International Institute of Information Technology Bangalore

Software Production Engineering

CS 816

December 15, 2023

Major Project Report

PennyPilot

Name of the student: Aakash Khot (IMT2020512)

Name of the student: Balaji Sankapal (IMT2020090)

Abstract

PennyPilot, our MERN app, is designed to streamline individual expense and income management. Its user-friendly interface simplifies financial record-keeping, reducing errors and saving time. Users can effortlessly track spending patterns, identify optimization opportunities, and make informed decisions about their personal finances.

PennyPilot leverages **DevOps** tools like Jenkins, Ansible (for IaC), Docker, and Git/GitHub to enhance its development and deployment processes. With continuous integration and deployment, automated infrastructure management, containerization, and version control, PennyPilot ensures an efficient and reliable expense management system.

Features of the Project

The Project is built using the MERN Stack. MERN stands for MongoDB, Express, React, Node, after the four key technologies that make up the stack.

1. **MongoDB:** Utilized as a NoSQL-based database, offering flexibility for efficient data storage.
2. **Express.js:** Chosen as the web framework for Node.js, it provides a minimal and flexible structure for building web applications.
3. **React.js:** Employed as a client-side JavaScript framework, enabling the creation of dynamic and interactive user interfaces.
4. **Node.js:** Utilized as the server-side JavaScript framework, facilitating the development of scalable and high-performance web applications.

Penny Pilot offers a user-friendly and feature-rich platform for expense management, providing several compelling reasons for individuals and businesses to use the application:

- **User-Centric Design:** Penny Pilot prioritizes usability with an intuitive and clean design. The straightforward login and signup process ensures accessibility for users of all technical levels.

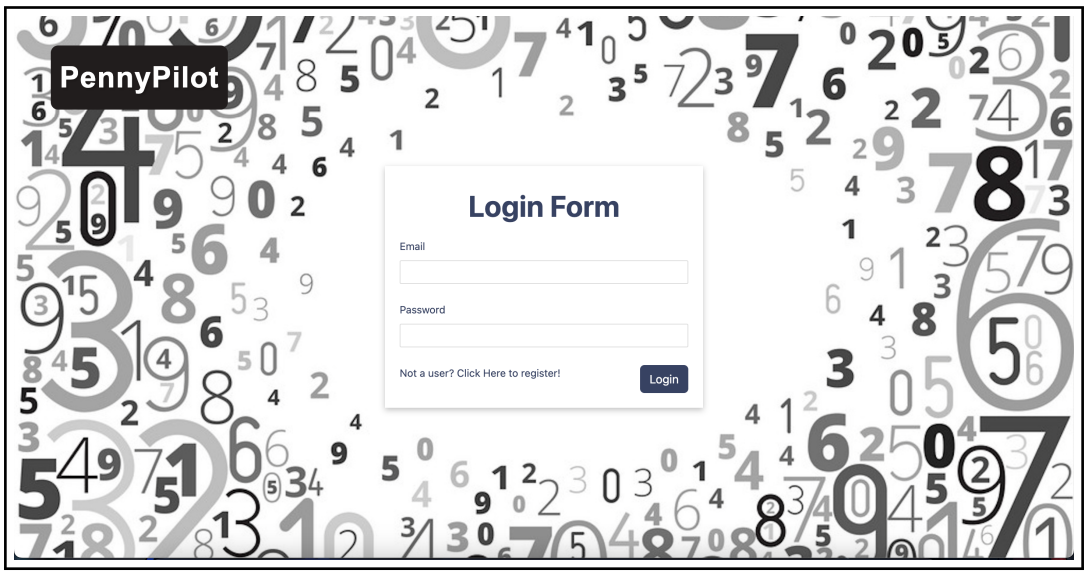


Figure 1: Login Page

Figure 2: Register Page

- **Effortless Expense Tracking:** The app simplifies expense recording, enabling users to add transactions quickly with just a few clicks for prompt and accurate financial tracking.

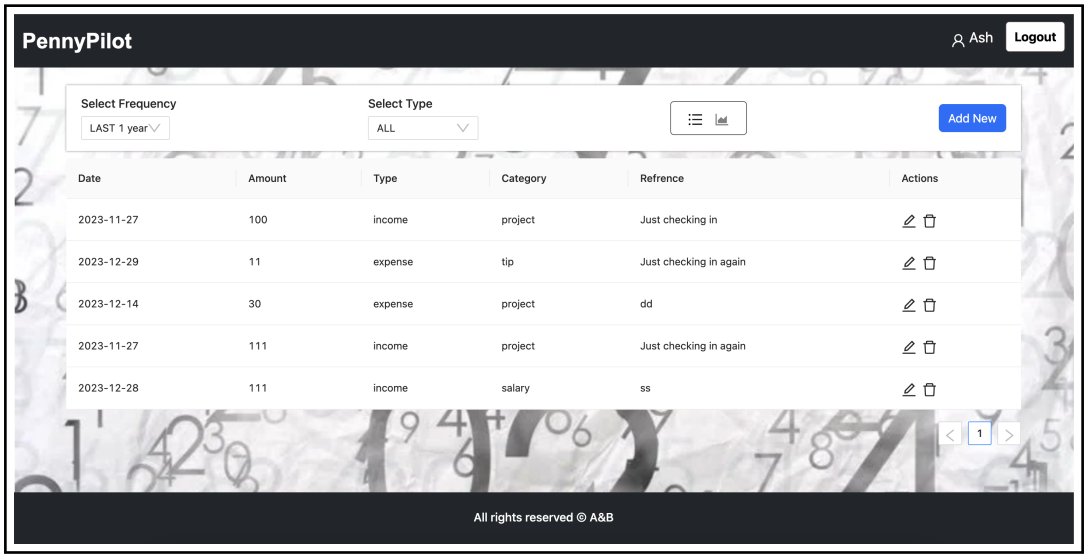


Figure 3: Home Page

- **Transaction Management:** Users can effortlessly add, edit, or delete transactions as needed, ensuring an up-to-date and accurate expense record with flexibility.
- **Robust Analytics:** Penny Pilot offers powerful analytics for users to gain insights into spending patterns, facilitating informed financial decision-making. Customizable dashboards and reports enhance the visualization of financial data.

