Internship Project: QuickDeliver Lite

Welcome to the QuickDeliver Lite project. Over the next six weeks, you'll build a simplified, real-time delivery management system—designed to function like a lightweight logistics platform. The idea is to simulate real-world development, step by step. You'll handle authentication, user roles, delivery tracking, feedback systems, and ultimately polish and deploy your work.

Each problem statement is structured to be completed within one week. The tasks are intentionally layered in a way that demands logical flow, validation, and real data relationships. This is not a "copy and paste from online" type of assignment—each piece will require you to think through the behavior and consequences of your code. The output should reflect practical understanding, not just completion.

You'll be using **Python** as your backend language—either **Flask** or **Django**, based on your comfort level. For storage, begin with **JSON files or SQLite** (you may optionally transition to more complex storage later). Frontend expectations are minimal—simple HTML and Bootstrap are enough, as long as the interface is functional and consistent.

Week 1: Build the Authentication System

Start by building the login and registration system. There are two user roles: **Customer** and **Driver**. Your application must handle both.

What to implement:

- A registration form that collects: name, email, password, and user type (customer or driver).
- Email should be unique. Prevent duplicate registration attempts.
- Passwords must be securely hashed before storage. Avoid plain-text password handling.
- A login form that authenticates users based on stored credentials.
- A logout mechanism that properly clears session data.
- Session handling must persist the user’s role and identity during active use.
- Based on the role, users should be restricted to the relevant parts of the application. For instance, drivers should not have access to customer-only features.

This level is deceptively simple—be careful with how you track and validate sessions. A poorly managed session will break the logic in later levels.

Week 2: Create and View Delivery Requests

Only customers should be able to create delivery requests. Each delivery request includes details about the package and the pickup/dropoff locations.

What to implement:

- A form that allows the logged-in customer to create a delivery request.
    - Required fields: pickup address, dropoff address, and a short note about the package.
    - Automatically assign the delivery a status of `"Pending"`.
    - Associate the request with the currently logged-in customer.
    - Add a timestamp when the delivery is created.
- Create a listing page that shows all delivery requests with status `"Pending"`. This page must be accessible to both customers and drivers.
- Delivery requests must be shown in reverse chronological order—newest first.

Think carefully about how you structure and retrieve your data. A single mistake here (like not tracking the user or forgetting the status) will affect the rest of your features.

Week 3: Allow Drivers to Accept Deliveries

Now it’s time for drivers to engage with the system. They should be able to see pending deliveries and accept any of them—if they act before another driver.

What to implement:

- A page that allows drivers to view all pending delivery requests.
- Each delivery should have an “Accept” button. Once clicked:
    - The status of the delivery should change to `"Accepted"`.
    - The driver’s name should be assigned to that delivery.
- No other driver should be allowed to accept the same delivery after it has been accepted.
- Create a “My Deliveries” page for each driver to view the list of deliveries they have accepted.

This level introduces shared access to data. You will need to build logic to prevent race conditions—multiple drivers must not be able to accept the same delivery.

Week 4: Enable Delivery Status Updates

This week, you'll simulate a real delivery in progress. Drivers should be able to update the status of their accepted deliveries as they move forward.

What to implement:

- Define the following allowed status flow: `"Accepted"` → `"In Transit"` → `"Delivered"`.
- Only the driver assigned to the delivery should be able to make these updates.
- Customers should be able to view the current status of their delivery at any time.
- Only the next valid status should be available. For example, if the current status is `"Accepted"`, only `"In Transit"` should be allowed next.

You must enforce strict control over who can update a delivery and what transitions are valid. This logic will be reused later in feedback and tracking.

Week 5: Add Feedback and Profile Pages

Once deliveries are completed, customers should be able to leave feedback for the driver. At the same time, all users should be able to view their history.

What to implement:

- Once a delivery is marked as `"Delivered"`, allow the customer to submit:
  - A rating from 1 to 5.
  - A short optional comment (limit: 200 characters).
- Create a profile page for each user:
  - Customers should see the list of deliveries they created, along with status and feedback options.
  - Drivers should see the list of deliveries they completed, and the ratings they have received.
- Ensure a customer cannot submit feedback more than once for the same delivery.
- Ensure users cannot access or rate deliveries that they were not involved in.

This level deals with permissions and conditional logic. Incorrect access control will expose user data and invalidate your system's integrity.

Week 6: Polish the Interface and Finalize the Project

This week, your focus is on refining the user experience, fixing inconsistencies, and submitting the project in a clean and professional format.

What to implement:

- Create a consistent navigation system with links to: login, logout, dashboard, delivery list, and profile.
- Use Bootstrap or minimal custom styling to ensure forms and tables are readable.
- Use flash messages or simple alert banners to provide user feedback after actions (e.g., “Delivery accepted”, “Logged out successfully”).
- Clean up the folder structure—separate templates, static files, and routes/controllers.
- Prepare a `README.md` file containing:
  - Setup instructions
  - Project description
  - Key features
  - Known bugs or limitations
- Optional (but appreciated):
  - Deploy your project on PythonAnywhere, Render, or similar.
  - Include screenshots of each page in your GitHub repository.

By now, you should have a complete end-to-end application. Aim for clarity, usability, and structure—these matter just as much as working code.

Weekly Submission Format

You are expected to submit the following at the end of each week:

- A short written summary of what you’ve implemented
- A GitHub repository link (or zipped project folder)
- Any issues you faced or bugs you couldn’t resolve
- Screenshots or a short walkthrough video (optional but encouraged)