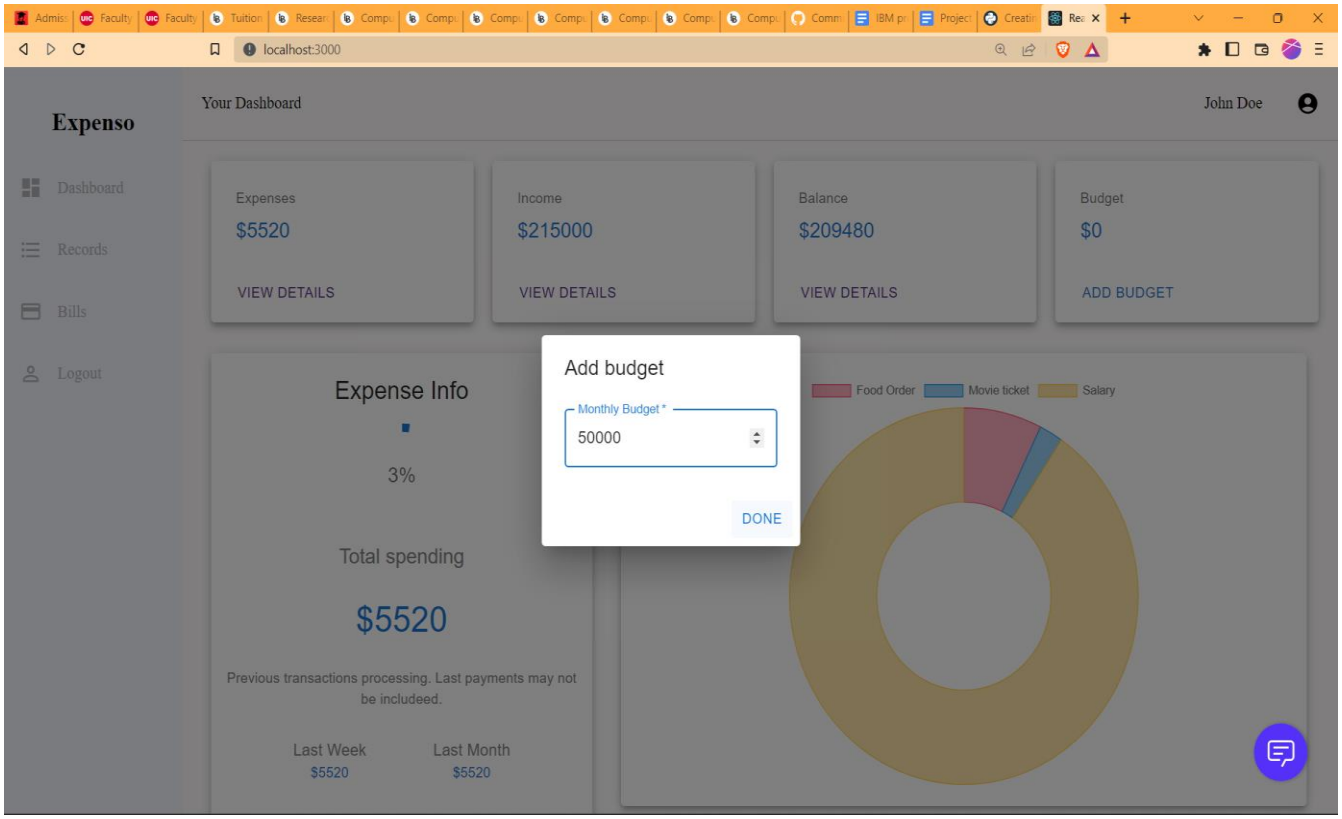
Users can add their transactions (expenses and income) to the records page and it will be displayed in the table. This can help them track how much they have earned and spent over time.



FIGURES 7.1

7.2 Features 2:

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



FIGURES 7.2

The other code features are submitted in github : refer the link ‘ GIT HUB ‘

8.TESTING:

8.1 TEST CASES:

- Login Page (Functional)
- Login Page (UI)
- Add Expense Page (Functional)

8.2 User Acceptance Testing:

1. Purpose of Document

The purpose of this document is to briefly explain the test coverage and open issues of the [ProductName] project at the time of the release to User Acceptance Testing (UAT).