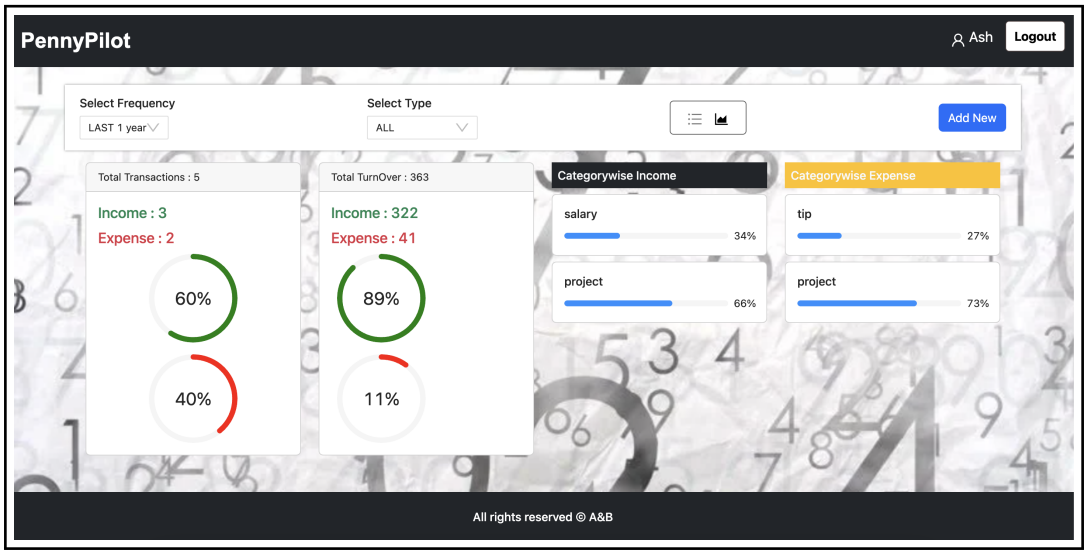


Figure 4: Analytics

In summary, Penny Pilot stands out as a valuable tool for anyone seeking an efficient, secure, and user-friendly solution for expense management. Its combination of essential functionalities, data analysis tools, and a commitment to user satisfaction positions it as a beneficial resource for individuals.

Why DevOps?

DevOps, a convergence of software development and IT operations, accelerates the software development lifecycle for frequent, high-quality releases. This approach emphasizes collaboration, communication, and automation between development and operations teams.
It encompasses continuous integration and delivery, automated testing, infrastructure as code, and continuous monitoring and feedback. By breaking down silos between teams, DevOps aims to boost efficiency, agility, and the overall quality of software development and operations.

- **Accelerated Releases:** DevOps shortens the development lifecycle, enabling faster and more frequent software releases.
- **Collaboration Emphasis:** Collaboration Emphasis
- **Automation Focus:** DevOps relies on automation, covering continuous integration, delivery, testing, infrastructure management, and monitoring.
- **Efficiency and Agility:** DevOps enhances efficiency and agility by promoting collaboration, breaking down team silos, and facilitating a quicker response to changes.
- **Quality Improvement:** The ultimate goal of DevOps is to improve the overall quality of software development and operations through streamlined processes and continuous feedback.

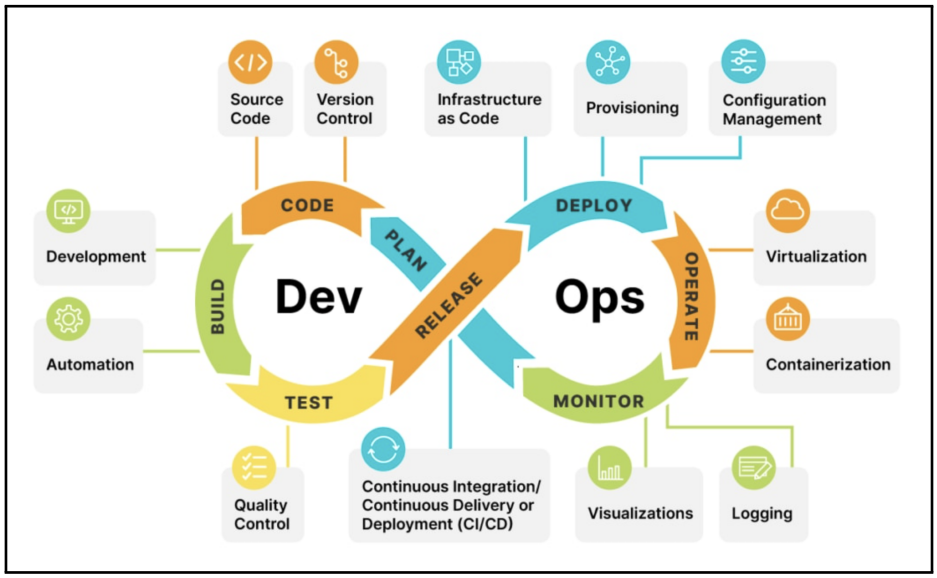


Figure 5: Cyclical Collection of Stages

Technology Stack

Languages

- **HTML:** Structures content on the web..
- **CSS:** Enhances web presentation with stylish design.
- **Javascript:** Adds interactive features for dynamic websites.

Framework

- **React.js:** Streamlines UI development in single-page applications.
- **Node.js:** Executes server-side JavaScript for scalable network applications.
- **Express.js:** Simplifies server-side JavaScript development for web applications.

Database

- **MongoDB:** NoSQL based Database

DevOps Tools

- **Git** and **Github:** Used for source code management and version control.
- **Jenkins:** Automates building and deploying code efficiently.
- **Docker** and **DockerHub:** : Used to containerise the application and store the container image remotely.
- **Ansible:** Automates IT tasks for configuration management.

- **ELK:** Analyzes and visualizes logs efficiently.

