**Blox assignment:**

**Summary:**

During my internship at cubiclespace.pro and through various academic projects, such as the Employee Management System, Gym Website, and Shoppinglyx, I gained extensive experience in web application development using Django and Django REST Framework, demonstrating a comprehensive understanding of backend development, API design, and user experience. These projects involved features like employee management, attendance and time-tracking, exercise routines and complaint management, as well as product management, order tracking, and shopping cart functionality, which I developed through an approach emphasizing collaboration with mentors, understanding features before implementation, and leveraging diverse learning resources, including Stack Overflow, YouTube tutorials, and ChatGPT, to overcome challenges and enhance my knowledge of Django's capabilities.

**Question 1:**

**Internship and Academic Projects**

**1. Employee Management System (Internship at cubiclespace.pro):**

a. **What did the system do?**  
The project was an employee management system built using **Django REST Framework**. It allowed employers to:

* Add and manage employees
* Handle **employee check-in/check-out** functionality
* Manage team assignments and departments
* Implement attendance and time-tracking features
* Provide a **screen tracking system** for employee activity monitoring
* Manage password reset and login functionalities for both employees and employers

The employee side of the system enabled users to:

* Check in/check out of work
* View their attendance and team information
* Reset passwords if forgotten

b. **What other systems have you seen in the wild like that?**  
Similar systems include platforms like **Zoho People**, **Workday**, and **BambooHR**. These platforms offer full employee management features, including attendance tracking, team management, and time-tracking functionalities.

c. **How did you approach the development problem?**  
My approach involved first thoroughly understanding each feature and how it would function within the system. I began by creating a solid foundational structure for the features before progressively improving them. Since my primary role was focused on building APIs and backend functionality, I concentrated on implementing the attendance tracking and department/team management features. I used Django REST Framework to build APIs for each feature, ensuring scalability and modularity. We adopted a **RESTful API** approach to keep things loosely coupled.

d. **What were interesting aspects where you copied code from Stack Overflow?**

~~During my internship, I didn't rely on Stack Overflow for code snippets or solutions for my tasks. Instead, I focused on leveraging the expertise of my colleagues and mentors.~~   
Rather than relying on Stack Overflow for code snippets or task solutions during my internship, I focused on leveraging the expertise and guidance of my colleagues and mentors.

They provided valuable guidance and support, particularly in implementing key features like email verification and password reset functionalities.

For instance, while working on the email verification process, I collaborated with a mentor who shared best practices and helped me understand how to securely handle user data and email communication. This direct interaction and knowledge transfer were more beneficial than simply copying code from online sources, as it allowed me to grasp the underlying concepts and principles.

Overall, this experience reinforced the importance of teamwork and mentorship in the development process, highlighting how collaboration can lead to a deeper understanding of complex features.

e. **What did you learn from some very specific copy-pasting?**

* One key learning was from an implementation of **token-based authentication** in Django REST Framework. The example from Stack Overflow helped me understand how to create secure API endpoints that use tokens for authentication, which is especially important for sensitive actions like check-in/check-out.

**2. Gym Website (Academic Project):**

a. **What did the system do?**  
The system allowed users to register and access exercise routines categorized by muscles. It was built using Django as the backend, where users could sign up, log in, and view a list of exercises based on muscle groups (e.g., chest, arms, legs). Each exercise page displayed detailed descriptions to help users understand how to perform them properly. Additionally, the system included a complaint management feature, enabling users to submit complaints about gym services or facilities, track their status, and receive updates.

b. **What other systems have you seen in the wild like that?**  
Websites like **Bodybuilding.com** and **MyFitnessPal** offer similar functionality by allowing users to browse exercises by body parts. These platforms also provide personalized workout routines, which could be an extension of my project.

c. **How did you approach the development problem?**  
I divided the project into several modules: user registration and login, muscle group pages, a complaint management system, and an admin interface to manage exercises and complaints. The project began with basic functionalities—ensuring users could log in and view exercises. I then integrated the complaint management feature, allowing users to file complaints, track their progress.

d. **What were interesting aspects where you copied code from Stack Overflow?**  
I took inspiration for implementing **user authentication** and **filtering data** from Stack Overflow posts, especially when dealing with Django's QuerySets for filtering exercises based on selected muscle groups.

e. **What did you learn from some very specific copy-pasting?**  
Through specific instances of copy-pasting code from Stack Overflow, I gained a deeper understanding of Django’s authentication mechanisms, particularly how to implement secure password handling and user session management. Additionally, I learned how to retrieve data from the backend, which allowed me to effectively display exercise details and complaint statuses to users.

