Task Documentation

This documentation provides an overview and detailed instructions for setting up and running a Login Task that uses Jinja templates and bulma css framework and javascript. The application has two routes: the index route (/) and the dummy route (/dummy), each rendering its respective HTML template.

➢ Install the required packages

```
pip install flask
```

➢ Create templates inside templates folder.

```
├── app.py
├── templates/
│   ├── index.html
│   └── dummy.html
```

➢ Create routes of the task.

```
from flask import Flask, render_template
app = Flask(__name__)


@app.route('/')
def index():
    return render_template('index.html')

@app.route('/dummy')
def dummy():
    return render_template('dummy.html')

if __name__ == "__main__":
    app.run(debug=True)
```

➢ Create layout for the task.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Document</title>
  <link rel="stylesheet" href="/static/css/bulma.css" />
  <link rel="stylesheet" href="/static/css/style.css" />
</head>
<body>
  <!-- Navbar -->
  <header>
    {% include '_navbar.html' %}
  </header>
  <div class="fixed-grid has-6-cols mb-0">
    <div class="grid">
      <!-- SideBar -->
      <div class="sideBar cell is-col-span-1 ">
        {% include '_sidebar.html' %}
      </div>
      <!-- Content -->
      <section class="Content cell is-col-span-5 hero is-fullheight ">
        <div class="hero-body pt-2 pb-0">
          {% block content %}
          {% include '_loginForm.html' %}
          {% endblock content%}
        </div>
      </section>
    </div>
  </div>
</body>
</html>
```

This HTML document represents a layout template for a web page. It uses the Bulma CSS framework for styling, as indicated by the linked CSS files, and it includes custom styles from a file named style.css. The document is structured with a header section that includes a navigation bar via a separate _navbar.html file. The main content is organized into a fixed grid with six columns, where the first column is a sidebar included from _sidebar.html, and the remaining five columns form the primary content area. This content area is defined as a full-height section styled as a "hero" component in Bulma, with padding adjustments. Within this main content section, a block named content is defined, which is designed to be overridden in different templates, and by default, it includes a _loginForm.html file. This structure allows for consistent layout across different pages, with flexible content management.

➢ Create login form.

```
<form class="mt-3 has-background-white">
                <div class="field ">
                    <label class="label has-text-black">Email</label>
                    <div class="control">
                        <input class="input is-primary has-background-white has-
text-black emailInput" name="email"
                            type="email" placeholder="Enter email" required>
                        <p class="help is-danger emailError"></p>

                    </div>
                </div>
                <div class="field">
                    <label class="label has-text-black">Password</label>
                    <div class="control">
                        <input class="input is-primary has-background-white has-
text-black passwordInput"
                            name="password" type="password" placeholder="Enter
password" required>
                    </div>
                </div>
                <div class="notification is-danger is-light error is-hidden">

                </div>
                <div class="notification is-success is-light success is-hidden">
                    Login Success
                </div>
                <div class="field">
                    <div class="control">
                        <label class="checkbox has-text-black">
                            <input type="checkbox">
                            Remember Me
                        </label>
                    </div>
                </div>
                <a href="#" class="has-text-link">Forget password?</a>
                <div>
                    <button type="submit" class="button loginButton is-primary
mt-3">
                        Login
                    </button>
                </div>
            </form>
```

This HTML snippet defines a login form styled using the Bulma CSS framework. The form has a white background (`has-background-white`) and includes fields for email and password input. Each field is wrapped in a `div` with the class `field`, containing a label with black text and an input element styled as a primary input with white background and black text. Validation messages for the email input are displayed in a `p` element with the class `help is-danger emailError`. Below the input fields, there are two notification areas: one for error messages (`notification is-danger is-light error is-hidden`) and one for success messages (`notification is-success is-light success is-hidden`). The form also includes a "Remember Me" checkbox and a "Forget password?" link styled as a link (`has-text-link`). Finally, there is a submit button (`button loginButton is-primary mt-3`) styled as a primary button, which triggers the login action when clicked. The form elements and styles ensure a user-friendly and visually appealing login interface.

➢ Create login logic.