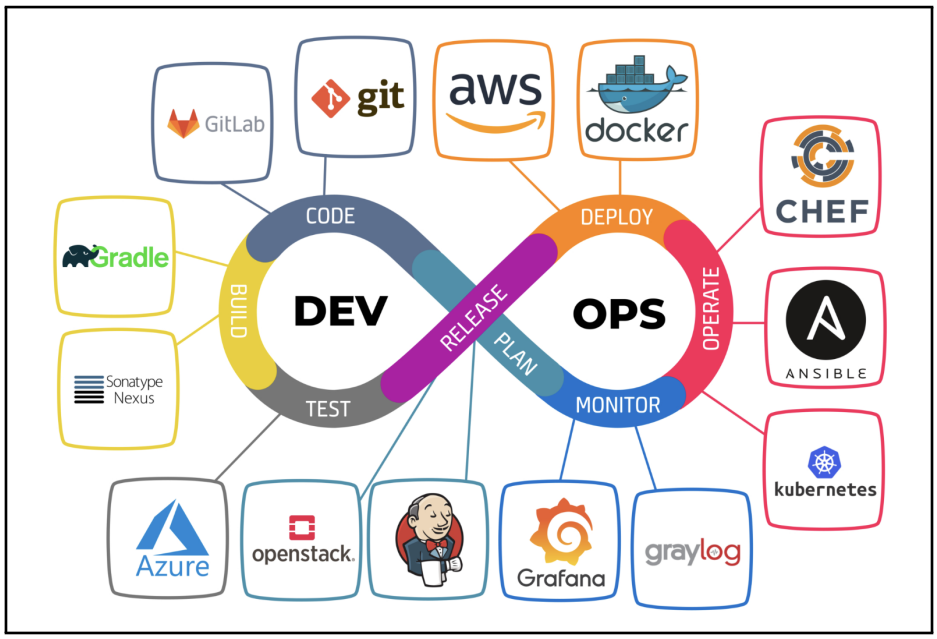


Figure 6: DevOps Tools

More Tools

- **Ant Design (antd):** UI library for React applications.
- **Jest and Supertest:** Testing framework and HTTP assertions.
- **Winston:** JavaScript logging library for Node.js applications.
- **Ngrok:** Securely exposes local web servers.

Link to Repositories

1. Github Repository:https://github.com/aakash2khot/PennyPilot
2. Docker Repository For Frontend: https://hub.docker.com/r/ashkt/pennypilot-frontend
3. Docker Repository for Backend: https://hub.docker.com/r/ashkt/pennypilot-backend

Project Development

Project Creation

Create a new React app on VS code and and ensure that react and nodejs are already installed.
Set up version control using Git and create a local repository for the project.
Push the local repository to a remote version control repository using GitHub.

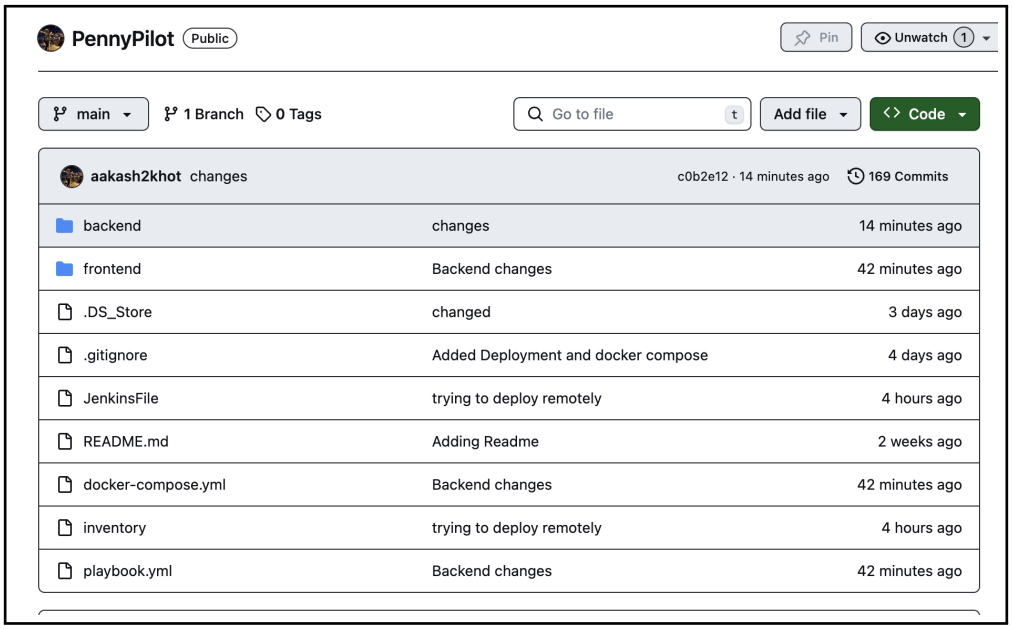
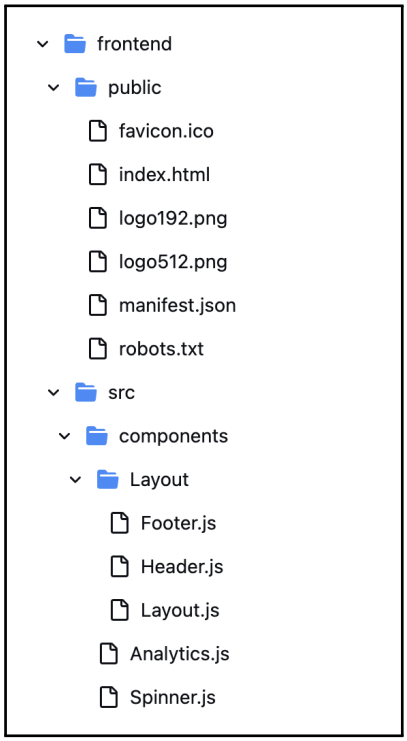
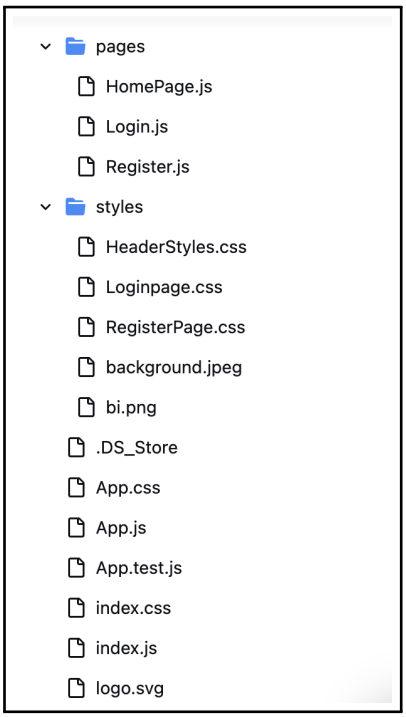