**3. Shoppinglyx (Academic Project)**

a. **What did the system do?**  
**Shoppinglyx** is an e-commerce platform built using **Django**. The system provides essential features for both admins and users:

* **Admin functionalities:**
  + Add/remove products
  + Manually track and update the status of orders in four steps (e.g., Order Placed, Packed, Shipped, Delivered)
* **User functionalities:**
  + User registration and login
  + View available products and filter by price range and category
  + Add products to a shopping cart where the total amount is automatically calculated
  + Place orders and view order status
  + Add multiple delivery addresses

b. **What other systems have you seen in the wild like that?**  
Similar platforms include **Amazon**, **Flipkart**, and **Shopify**. These platforms offer advanced features like product filters, dynamic pricing, and real-time order tracking, which inspired some aspects of my project like product filtering and order management.

c. **How did you approach the development problem?**  
I began by designing the **user login and product catalog** features. Next, I focused on the shopping cart functionality, where users could add items and see real-time total price updates. The order tracking system was implemented step by step, while the cart’s pricing logic was tied to the product model.

I broke the project into the following phases:

1. **User and Admin management**: Building registration, login, and role-based functionality.
2. **Product Management**: Adding, removing, and displaying products with filters.
3. **Cart and Checkout**: Adding products to the cart and calculating the total amount.
4. **Order Management**: Implementing order tracking and multi-address support.

d. **What were interesting aspects where you copied code from Stack Overflow?**

In my project, I did not rely on Stack Overflow for code snippets or solutions. Instead, I utilized alternative resources such as YouTube tutorials and ChatGPT to solve specific challenges I encountered. Here are a few areas where I found valuable guidance:

* **Cart Calculation Logic:** I explored various YouTube tutorials that demonstrated how to manage shopping cart sessions and dynamically calculate the total price for multiple items. These resources helped me understand how to ensure data persistence across different pages in my application.
* **Order Status Updates:** For managing order statuses in Django models, I utilized examples from ChatGPT to develop a system where the admin could manually update the order state (placed, packed, shipped, delivered). The conversational format allowed me to ask specific questions and receive tailored advice.
* **Product Filters:** When implementing filters for products based on price and category, I referred to YouTube videos that showcased common Django patterns for filtering QuerySets. This hands-on approach helped me grasp the concepts effectively.

By using these resources, I was able to gain a deeper understanding of the features I was working on, which ultimately contributed to the success of my project.

e. **What did you learn from some very specific copy-pasting?**

* For the **shopping cart logic**, I copied an example of how to store cart data in sessions, which helped me understand how to manage data across different views while maintaining the user experience. I learned how to **persist session data** to ensure the cart remains updated even when a user navigates away.
* Implementing **order tracking** taught me how to work with multiple models and integrate them to handle changes in real time. This example showed me how to create clear **model relationships** and improve the admin’s control over the order workflow.

Question 4c: If you were the API designer, what would you do to implement this behavior?

As an API designer implementing rate limiting using a token bucket algorithm, I would start by defining the rate limits, specifying how many API calls a user can make within a certain timeframe, such as 20 requests per minute. I would create a token bucket with a maximum capacity (e.g., 20 tokens) and set a refill rate to allow tokens to replenish over time (e.g., one token every three seconds). When a request is made, the system would check for available tokens; if a token is present, it would be deducted for the API call, while a lack of tokens would result in a "429 Too Many Requests" response. Additionally, I would implement time tracking to monitor the last refill time, ensuring tokens are added at the appropriate rate. To keep users informed, I would include headers in the responses indicating the number of remaining requests (e.g., X-RateLimit-Remaining). This systematic approach would help ensure fair API usage, prevent abuse, and maintain optimal system performance.