TABLE 8.2.1

2. Defect Analysis

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	10	4	2	8	15
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	9	2	4	11	20
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	0	1	8
Totals	22	14	11	22	51

3. Test Case Analysis

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

TABLE 8.2.2

Section	Total Cases	Not Tested	Fail	Pass
Interface	7	0	0	7
Login	43	0	0	43
Logout	2	0	0	2

9. RESULTS

9.1 Performance Metrics

TABLE 9.1

			NFT - Risk Assessment						
o	Project Name	Scope/feature	Changes	Hardware Changes	Software Changes	Impact of Downtime	Load/Volume Changes	Risk Score	Justification
1	Personal Expense Tracker Application	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				
S.No	Project Overview	NFT Test approach	NFR - Met	Test Outcome	GO/NO-GO decision	Recommendations	Identified Defects (Detected/Closed/Open)	Approvals/SignOff
1	Plasma Donor	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:

One of the major pros of tracking spending is always being aware of the state of one's personal finances. Tracking what you spend can help you stick to your budget, not just in a general way, but in each category such as housing, food, transportation and gifts. While a con is that manually tracking all cash that is spent can be irritating as well as time consuming, a pro is that doing this automatically can be quick and simple.

Another pro is that many automatic spending tracking software programs are available for free. Having the program on a hand-held device can be a main pro since it can be checked before spending

occurs in order to be sure of the available budget. Another pro is that for those who just wish to keep tracking spending by hand with a paper and pen or by entering data onto a computer spreadsheet, these options are also available. Some people like to keep a file folder or box to store receipts and record the cash spent each day. A pro of this simple daily tracking system is that it can make one more aware of where the money is going way before the end of a pay period or month.

DISADVANTAGES:

A con with any system used to track spending is that one may start doing it then taper off until it's forgotten about all together. Yet, this is a risk for any new goal such as trying to lose weight or quit smoking. If a person first makes a budget plan, then places money in savings before spending any each new pay period or month, the tracking goal can help. In this way, tracking spending and making sure all receipts are accounted for only needs to be done once or twice a month.

Even with constant tracking of one's spending habits, there is no guarantee that financial goals will be met. Although this can be considered to be a con of tracking spending, it could be changed into a pro if one makes up his or her mind to keep trying to properly manage all finances. Another con that may occur when spending is being tracked is an error, but this may also be able to be changed into a pro if the person does regular tracking. Frequent tracking of cash spending can allow one to catch and correct errors so that the budget plan is still able to be adhered to despite the mistake.

11. CONCLUSION:

A comprehensive money management strategy requires clarity and conviction for decision-making. You will need a defined goal and a clear vision for grasping the business and personal finances. That’s when an expense tracking app comes into the picture. An expense tracking app is an exclusive suite of services for people who seek to handle their earnings and plan their expenses and savings efficiently. It helps you track all transactions like bills, refunds, payrolls, receipts, taxes, etc., on a daily, weekly, and monthly basis.

12. FUTURE SCOPE:

- Achieve your business goals with a tailored mobile app that perfectly fits your business.
- Scale-up at the pace your business is growing.
- Deliver an outstanding customer experience through additional control over the app.
- Control the security of your business and customer data.
- Open direct marketing channels with no extra costs with methods such as push notifications.
- Boost the productivity of all the processes within the organization.
- Increase efficiency and customer satisfaction with an app aligned to their needs.Seamlessly integrate with existing infrastructure.
- Ability to provide valuable insights.
- Optimize sales processes to generate more revenue through enhanced data collection.
- Robo Advisors: Get expert investment advice and solutions with the Robo-advisors feature. This feature will analyze, monitor, optimize, and improve diversification in investments by turning data into actionable insights in real-time.
- Chats: Equip your expense tracking app with a bot that can understand and answer all user queries and address their needs such as account balance, credit score, etc.
- Prediction: With the help of AI, your mobile app can predict your next purchase, according to your spending behavior. Moreover, it can recommend products and provide unique insights on saving money. It brings out the factors causing fluctuations in your expenses.

- Employee Travel Budgeting: Most businesses save money with a travel budgeting app as it helps prepare a budget for an employee's entire business trip. The feature will predict the expenses and allocate resources according to the prediction.