Figure 7: Github

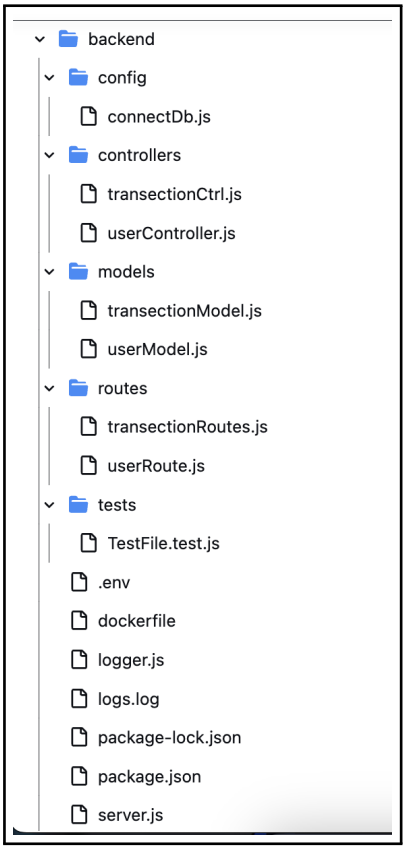
Here are few snippets of our Project Workflow:



(a) Frontend



(b) Frontend



(c) Backend

Figure 8: Workflow

API Documentation

User Authentication and Registration

1. **Login User:** The login functionality is achieved through the POST /api/v1/users/login endpoint. Users are required to provide their email and password for authentication.

2. **Register User:** User registration is handled by sending a request to the POST /api/v1/users /register endpoint. The request must include user details such as name, email, and password.

Transaction Management

1. **Add Transaction:** To add a new transaction, users utilize the POST /api/v1/transections /add-transection endpoint, providing transaction details. A successful addition results in a confirmation message.

2. **Edit Transaction:** Editing existing transactions is accomplished through the POST /api/v1/transections/edit-transection endpoint. Users need to provide the transaction ID and the desired changes in the payload. A successful edit prompts a confirmation message.

3. **Delete Transaction:** Transactions can be deleted by making a request to POST /api/v1 /transections/delete-transection, specifying the transaction ID. Successful deletion yields a confirmation message.

4. **Get All Transactions:** Retrieving all transactions involves a request to the POST /api/v1/transections/get-transection endpoint, requiring user ID, frequency, date range, and transaction type. The response comprises an array of transactions meeting the specified criteria.

Database

The database for the application was MongoDB. MongoDB is a document database that is free to use. Each entry in a MongoDB database is a document expressed in BSON, a binary representation of the data, rather than tables of rows or columns like SQL. This data can then be retrieved in JSON format by applications.

```
// MONGO_TEST_URL=mongodb+srv://ash:123@cluster0.eioachd.mongodb.net/expenseApp
// MONGO_URL=mongodb://mongo:27017/expenseApp
// NODE_ENV is 'test' when used for testing
const connectDb = async () => {
  try {
    const URL = process.env.NODE_ENV === 'test' ? process.env.MONGO_TEST_URL : process.env.MONGO_URL;
    await mongoose.connect(URL);
    console.log(`Server Running On mongo --  ${mongoose.connection.host}`.bgCyan.white);
    logger.info("Connected to Mongo Database");
  } catch (error) {
    logger.error(error);
    console.log(`${error}`.bgRed);
    console.log("Connecting to Mongo Database Failed")
  }
};
```

Figure 9: MongoDb Connection

In our setup, we employ MongoDB Atlas, the cloud version, for testing purposes. This ensures reliable and scalable testing in a cloud environment. For deployment, we opt for a MongoDB container sourced from Docker. This containerized approach facilitates consistency and ease of deployment, allowing for a seamless transition from testing to production environments.
Snippet of Connection to MongoDB container from Compass:-

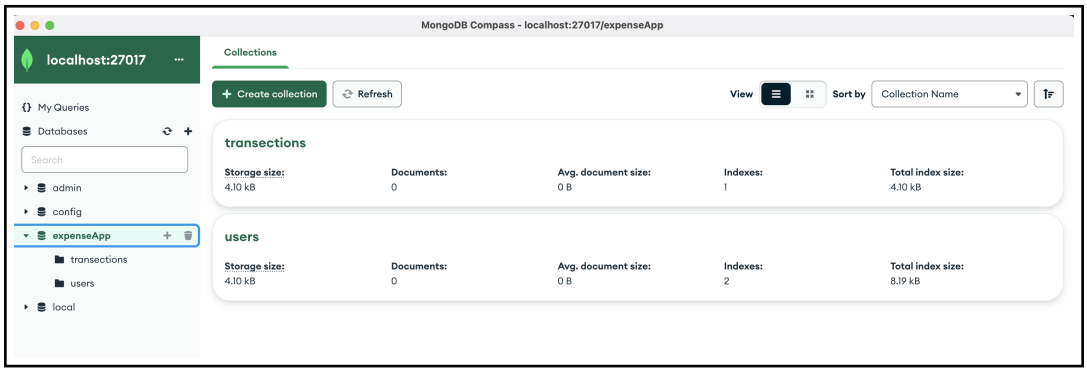


Figure 10: MongoDb Connection

Testing

We have performed Backend Testing using Supertest. Supertest is used for testing Node.js applications that communicate over HTTP and test for HTTP assertions.
The "TestFile " file has all test cases and it has extension .test.js, which Jest will be able to identify it as a testfile. We have written test cases for all functionalities (login, register, add transaction etc).

Initially we created a testing database in Atlas and used its URL as our production database and we executed npm run test.

```
PASS tests/TestFile.test.js
  User Controller - Login
    ✓ should log in a user with valid credentials (772 ms)
    ✓ should handle login failure with invalid credentials (40 ms)
    ✓ should handle login failure with missing credentials (36 ms)
  User Registration
    ✓ should register a new user (53 ms)
    ✓ should fail to register a user with an existing email (99 ms)
  Transaction
    ✓ should add a new transaction (49 ms)
    ✓ should edit an existing transaction (34 ms)
    ✓ should delete a transaction (43 ms)
    ✓ should handle invalid transaction ID (15 ms)

Test Suites: 1 passed, 1 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        2.984 s, estimated 4 s
Ran all test suites.
```

Figure 11: Testing Results

Later we added testing as a stage in Jenkins. So every time we run the pipeline, It will check whether the test cases are correct or not.