```
let loginButton = document.querySelector('.loginButton');
loginButton.addEventListener('click', function (e) {
  e.preventDefault();
  let email = document.querySelector('.emailInput').value;
  let password = document.querySelector('.passwordInput').value;
  let resError = document.querySelector('.error');
  let resSuccess = document.querySelector('.success');
  loginButton.classList.add('is-loading');
  setTimeout(async () => {
    let data = {
      "email": email,
      "password": password
    }
    try {
      let res = await axios.post('https://2cdhd3vem9.execute-api.us-east-
2.amazonaws.com/dev/auth/login', data)
      resError.classList.remove('is-hidden');
      resSuccess.classList.remove('is-hidden');
      let userData = await res.data.user;
      localStorage.setItem('userData', JSON.stringify(userData))
      window.location.href = 'http://127.0.0.1:5000/dummy'
    } catch (error) {
      resError.innerHTML = error.response.data.message;
      resError.classList.remove('is-hidden');
    }
    loginButton.classList.remove('is-loading');
  }, 5000);
});
```

This JavaScript code snippet handles the login functionality for a form by adding an event listener to the login button (loginButton). When the button is clicked, the default form submission is prevented using e.preventDefault(). The values of the email and password fields are retrieved from the form inputs. Loading animations are added to the login button by adding the is-loading class. After a 5-second delay, an asynchronous function is executed to send a POST request to a specified API endpoint using Axios. The request contains the email and password as payload. If the login is successful, both error and success notification elements are displayed, the user's data from the response is stored in localStorage, and the user is redirected to a new URL. If the login fails, the error message from the response is displayed in the error notification element. Finally, the loading animation is removed from the login button regardless of the outcome. This script ensures a smooth user experience by providing visual feedback and handling authentication logic on the client side.

➢ Create the dummy page.

```
{% extends 'index.html' %}
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    {% block content %}

    <div class="container is-widescreen is-align-self-flex-start">
        <div class="card">
            <header class="card-header">
                <p class="card-header-title">User Data</p>
            </header>
            <div class="card-content p-4">
                <div class="content userDataCard">

                </div>
            </div>
            <footer class="card-footer">
                <button class="button js-modal-trigger editButton card-footer-
item has-text-primary">Edit</button>
            </footer>
        </div>
    </div>

    <!-- id="modal-js" -->
    <div id="modal-js" class="modal">
        <div class="modal-background"></div>
        <div class="modal-card">
            <header class="modal-card-head">
                <p class="modal-card-title">Edit</p>
                <button class="delete" onclick="closeModal()" aria-
label="close"></button>
            </header>

            <footer class="modal-card-foot">
                <div class="buttons">
                    <button class="button is-success" onclick="closeModal()">Are
you sure?</button>
                    <button class="button" onclick="closeModal()">Cancel</button>
                </div>
            </footer>
        </div>
    </div>

    <script src="../static/js/dummy.js"></script>
    {% endblock content %}
</body>

</html>
```

This HTML template extends from a base template named `index.html` and defines a dummy page for displaying and editing user data. The main content block (`{% block content %}`) features a container that holds a card component, styled with Bulma CSS classes. The card includes a header with the title "User Data", a content section where user data will be displayed, and a footer with an "Edit" button that triggers a modal. The modal (`modal-js`) has a dark background and a card layout for editing purposes, containing a header with a close button, and a footer with confirmation and cancellation buttons. The modal provides a user interface for editing actions, ensuring the user can confirm or cancel changes before proceeding. Additionally, a JavaScript file (`dummy.js`) is included to manage the interactive behaviors, such as opening and closing the modal. This template ensures a structured and responsive layout for user interaction with their data.

➢ Create logic of dummy page.

```
let editButton = document.querySelector('.editButton')
let userData = localStorage.getItem('userData')
let userCard = document.querySelector('.userDataCard')


userCard.innerHTML = `<p>country id : ${JSON.parse(userData).country_id}</p>
<p>entity id : ${JSON.parse(userData).entity_id}</p>
<p>entity name : ${JSON.parse(userData).entity_name}</p>
<p>entity type : ${JSON.parse(userData).entity_type}</p>
<p>organization level abbreviation :
${JSON.parse(userData).organization_level_abbreviation}</p>
<p>organization level id : ${JSON.parse(userData).organization_level_id}</p>
<p>shipping : ${JSON.parse(userData).permissions.Shipping.map(elem => elem)}</p>
<p>users management : ${JSON.parse(userData).permissions['Users
Management'].map(elem => elem)}</p>
<p>region id : ${JSON.parse(userData).region_id}</p>
<p>user email : ${JSON.parse(userData).user_email}</p>
<p>user id : ${JSON.parse(userData).user_id}</p>
<p>user name : ${JSON.parse(userData).user_name}</p>
<p>user phone: ${JSON.parse(userData).user_phone}</p>
<p>user role: ${JSON.parse(userData).user_role}</p>
`