13. APPENDIX:

SOURCE CODE LINK:

```
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)

    # configuration1
    # 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 contex 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>
   <img          src="https://image.shutterstock.com/image-vector/whitecoupon-banner-word-over-260nw-
2213547155.jpg" alt="over budget!"/>
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')
```

OUTPUTS

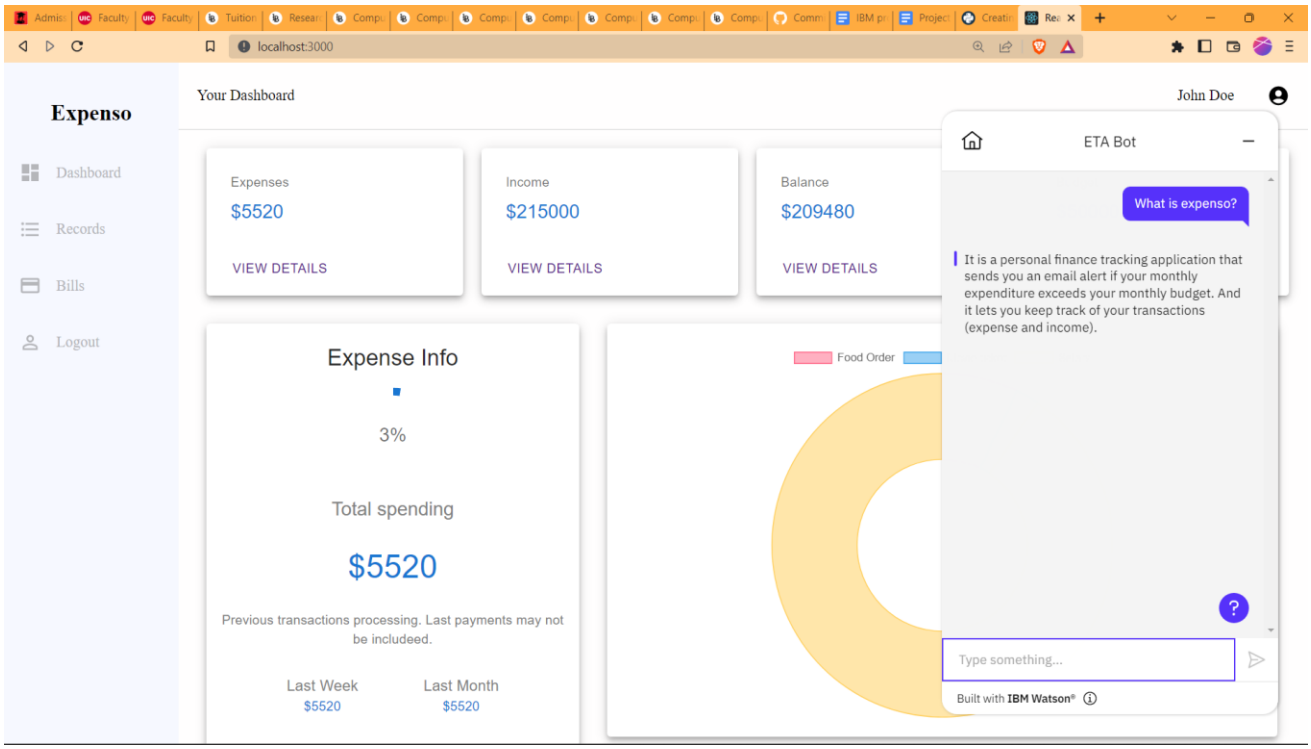


FIGURE DASHBOARD 1

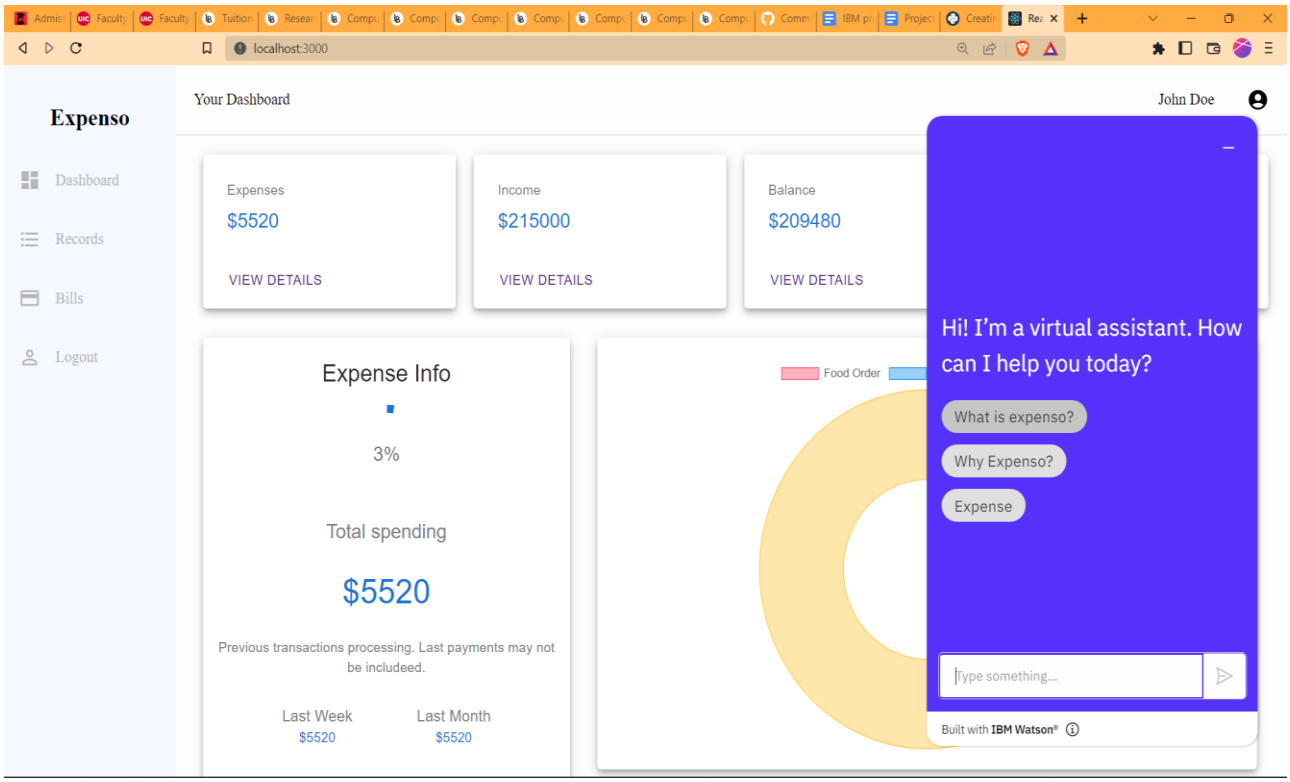


FIGURE WATSON CHAT 2

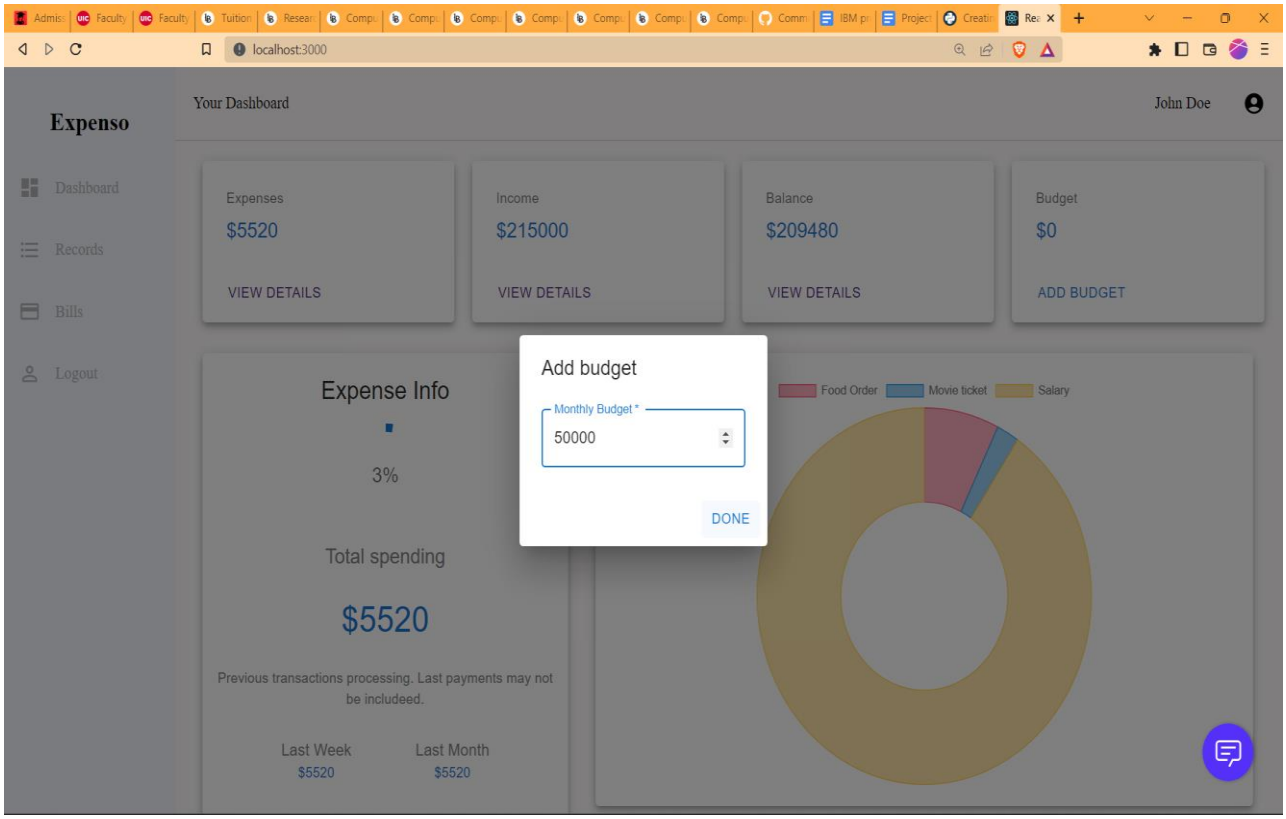


FIGURE ADD BUDGET 3

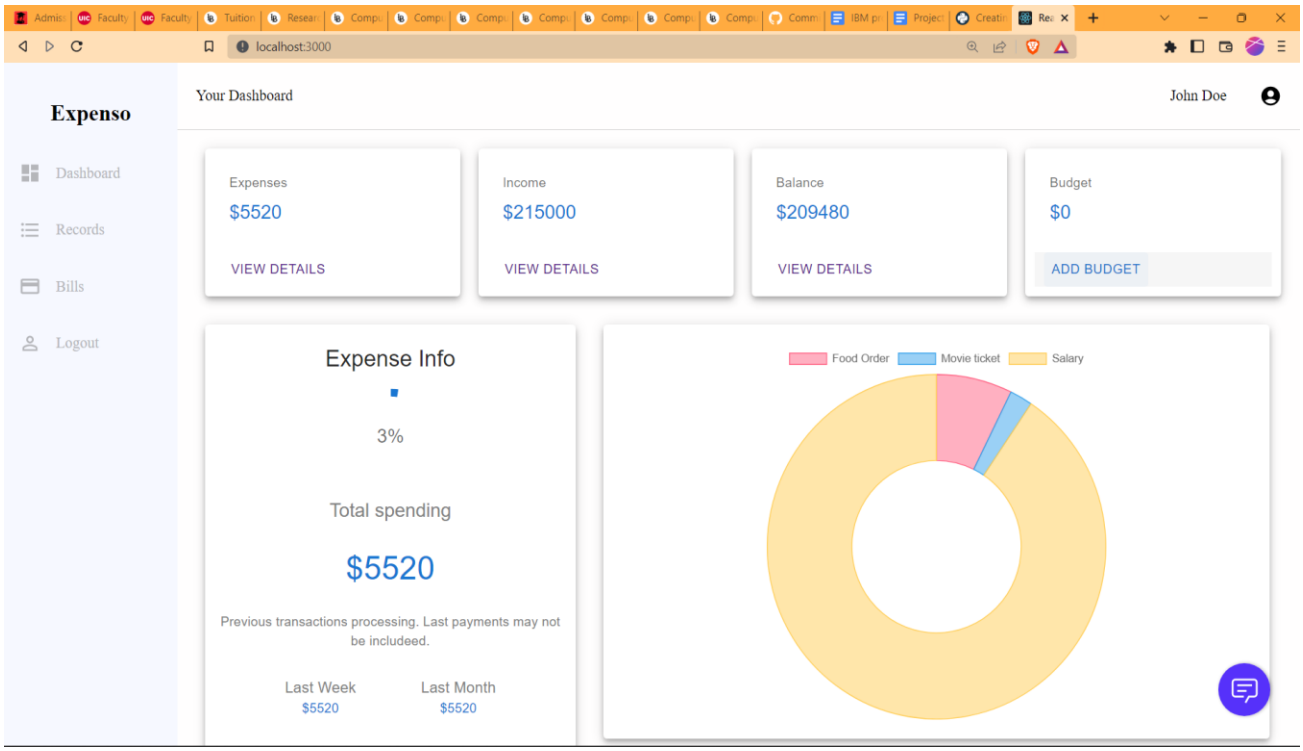