Automate Build and Testing

Automate the build and testing processes every time changes are pushed to the version control repository. Used Jenkins as the automation tool for continuous integration.
Make sure that jenkins are installed in your system with proper plugins. List of plugins that need to be present are Ansible, Docker, Docker pipeline, Git Client, Git Plugin, Github, Nodejs etc.
Created a new pipeline project on jenkins.To establish an effective CI/CD pipeline in Jenkins, we navigate to the Jenkins dashboard and create a new pipeline with an appropriate name. After configuring the pipeline settings, we define the pipeline script, which outlines the sequence of steps for execution.

```
pipeline{
    agent any
    tools {nodejs "nodejs"}
    environment {
        CI = 'true'
        registryfrontend = 'ashkt/pennypilot-frontend'
        registrybackend = 'ashkt/pennypilot-backend'
        DOCKERHUB_CREDENTIALS = credentials('DockerHubCred')
        // registryCredential = 'DockerHubCred'
        dockerimage = ''
    }

    stages{
        stage('Stage 1: Git Clone'){
            steps{
                git url:'https://github.com/aakash2khot/PennyPilot.git',
                branch: 'main'
                // credentialsId: 'Credential_Git'
            }
        }
        stage("Running Tests"){
            steps{
                script{
                dir('backend') {
                    sh 'npm install'
                    sh 'npm run test'
                }
                }

            }
        }
        stage('Build Frontend Docker Image') {
            environment {
                IMAGE_NAME = ''
            }
            steps {
                script{
                dir('frontend') {
                    dockerimage = sh '/usr/local/bin/docker build -t '+registryfrontend+':latest .'
                }
            }
        }
```

(a) Cloning/Testing

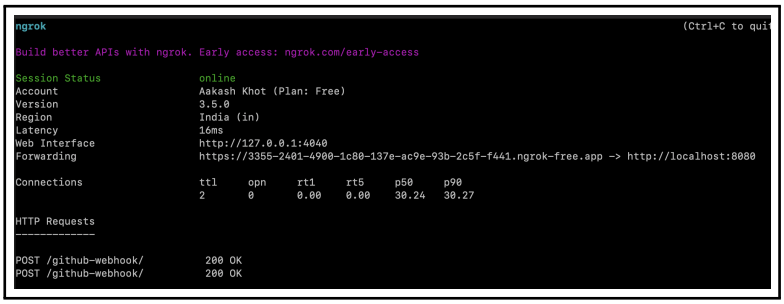
```
stage('Build Backend Docker Image') {
    steps {
        script{
        dir('backend') {
            dockerimage = sh '/usr/local/bin/docker build -t '+registrybackend+':latest .'
} }
}
}
stage('Push Frontend Image to dockerHub') {
    steps {
        script{
        dir('frontend') {

            sh '/usr/local/bin/docker push ' +registryfrontend+':latest'
        }
    } }
}
stage('Push Backend Image to dockerHub') {
    steps {
        script{
        dir('backend') {
            sh '/usr/local/bin/docker login -u '+DOCKERHUB_CREDENTIALS_USR+ " -p "+ DOCKERHUB_CREDENTIALS_PSW
            sh '/usr/local/bin/docker push ' +registrybackend+':latest'
        }
    } }
}
stage('Clean docker images'){
steps{
    script{
        sh '/usr/local/bin/docker container prune -f'
        sh '/usr/local/bin/docker image prune -f'
       }
   }
}
stage('Deploy') {
steps {
    sh '/Users/aakashkhot/Library/Python/3.12/bin/ansible-playbook playbook.yml -i inventory'
   }
}
```

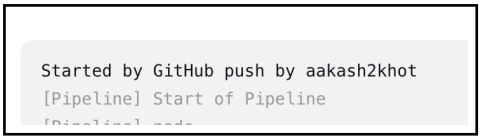
(b) Deploying

Figure 12: Pipeline Script

This script serves as the blueprint for the pipeline's operation, including tasks such as cloning the remote repository and deploying it using Ansible. It plays a vital role in ensuring the smooth and consistent flow of the DevOps workflow, automating critical processes in a structured manner.
On the same page where we define the Jenkins pipeline, we enable the "GitHub hook trigger for GITScm polling" option. This step ensures that a build is triggered automatically whenever a push is made to the remote GitHub repository.
To facilitate this seamless integration, we utilize ngrok, a tool that creates secure tunnels to localhost, allowing Jenkins to receive GitHub webhook events, ensuring immediate and automated builds upon repository updates.



(a) Ngrok



(b) Trigger

Figure 13: GitSCM polling

Docker

Docker is a platform and set of tools designed to facilitate the creation, deployment, and running of applications in lightweight, portable containers.

In our setup, we've crafted two Docker containers from frontend and backend images, alongside one container from the official MongoDB image.
The Dockerfile, a configuration file housing instructions, serves as a blueprint to build Docker images, providing consistency and efficiency in our application deployment process.

```
1   # Use an official Node.js runtime as a parent image
2   FROM node:18
3
4   # Set the working directory to /frontend
5   WORKDIR /frontend
6
7   # Copy the package.json and package-lock.json files to the container
8   COPY package*.json ./
9
10  # Install the dependencies
11  RUN npm install
12
13  # Copy the rest of the frontend code to the container
14  COPY . .
15
16  # Build the production-ready code
17  RUN npm run build
18
19  # Expose the port that the application will run on
20  EXPOSE 3000
21
22  # Start the frontend server
23  CMD ["npm", "start"]
```

(a) Docker file Frontend

```
1   # Use the base image with Node.js installed
2   FROM node:18
3
4   # Set the working directory to /frontend
5   WORKDIR /backend
6
7   # Copy the package.json and package-lock.json files to the container
8   COPY package*.json ./
9
10  RUN npm install
11
12  # Copy the rest of the backend code to the container
13  COPY . .
14
15  # Expose the port that the application will run on
16  EXPOSE 8082
17
18  # Start the backend server
19  CMD ["node", "server.js"]
20  # CMD ["npm", "run","dev"]
```

(b) Docker file Backend

Figure 14: Docker Files

Upon executing docker build, we successfully create Docker images for both the frontend and backend components. Subsequently, to make these images accessible and shareable, we push them to DockerHub. This ensures a centralized repository for our Docker images, allowing for easy distribution and deployment across various environments.

