1. Find the WebID profile document and display the necessary attributes

Step 1:

Set Up a Python Virtual Environment First, make sure you have Python installed on your system. Then, follow these steps to set up a virtual environment:

Create a virtual environment (you might need to install the venv module if it's not already installed):

python -m venv venv

venv\Scripts\activate

Step 2:

Install FastAPI and Uvicorn

pip install fastapi[“all”]

Step 3:

Create a FastAPI Application with file name as `main.py` and write the code below.

from fastapi import FastAPI

app = FastAPI()

# Sample WebID profile data

webid\_profile = [

{

"id": 1,

"name": "John",

"email": "john@example.com",

"location": "New York"

},

{

"id": 2,

"name": "Doe",

"email": "doe@example.com",

"location": "New York"

}

]

@app.get("/profile")

async def get\_webid\_profile():

return webid\_profile

@app.get("/profile/{webid}")

async def get\_webid\_profile(webid: str):

for profile in webid\_profile:

if profile["id"] == int(webid):

return [profile]

Step 4:

Run the FastAPI Application

uvicorn main:app --reload

2. Set and access the primary authentications with account recovery mechanisms

Step 1:

Set Up a Python Virtual Environment First, make sure you have Python installed on your system. Then, follow these steps to set up a virtual environment:

Create a virtual environment (you might need to install the venv module if it's not already installed):

python -m venv venv

venv\Scripts\activate

Step 2:

Install FastAPI and Uvicorn

pip install fastapi[“all”]

Step 3:

Create a FastAPI Application with file name as `main.py` and write the code below.

from fastapi import FastAPI, HTTPException

from fastapi.responses import HTMLResponse

from pydantic import BaseModel

import random

import smtplib

from email.mime.text import MIMEText

app = FastAPI()

# Sample user data as a list (replace with a database in a real application)

users\_db = []

# Temporary storage for password reset tokens (replace with a database in a real application)

password\_reset\_tokens = {}

# Model for user registration

class User(BaseModel):

email: str

password: str

# Model for requesting a password reset

class PasswordResetRequest(BaseModel):

email: str

# Model for password reset

class PasswordReset(BaseModel):

email: str

token: str

new\_password: str

# Route for user registration

@app.post("/api/signup", response\_model=dict)

def register\_user(user: User):

# Check if the email already exists

if any(existing\_user.email == user.email for existing\_user in users\_db):

raise HTTPException(status\_code=400, detail="Email already exists")

users\_db.append(user)

return {"message": "User registered successfully"}

# Route for user login

@app.post("/api/signin", response\_model=dict)

def login\_user(user: User):

# Check if the email and password match

if any(existing\_user.email == user.email and existing\_user.password == user.password for existing\_user in users\_db):

return {"message": "User login successful"}

else:

raise HTTPException(status\_code=400, detail="Invalid credentials")

# Route to serve the HTML form for signup

@app.get("/signup", response\_class=HTMLResponse)

async def signup\_form():

with open("./templates/signup.html", "r") as html:

return HTMLResponse(content=html.read())

# Route to serve the HTML form for signup

@app.get("/signin", response\_class=HTMLResponse)

async def signup\_form():

with open("./templates/signin.html", "r") as html:

return HTMLResponse(content=html.read())

# Password Reset Routes

@app.post("/api/password-reset/request", response\_model=dict)

def request\_password\_reset(request: PasswordResetRequest):

user\_email = request.email

# Check if the email exists in the user database

if any(existing\_user.email == user\_email for existing\_user in users\_db):

# Generate a password reset token (for simplicity, we're using random here)

reset\_token = str(random.randint(1000, 9999))

# Store the token in temporary storage (replace with a database)

password\_reset\_tokens[user\_email] = reset\_token

# Send a password reset email (you should replace this with your email sending logic)

send\_password\_reset\_email(user\_email, reset\_token)

return {"message": "Password reset email sent successfully"}

else:

raise HTTPException(status\_code=400, detail="Email not found")

def send\_password\_reset\_email(email, token):

# Replace this with your actual email sending logic

# Example using smtplib (you may need to configure your email server settings)

server = smtplib.SMTP("smtp.outlook.com", 587)

server.starttls()

server.login("your\_username", "your\_password")

message = f"Click this link to reset your password: http://localhost:8000/reset?email={email}&token={token}"

msg = MIMEText(message)

server.sendmail(email, email, msg.as\_string())

server.quit()

