AutoInvento simplify the complexities of inventory control within auto repair shops, it involves creating a user-friendly interface for mechanics to request inventory, enabling the owner to review and take action on requests, and providing a mechanism for mechanics to view the status of their requests among other things.

Members of the Team

Bonaventure Anselm (Backend)

Aina Racheal Damilola (Frontend)

Project Description - General Workflow

- Users (mechanics and the owner) have dashboards for viewing
- Inventory items data
- Mechanics data
- Suppliers data

General Workflow Cont’d

The owner who has superuser privileges can

- Add Mechanics or other superusers;
- Inventory item;
- Suppliers;
- and so on.

General Workflow Cont’d

- Mechanics can submit inventory request to the owner
- The owner is notified of the pending request via email
- He can approve or reject requests
- The mechanic can view request statuses from his dashboard
- If request is approved, that quantity is deducted from the inventory table

General Workflow Cont’d

The owner has bar charts in his dashboard for visualizing

- ❖ Usage data
- ❖ Supply data

He can also see inventory status when viewing the inventory list as

- ❖ Low-stock items are shown in orange colour
- ❖ Out-of-stock items are shown in red colour, and
- ❖ Others have default page colour

App Architecture

Frontend:

- We used html, CSS and Javascript to build the frontend, enabling easy user interaction with the app.

Backend:

- We used flask, a python web framework to implement the server-side logic which handles requests, interacts with the database and sends response to the frontend

App Architecture Cont’d

Database:

- We are using Sqlite to store our data currently but will use either MySql or PostgreSql for the same purpose in production
- We used SqlAlchemy, a python SQL toolkit for interaction with the database

ORM (Object-Relational Mapping):

- With SqlAlchemy, interacting with the database using classes and objects instead of raw SQL queries was achieved

App Architecture Cont’d

Authentication and Authorization:

- Mechanics can only use the app when authenticated, and they are created only by the owner who is superuser
- There are also limits to what mechanics can do as against the owner that has superuser privileges

App Architecture Cont’d

Email Service:

- We used Flask-mail to implement email request to notify the owner when mechanics submits requests

Charting Library (Chart.js):

- The owner can visualize inventory item usage and supply data. This was implemented using JS based Chart.js library

Technologies Used

Language:

- Python
- JavaScript
- HTML/CSS

Frameworks:

- Flask
- Bootstrap

Libraries:

- Flask-mail
- Flask-login
- SqlAlchemy
- Flask-Bcrypt
- PyJWT
- Pillow
- Flask-WTF
- ChartJS
- Jinja2 Templates

Development Report - Successes

We were able to build AUTOINVENTO, our auto repair inventory management app project. Our aim is to develop a web application that streamlines the process of managing inventory for an auto repair shop.

The project involved creating a user-friendly interface for mechanics to request inventory, enabling the owner to review and take action on requests, and providing a mechanism for mechanics to view the status of their requests among other things.

Our app can do just that and a bit more

We also achieved the following:

- ❖ Functional User Interface
- ❖ Database Integration
- ❖ Request Notifications
- ❖ Charting Functionality
- ❖ and so on.

Live Demo

Challenges

Request Handling:

- Implementing the logic for approving or rejecting requests posed a challenge. A fully NULL primary key identity error was encountered, leading to some issues in request processing.

Email Service Integration:

- There were difficulties in configuring the email service to notify the owner upon new request submissions. In fact, the email services still fail some times when using the app.

Pagination Implementation:

- Introducing pagination for inventory items presented some challenges. An error occurred when attempting to paginate the results because of how I was calling the paginate fxn

Areas for Improvement

Error Handling:

- The application could benefit from more robust error handling to provide informative feedback to users in case of unexpected events.

Unit Testing:

- Implementing unit tests for various components would enhance the stability and reliability of the application but we have not done it yet

User Experience:

- Further enhancements in the UI/UX design could improve overall usability and user satisfaction.

Lessons Learned

Database Management:

- Working with SQLAlchemy provided valuable experience in managing database interactions efficiently.

Email Integration:

- Overcoming challenges with the email service highlighted the importance of thorough configuration and testing.

Debugging Skills:

- Troubleshooting issues with request processing and pagination improved debugging skills.

Next Steps

In--App Notification:

- We want to implement an in--app notification for requests in owner's dashboard to compliment email notifications.

Automatic inventory reordering:

- This will complete the app, so we are looking forward to implementing it soon

Deployment:

- It'll be good to put the app to real world use cases to understand other challenges we may have missed

Conclusion

We were a bit stretched working on this project, but we are happy considering the exposure and the learning experiences it provided.

We actually learnt a lot from it and will keep expanding the experience

Thank You!