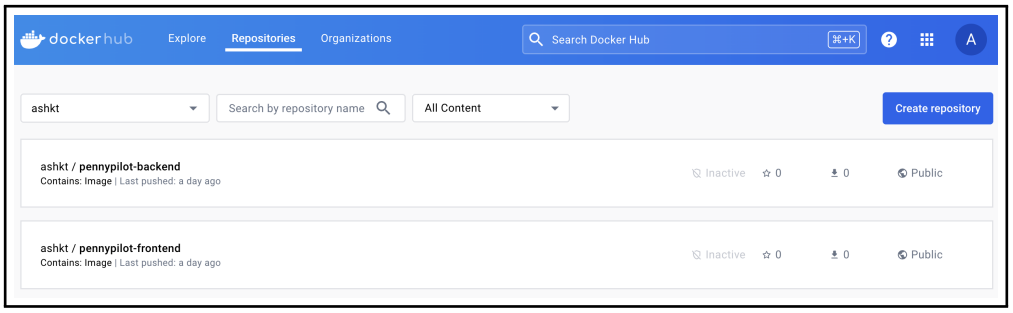
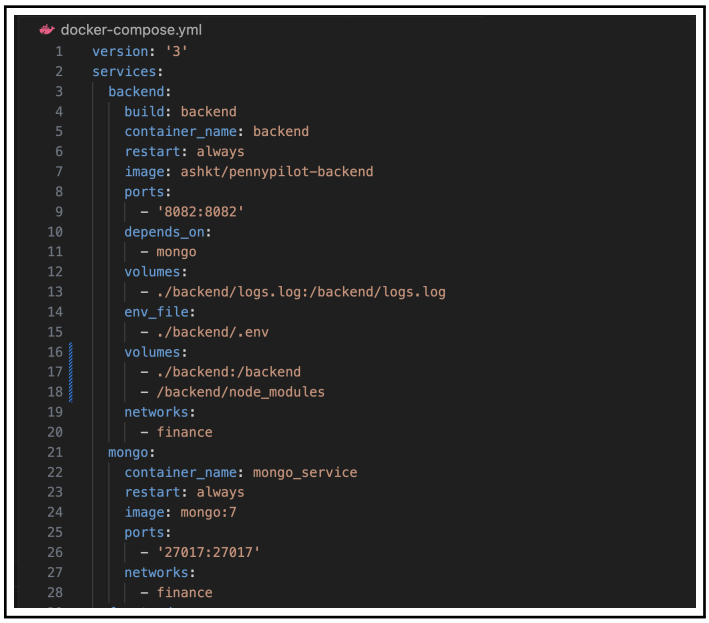


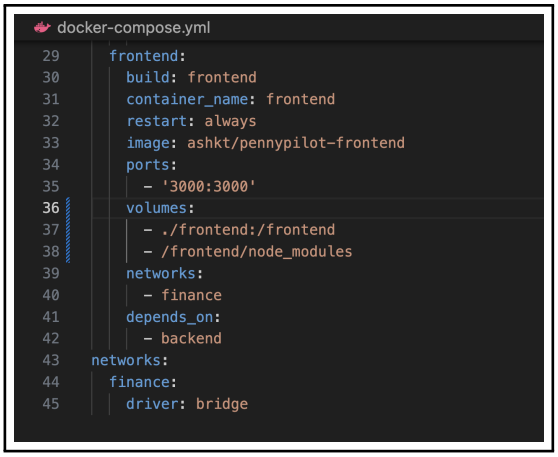
Figure 15: DockerHub

Docker Compose

Docker Compose is a tool used to define and manage multi-container Docker applications. We can use YAML file to define the services, networks, and volumes that make up a multi-container application.



(a) Backend – Mongo



(b) Frontend

Figure 16: Docker Compose file

The Docker Compose file orchestrates three services—backend, mongo, and frontend—forming the PennyPilot app. The backend and frontend services are built from local directories. They interact via a "finance" network. The mongo service uses the official MongoDB image, all connected through this custom network, facilitating efficient communication between components. The configuration ensures seamless deployment and coordination of the PennyPilot application's backend, frontend, and MongoDB services.

The backend service in the Docker Compose code utilizes a volume to connect the local log file (./backend/logs.log) with the container directory (/backend/logs.log). This volume setup enables access to log data.

Deployment

Implemented Ansible as an Infrastructure as Code (IaaC) tool to ensure consistent project environments. Set up Ansible playbook, a set of tasks, to manage and pull the Docker container images from Docker Hub.
Make sure that ansible is installed and configured in your system.

The managed nodes are the nodes on which the deliverables are distributed, and there is only one control node. Only the control node requires the installation of Ansible.
Ansible uses ssh to connect to the managed nodes. Ansible provides a collection of commands in the form of a yml file that must be run on the managed node. It's known as a playbook.

playbook.yml

```yaml
---
- name: Deploying Project.
  hosts: localhost
  #For remote access
  # hosts: remote
  vars:
    ansible_python_interpreter: /usr/local/bin/python3
  tasks:
    - name: Build and start containers
      command: "/opt/homebrew/bin/docker-compose -f
      /Users/aakashkhot/Documents/IIITB_Sem7/PennyPilot/docker-compose.yml up -d"

```

Figure 17: Playbook

We also provide information on the managed nodes in addition to the playbook. This is referred to as inventory.

inventory

```
[localhost]
127.0.0.1 ansible_connection=local ansible_user=aakashkhot
[remote]
117.205.13.254 ansible_host=117.205.13.254 ansible_user=balajibabasahebsankapal

```

Figure 18: Inventory

Successful build of Jenkins Pipeline

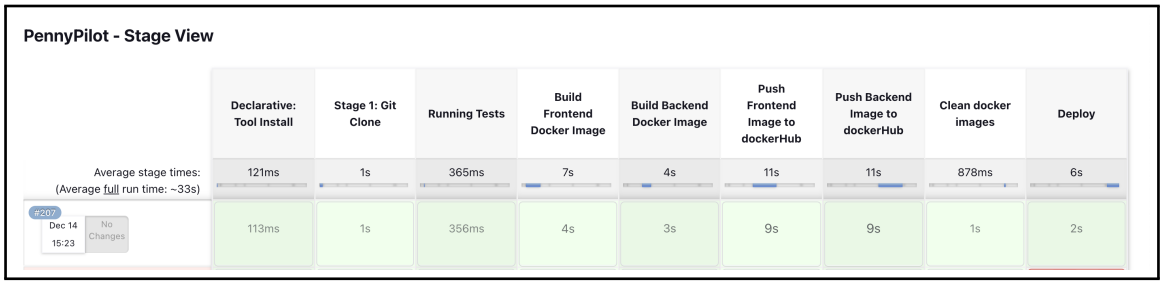


Figure 19: Pipeline – Build

Successful creation of Docker container

```
aakashkhot@Aakashs-MacBook-Pro ~ % docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS          PORTS                      NAMES
7f501955c062   ashkt/pennypilot-frontend  "docker-entrypoint.s…"   28 seconds ago   Up 26 seconds   0.0.0.0:3000->3000/tcp     frontend
e48f44deaa0c   ashkt/pennypilot-backend   "docker-entrypoint.s…"   28 seconds ago   Up 26 seconds   0.0.0.0:8082->8082/tcp     backend
a405fd5dc90b   mongo:7                    "docker-entrypoint.s…"   28 seconds ago   Up 26 seconds   0.0.0.0:27017->27017/tcp   mongo_service
```

Figure 20: Containers

Continuous Monitoring

We need to continuously monitor the project to check if the nodes are running properly and managed.
ELK (Elasticsearch, Logstash, Kibana) is a powerful open-source stack for log and data analysis, providing tools for centralized logging, log parsing, and visualization.

