Code Review

Functionality Reviewed

- **Signup Functionality** (/signup route in Flask application).

---

1. Code Review Plan

Checklist:

1. **Functionality**:
    - Does the code handle both GET and POST requests correctly?
    - Are all user roles (Customer, Salesperson, Admin) handled properly?
2. **Security**:
    - Are SQL queries parameterized to prevent SQL injection?
    - Is user input validated for missing or invalid data?
    - Is password hashing implemented?
3. **Error Handling**:
    - Does the code handle database connection errors gracefully?
    - Are invalid roles handled with appropriate feedback?
4. **Code Quality**:
    - Are there redundant or unnecessary lines of code?
    - Is the code modular and reusable?
5. **Post-Signup Behavior**:
    - Does the user get redirected appropriately after successful signup?

Code Review

2. Findings

Review Criteria	Observation	Recommendation
GET and POST Handling	The functionality supports both `GET` and `POST` methods correctly.	No changes needed.
Role Handling	The code handles roles correctly by routing data to the appropriate database table based on the user role.	Add a default case to handle unexpected roles explicitly with a custom error message.
SQL Queries	Queries use parameterized placeholders (`?`) to prevent SQL injection.	Consider using hashed passwords instead of storing plaintext passwords.
Validation	No validation is done to check if the email already exists in the database.	Add a query to check if the email is unique before inserting a new record.
Error Handling	Errors are handled using a `try...except` block, but the error message is directly returned to the user, exposing database-related information.	Log the error internally and return a user-friendly message like '**Something went wrong. Try again later.**'
Code Quality	The code is clean, but the SQL query logic for roles is repeated.	Refactor the code to make it modular by moving the role-based logic to a helper function.
Redirection	After successful signup, the user is redirected to the homepage (`/`).	No changes needed.

Code Review

3. Updates to Code

Before Review:

```
if role == 'Customer':
   query = "INSERT INTO Customer (Name, Email, Password) VALUES (?, ?, ?)"
elif role == 'Salesperson':
   query = "INSERT INTO SalesPerson (Name, Email, Password) VALUES (?, ?, ?)"
elif role == 'Admin':
   query = "INSERT INTO Admin (Name, Email, Password) VALUES (?, ?, ?)"
else:
   return "Invalid role selected"
```

After Review (Refactor the role handling logic to reduce redundancy:):

```
role_table_mapping = {
   'Customer': 'Customer',
   'Salesperson': 'SalesPerson',
   'Admin': 'Admin'
}
if role in role_table_mapping:
   query = f"INSERT INTO {role_table_mapping[role]} (Name, Email, Password) VALUES (?, ?, ?)"
else:
   return "Invalid role selected"
```

Before Review (No Email Uniqueness Check):

```
cursor.execute(query, (name, email, password))
```

After Review (With Email Uniqueness Check):

```
check_email_query = f"SELECT COUNT(*) FROM {role_table_mapping[role]} WHERE Email = ?"
cursor.execute(check_email_query, (email,))
if cursor.fetchone()[0] > 0:
   return "Email already exists. Please use a different email."

cursor.execute(query, (name, email, password))
```

Code Review

Before Review (Error Messages Expose Details):

```
except Exception as e:
   return f"Error: {e}"
```

After Review (Friendly Error Messages):

```
except Exception as e:
   print(f"Error during signup: {e}")
   return "An error occurred. Please try again later."
```

4. Results

- **Issues Resolved**:
    - Prevents duplicate email signups.
    - Improved security with proper error handling.
    - Simplified and modular role handling.
- **Remaining Concerns**:
    - Passwords are stored as plaintext (implement hashing in future iterations).