**Name: Ubaid Ur Rehman**

**Task 1: Database Schema Ticket for the "Users" Table**

**Ticket Title:** Database Schema Creation for "Users" Table

**Description:**  
We need to create a database schema for the "Users" table, which will store staff details. This table will capture information from the user profile section and interface. The structure should be designed to support the following fields and their respective data types.

**Schema Details:**

* **user\_id (Primary Key):** Unique identifier for each user (UUID or INT).
* **first\_name:** User's first name (VARCHAR).
* **last\_name:** User's last name (VARCHAR).
* **email:** User's email address (VARCHAR).
* **password:** Encrypted password (VARCHAR).
* **creation\_type\_id:** Selected content type from the interface (Foreign Key referencing Content Types table).
* **slack\_member\_id:** User's Slack member ID (VARCHAR).
* **telegram\_username:** User's Telegram username (VARCHAR).
* **ip\_restriction\_enabled:** Boolean value indicating if IP restrictions are enabled (BOOLEAN).
* **created\_at:** Timestamp of when the record was created (DATETIME).
* **updated\_at:** Timestamp of the last update to the record (DATETIME).

**Acceptance Criteria:**

* Schema supports all the specified fields.
* Appropriate data types are assigned to each field.
* Necessary constraints (e.g., NOT NULL, UNIQUE) are applied where applicable.
* creation\_type\_id links to a Content Types table.

**Task 2: API Endpoint Ticket for User Update Interface (Table 2)**

**Ticket Title:** API Endpoint for User Update in "Users" Table

**Description:**  
Create an API endpoint for updating user information in the "Users" table. The endpoint should allow staff to modify user details based on their input in the user profile section.

**Endpoint Details:**

* **HTTP Method:** PUT
* **URL:** /api/users/{user\_id}
* **Parameters:**
  + user\_id (path parameter): ID of the user to be updated.
* **Request Body (JSON):**
  + first\_name: Updated first name.
  + last\_name: Updated last name.
  + email: Updated email.
  + password: New encrypted password (optional).
  + slack\_member\_id: Updated Slack member ID (optional).
  + telegram\_username: Updated Telegram username (optional).
  + ip\_restriction\_enabled: Enable or disable IP restriction (boolean).
* **Response:**
  + 200 OK: On successful update, return the updated user data.
  + 404 Not Found: If the user ID is invalid.
  + 400 Bad Request: If the input data is invalid.

**Acceptance Criteria:**

* API allows updating all relevant fields in the "Users" table.
* Only valid and existing user records are updated.
* Input validation for the data fields (email format, valid usernames, etc.).

**Task 3: Design Tickets for Staff Interfaces**

**Ticket 1: Design Staff List Interface**

**Ticket Title:** Design for Staff List Interface

**Description:**  
Design the interface for displaying a list of staff members. The list should be searchable and paginated to allow easy navigation. This interface will be used by staff to view and manage other staff members.

**Design Requirements:**

* Search functionality for staff based on name, email, or other attributes.
* Pagination control for navigating between pages of results.
* Display of essential information for each staff member (e.g., name, email, role).
* Actions like view, edit, and delete should be available on each list item.
* Responsive design for mobile and desktop.

**Acceptance Criteria:**

* Clear, user-friendly design that supports search and pagination.
* Interface aligns with the project structure outlined in Table 1.

**Task 4: UI Tickets for Staff Interfaces**

**Ticket 1: UI Development for Staff Creation Interface**

**Ticket Title:** UI for Staff Creation Interface

**Description:**  
Develop the user interface for creating a new staff member. The interface should allow the input of all necessary details such as name, email, password, Slack ID, Telegram username, and content type.

**UI Requirements:**

* Form fields for entering user details (first name, last name, email, password, Slack ID, etc.).
* Dropdown or selection for creation\_type\_id.
* Toggle switch for ip\_restriction\_enabled.
* Validation for all required fields.
* Submit and Cancel buttons.
* Responsive layout that works on mobile and desktop devices.

**Acceptance Criteria:**

* The UI is fully functional with form validation.
* Responsive and adheres to the project structure outlined in Table 1.
* Smooth transitions and form submission without errors.

**Task 5: Feature Integration Tickets for Staff Interfaces**

**Ticket 1: Feature Integration for Staff Delete Interface**

**Ticket Title:** Feature Integration for Staff Deletion

**Description:**  
Integrate the staff deletion feature into the system. This functionality allows staff to delete a user record from the database through the interface. The deletion should prompt the user for confirmation before proceeding.

**Integration Requirements:**

* Button or option in the staff list interface to delete a staff member.
* A confirmation prompt before the deletion is finalized.
* Deletion of the staff member from the "Users" table upon confirmation.
* Notification message (e.g., "Staff member successfully deleted") after the deletion.
* Proper error handling if the deletion fails (e.g., user does not exist).