As we already have log file generated, we will directly upload the log file in elastic cloud and visualize the data in Kibana.

```
2023-12-14T12:49:07.697Z  info: Fetched all Transaction you have done!
2023-12-14T12:49:39.688Z  info: Transaction created successfully of Amount 30 as income
2023-12-14T12:49:40.961Z  info: Fetched all Transaction you have done!
2023-12-14T12:49:40.968Z  info: Fetched all Transaction you have done!
2023-12-14T12:52:11.866Z  info: Fetched all Transaction you have done!
2023-12-14T12:52:16.510Z  info: Fetched all Transaction you have done!
2023-12-14T12:52:20.205Z  info: Fetched all Transaction you have done!
2023-12-14T12:52:36.312Z  info: Transaction created successfully of Amount 111 as income
2023-12-14T12:52:38.486Z  info: Fetched all Transaction you have done!
2023-12-14T12:52:38.488Z  info: Fetched all Transaction you have done!
2023-12-14T12:54:40.284Z  info: User Found - Login Successful
2023-12-14T12:54:40.313Z  info: Fetched all Transaction you have done!
2023-12-14T12:54:40.315Z  info: Fetched all Transaction you have done!
2023-12-14T13:17:46.244Z  info: Server is running on port 8082
2023-12-14T13:17:46.253Z  info: Connected to Mongo Database
2023-12-14T14:46:01.349Z  info: Server is running on port 8082
2023-12-14T14:46:01.357Z  info: Connected to Mongo Database
2023-12-14T14:46:19.403Z  info: New User successfully Registered
2023-12-14T14:46:25.216Z  info: User Found - Login Successful
```

Figure 21: Logs

Now Visualizing log file on Kibana according to either timestamp with default 30 min interval or on the basis of count of records.