editButton.addEventListener('click', () => {
    document.querySelector('#modal-js').classList.add('is-active');
})

function closeModal() {
    const modal = document.getElementById('modal-js');
    modal.classList.remove('is-active');
}
```

This JavaScript code manages the dynamic content and interactivity for the dummy page that displays user data. It starts by selecting the "Edit" button and retrieving user data stored in localStorage under the key userData. The user data is parsed from JSON format and injected into the userCard element as HTML content, presenting various user details such as country ID, entity ID, entity name, permissions, and contact information. Each piece of user information is extracted from the parsed object and formatted into individual paragraphs.

The script also adds an event listener to the "Edit" button, which activates a modal when clicked. This is done by adding the is-active class to the modal element, making it visible. Additionally, the closeModal function is defined to close the modal by removing the is-active class when invoked. This function is called by the close button and other action buttons within the modal, allowing users to either confirm or cancel their actions. The overall logic ensures that user data is dynamically displayed and the edit functionality is handled smoothly through a modal interface.

➢ UI

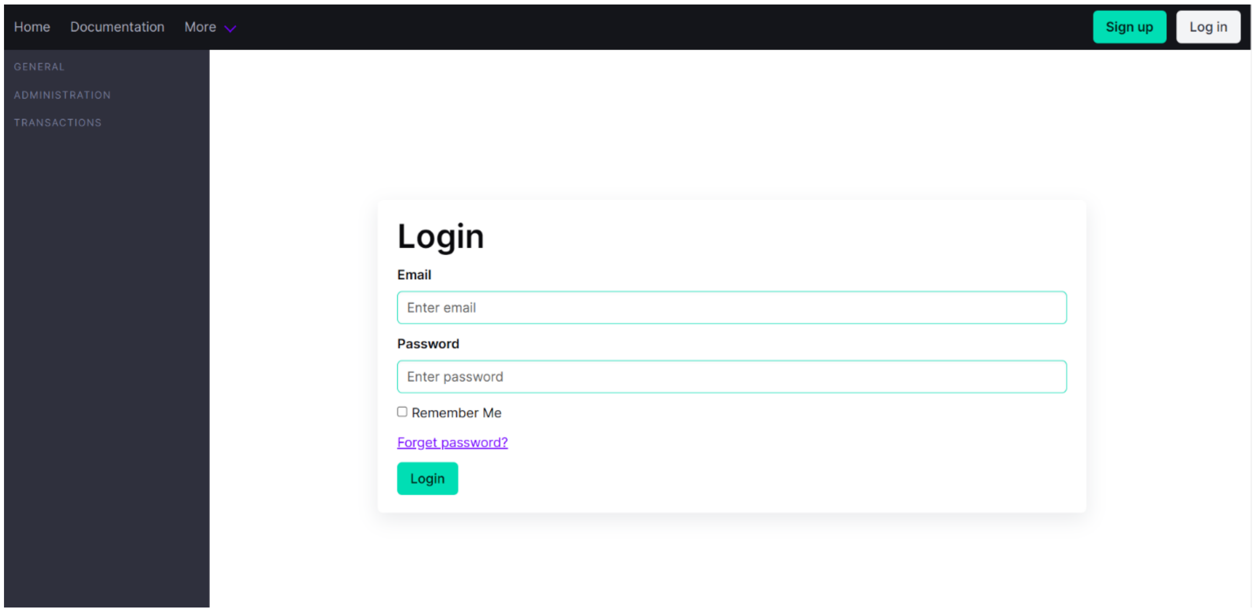
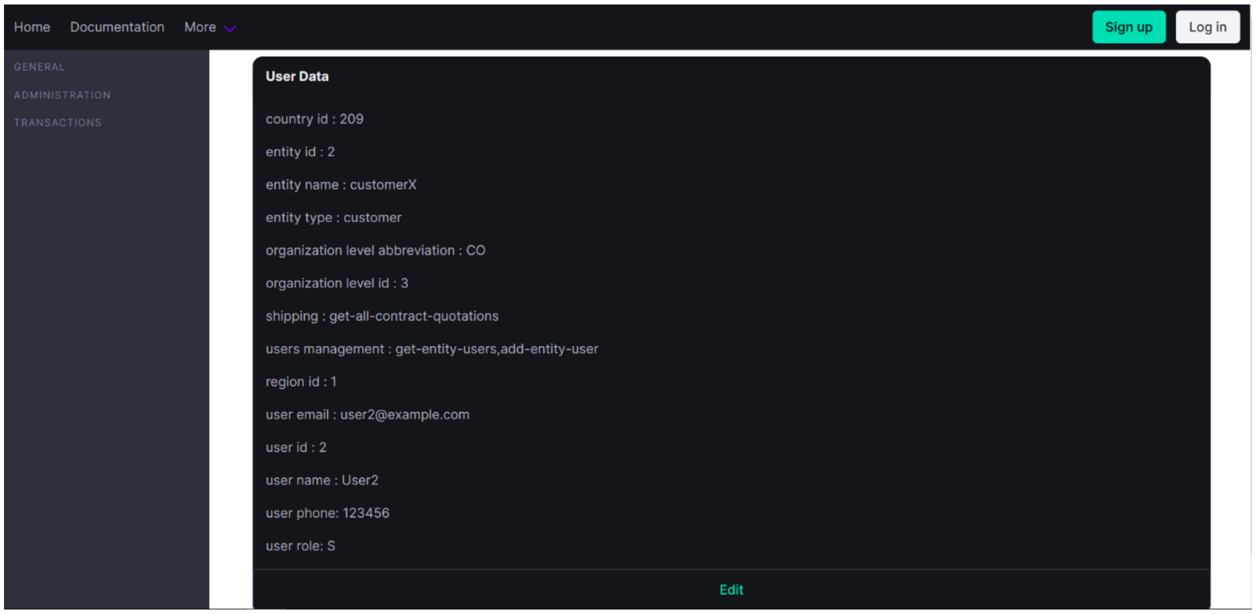


Figure 1: Login Form



User Data

country id : 209

entity id : 2

entity name : customerX

entity type : customer

organization level abbreviation : CO

organization level id : 3

shipping : get-all-contract-quotations

users management : get-entity-users,add-entity-user

region id : 1

user email : user2@example.com

user id : 2

user name : User2

user phone: 123456

user role: S

Figure 2: Dummy page

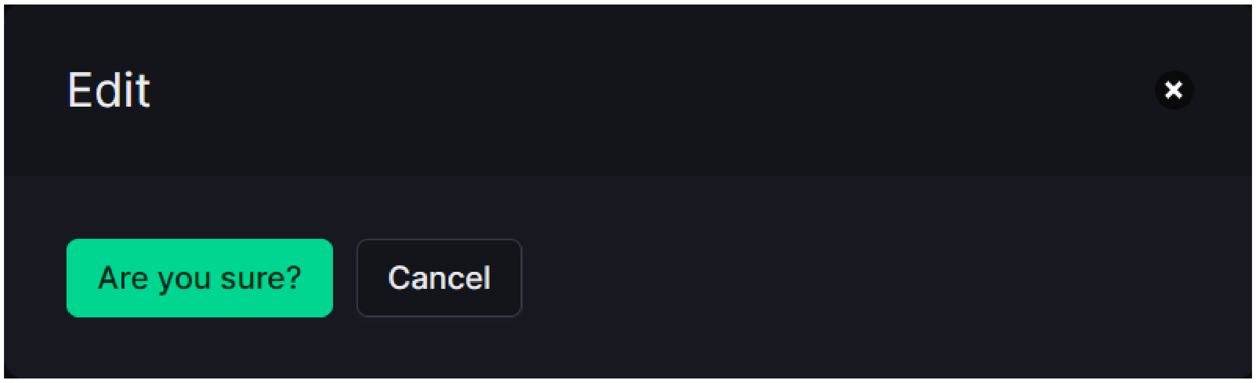


Figure 3: Modal