**Acceptance Criteria:**

* Staff deletion functionality works as expected with confirmation prompt.
* User feedback is provided upon successful deletion or failure.
* No residual data is left behind after deletion.

**Task 6: API Automation using Python and Requests**

**Question 1: Function to retrieve access token from Login API**

1. **Inputs:** Email, Password
2. **Outputs:** Token, Time taken
3. **Function:**
   * Send a POST request to the login endpoint of Reqres API.
   * Capture the token from the response and calculate the time taken for the API call.
   * Return the token and the time taken.

python

Copy code

import requests

import time

def get\_access\_token(email, password):

url = "https://reqres.in/api/login"

data = {"email": email, "password": password}

start\_time = time.time()

response = requests.post(url, json=data)

end\_time = time.time()

time\_taken = end\_time - start\_time

if response.status\_code == 200:

token = response.json().get('token')

return token, time\_taken

else:

raise Exception("Failed to get token")

**Question 2: Function to create a user using the Create User API**

1. **Inputs:** Name, Job
2. **Outputs:** Name, Job, ID, Created at, Time taken
3. **Function:**
   * Send a POST request to create a new user.
   * Return the details including the time taken.

python

Copy code

def create\_user(name, job):

url = "https://reqres.in/api/users"

data = {"name": name, "job": job}

start\_time = time.time()

response = requests.post(url, json=data)

end\_time = time.time()

time\_taken = end\_time - start\_time

if response.status\_code == 201:

result = response.json()

return result['name'], result['job'], result['id'], result['createdAt'], time\_taken

else:

raise Exception("Failed to create user")

**Question 3: Function to update a user using the Update User API**

1. **Inputs:** Name, Job
2. **Outputs:** Name, Job, ID, Updated at, Time taken
3. **Function:**
   * Send a PUT request to update an existing user.
   * Return the updated information including the time taken.

def update\_user(user\_id, name, job):

url = f"https://reqres.in/api/users/{user\_id}"

data = {"name": name, "job": job}

start\_time = time.time()

response = requests.put(url, json=data)

end\_time = time.time()

time\_taken = end\_time - start\_time

if response.status\_code == 200:

result = response.json()

return result['name'], result['job'], result['updatedAt'], time\_taken

else:

raise Exception("Failed to update user")

**Question 4: Function to delete a user using the Delete User API**

1. **Inputs:** None
2. **Outputs:** Time taken
3. **Function:**
   * Send a DELETE request to remove a user and return the time taken.

def delete\_user(user\_id):

url = f"https://reqres.in/api/users/{user\_id}"

start\_time = time.time()

response = requests.delete(url)

end\_time = time.time()

time\_taken = end\_time - start\_time

if response.status\_code == 204:

return time\_taken

else:

raise Exception("Failed to delete user")

**Task 7: Automated UI Testing using Selenium**

**Question 1: Automated test to log in to Swag Labs**

1. **Steps:**
   * Navigate to the login page.
   * Enter username and password.
   * Verify successful login by checking the URL or an element on the dashboard.

from selenium import webdriver

from selenium.webdriver.common.by import By

import time

def login\_to\_swag\_labs(username, password):

driver = webdriver.Chrome()

driver.get("https://www.saucedemo.com/")

driver.find\_element(By.ID, "user-name").send\_keys(username)

driver.find\_element(By.ID, "password").send\_keys(password)

driver.find\_element(By.ID, "login-button").click()

time.sleep(3)

# Verify successful login by checking the page title or URL

if "inventory" in driver.current\_url:

print("Login successful")

else:

print("Login failed")

driver.quit()

**Question 2: Automated test to add and remove an item from cart**

1. **Steps:**
   * Log in to the site.
   * Add an item to the cart and verify the cart count.
   * Remove the item from the cart and verify the cart is empty.

def add\_and\_remove\_item(username, password):

driver = webdriver.Chrome()

driver.get("https://www.saucedemo.com/")

# Login steps

driver.find\_element(By.ID, "user-name").send\_keys(username)

driver.find\_element(By.ID, "password").send\_keys(password)

driver.find\_element(By.ID, "login-button").click()

# Add item to cart

driver.find\_element(By.CLASS\_NAME, "btn\_inventory").click()

cart\_count = driver.find\_element(By.CLASS\_NAME, "shopping\_cart\_badge").text

assert cart\_count == "1", "Item not added to cart"

# Remove item from cart

driver.find\_element(By.CLASS\_NAME, "btn\_secondary").click()

time.sleep(2)

assert not driver.find\_elements(By.CLASS\_NAME, "shopping\_cart\_badge"), "Item not removed from cart"

driver.quit()