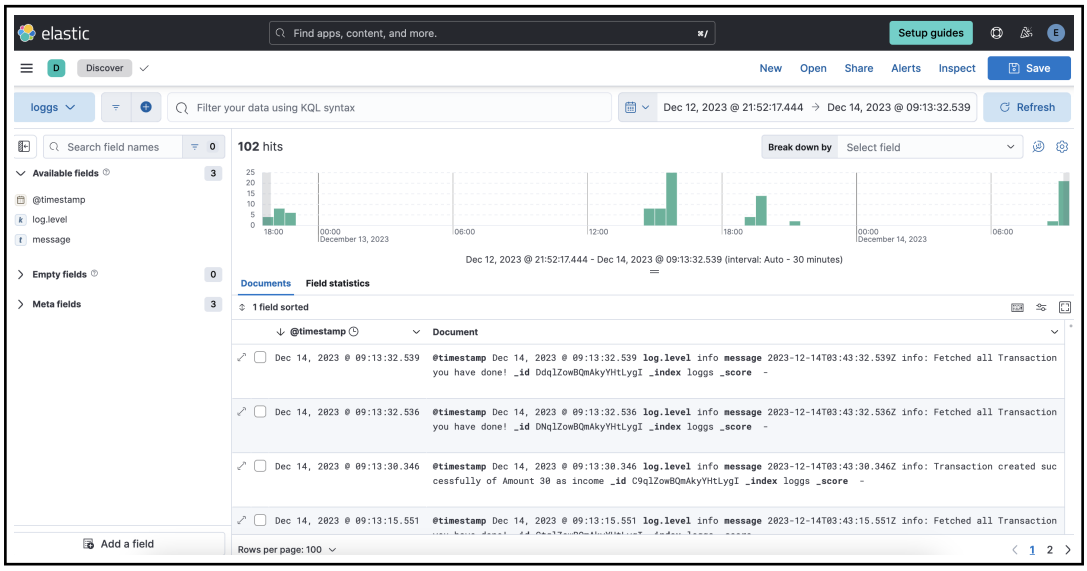


Figure 22: Viewing Index in Discover

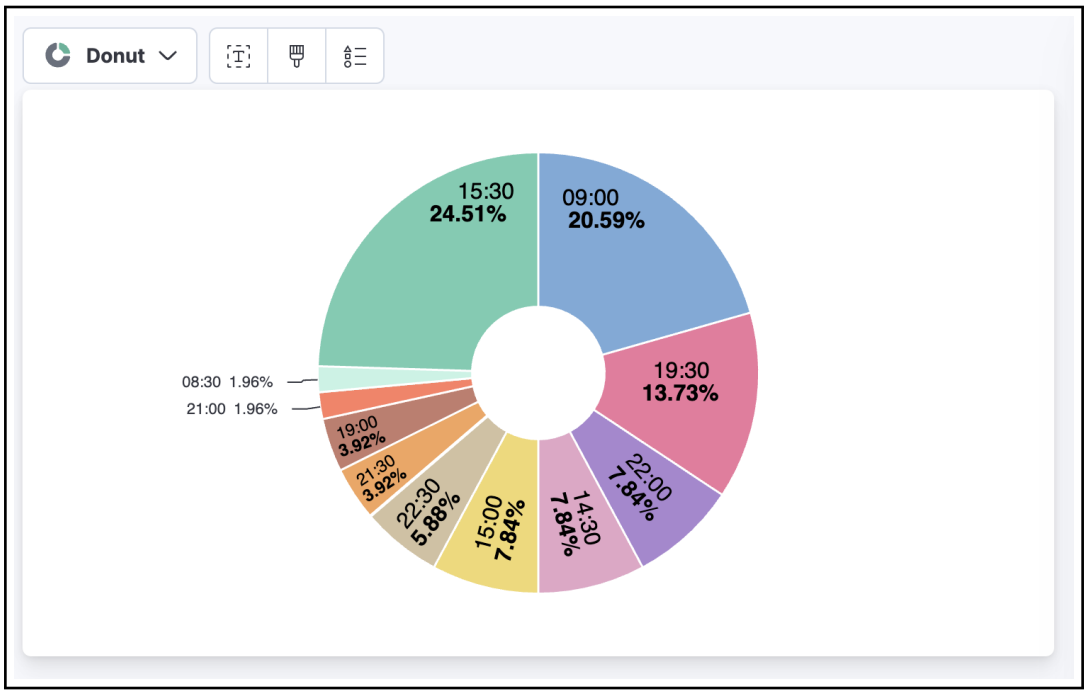


Figure 23: Donut structure

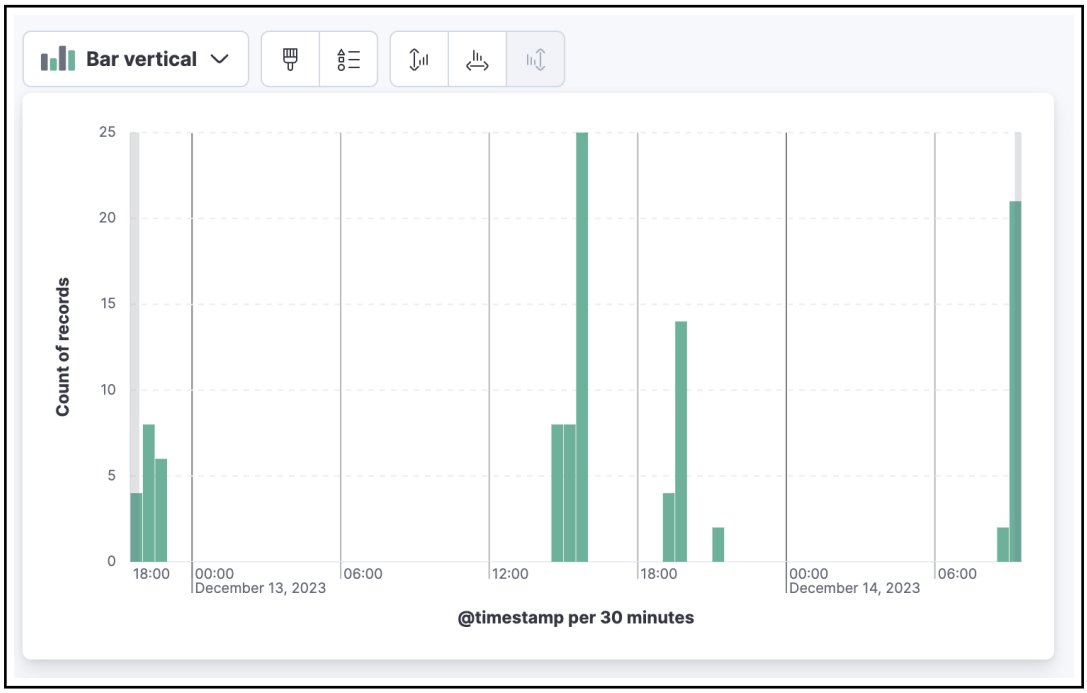


Figure 24: Veritcal bar

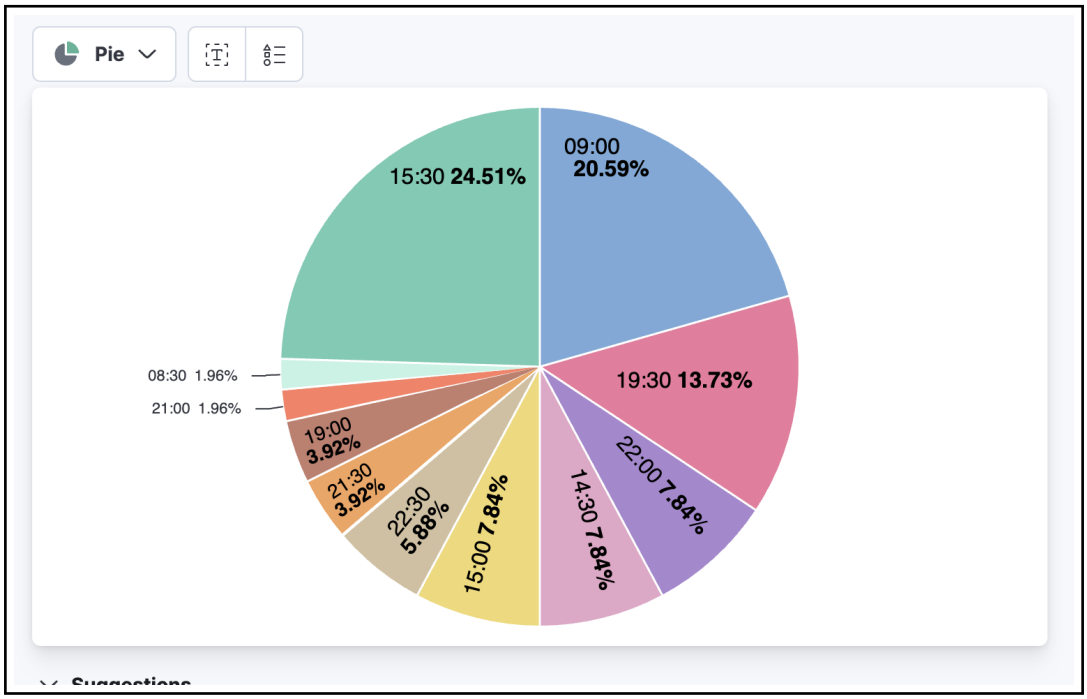


Figure 25: Pie structure

Problems Faced

1. Faced issues connecting backend and frontend containers, solved by placing them in a shared network. Also addressed problems with MongoDB connectivity by including it in the same network.

2. Encountered testing challenges when integrating MongoDB Atlas for testing and deploying

using a MongoDB container.Resolved this by utilizing distinct ports and introducing a variable that adapts based on whether the application is in test mode or deployed.

3. Encountered challenges with tool paths due to varied installations, some through Homebrew and others via pip. Resolved by carefully inspecting and managing the diverse installation paths.

Conclusion

We were successfully able to create a web application for PennyPilot, using MERN Stack and DevOps tools. It facilitates to keep track of savings and expenditure for multiple users.

References

[1] https://fullstackopen.com/
[2] https://github.com/john-smilga/node-express-course
[3] https://mongoosejs.com/docs/api.htm
[4] https://docs.ansible.com/
[5] https://blog.ktz.me/secret-management-with-docker-compose-and-ansible/
[6] https://manglekuo.medium.com/running-mongodb-on-docker-with-macos-b08324f5aab2
[7] https://stackoverflow.com/