FIGURE EXPENSE CHART 4

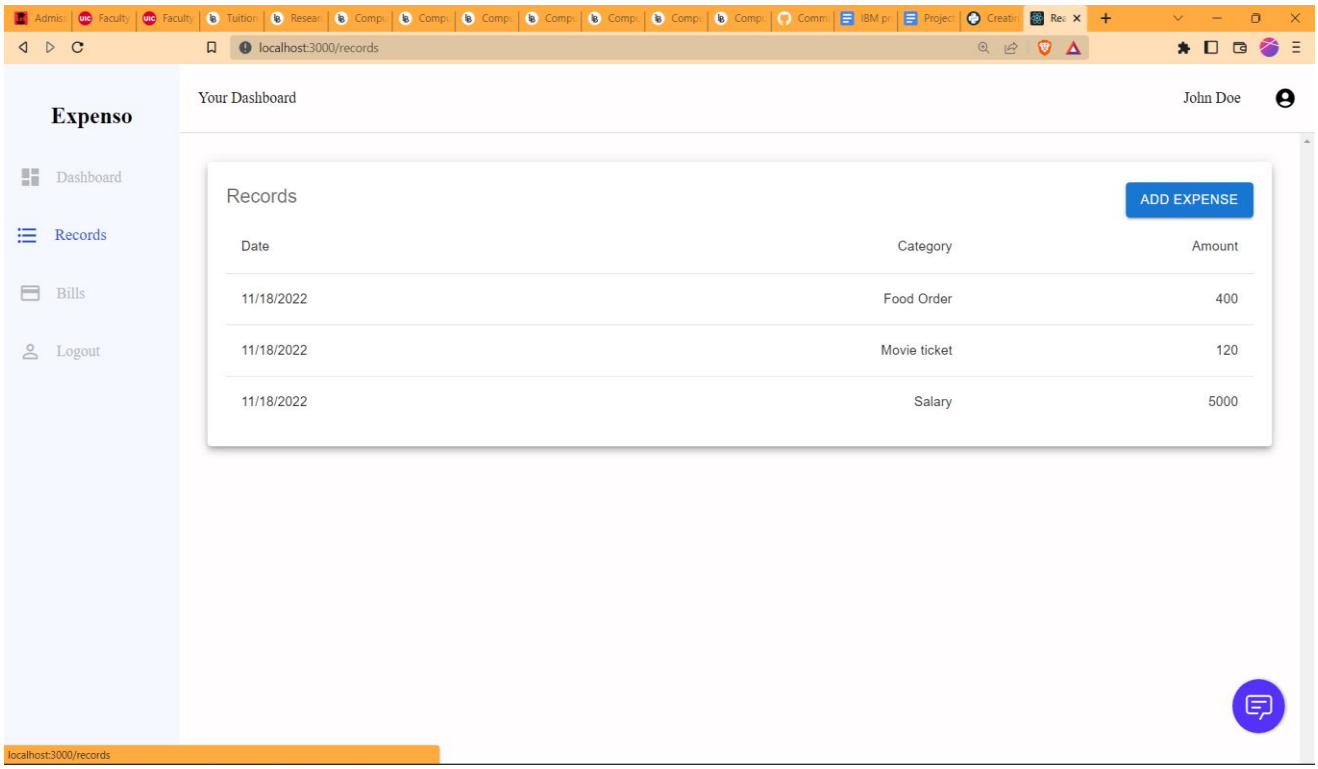


FIGURE RECORDS 5

GIT HUB :

https://github.com/IBM-EPBL/IBM-Project-52931-1661232309

DEMO VIDEO LINK :

https://youtu.be/t0V8X1kDi4I