@app.post("/api/password-reset/reset", response\_model=dict)

def reset\_password(reset\_data: PasswordReset):

user\_email = reset\_data.email

token = reset\_data.token

new\_password = reset\_data.new\_password

# Check if the token matches the one in temporary storage

if user\_email in password\_reset\_tokens and password\_reset\_tokens[user\_email] == token:

# Reset the user's password (you should update this to store in your database)

for user in users\_db:

if user.email == user\_email:

user.password = new\_password

break

# Clear the token from temporary storage

del password\_reset\_tokens[user\_email]

return {"message": "Password reset successful"}

else:

raise HTTPException(status\_code=400, detail="Invalid token")

# Route to serve the HTML form for password reset

@app.get("/password-reset", response\_class=HTMLResponse)

async def password\_reset\_form():

with open("./templates/passwordreset.html", "r") as html:

return HTMLResponse(content=html.read())

Step 4:

Create templates folder for web page render

Signin.html

<!DOCTYPE html>

<html>

<head>

<title>User singin</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

}

.container {

max-width: 500px;

margin: 0 auto;

padding: 20px;

background-color: #fff;

border-radius: 5px;

box-shadow: 0 2px 5px #ccc;

}

h2 {

text-align: center;

color: #333;

}

.form-group {

margin: 10px 0;

}

label {

font-weight: bold;

}

input[type="email"],

input[type="password"] {

width: 100%;

padding: 10px;

margin-top: 5px;

margin-bottom: 20px;

border: 1px solid #ccc;

border-radius: 3px;

}

.btn {

background-color: #333;

color: #fff;

padding: 10px 15px;

border: none;

border-radius: 3px;

cursor: pointer;

}

</style>

</head>

<body>

<div class="container">

<h2>User Login</h2>

<form id="signin-form" method="post">

<div class="form-group">

<label for="email">email</label>

<input type="email" id="email" name="email" required />

</div>

<div class="form-group">

<label for="password">Password</label>

<input type="password" id="password" name="password" required />

</div>

<div>

<button class="btn" type="submit">Sing in</button>

<a style="cursor: pointer;" href="/signup">create new account?</a>

<br />

<a style="cursor: pointer;" href="/password-reset">forgot password?</a>

</div>

</form>

</div>

<script>

// Your JavaScript code for user registration

const form = document.getElementById("signin-form");

form.addEventListener("submit", (event) => {

event.preventDefault();

fetch("/api/signin", {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({

email: form.elements.email.value,

password: form.elements.password.value,

}),

})

.then((response) => response.json())

.then((data) => {

alert(data?.message || data?.detail);

})

.catch((error) => console.error(error));

});

</script>

</body>

</html>

Signup.html

<!DOCTYPE html>

<html>

<head>

<title>User Registration</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

}

.container {

max-width: 500px;

margin: 0 auto;

padding: 20px;

background-color: #fff;

border-radius: 5px;

box-shadow: 0 2px 5px #ccc;

}

h2 {

text-align: center;

color: #333;

}

.form-group {

margin: 10px 0;

}

label {

font-weight: bold;

}

input[type="email"],

input[type="password"] {

width: 100%;

padding: 10px;

margin-top: 5px;

margin-bottom: 20px;

border: 1px solid #ccc;

border-radius: 3px;

}

.btn {

background-color: #333;

color: #fff;

padding: 10px 15px;

border: none;

border-radius: 3px;

cursor: pointer;

}

</style>

</head>

<body>

<div class="container">

<h2>User Registration</h2>

<form id="registration-form" method="post">

<div class="form-group">

<label for="email">email</label>

<input type="email" id="email" name="email" required />

</div>

<div class="form-group">

<label for="password">Password</label>

<input type="password" id="password" name="password" required />

</div>

<div>

<button class="btn" type="submit">Register</button>

<a style="cursor: pointer" href="/signin">already have an account?</a>

</div>

</form>

</div>

<script>

// Your JavaScript code for user registration

const form = document.getElementById("registration-form");

form.addEventListener("submit", (event) => {

event.preventDefault();

fetch("/api/signup", {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({

email: form.elements.email.value,

password: form.elements.password.value,

}),

})

.then((response) => response.json())

.then((data) => {

alert(data?.message || data?.detail);

})

.catch((error) => console.error(error));

});

</script>

</body>

</html>

Passwordreset.html

<!DOCTYPE html>

<html>

<head>

<title>Password Reset Request</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

}

.container {

max-width: 500px;

margin: 0 auto;

padding: 20px;

background-color: #fff;

border-radius: 5px;

box-shadow: 0 2px 5px #ccc;

}

h2 {

text-align: center;

color: #333;

}

.form-group {

margin: 10px 0;

}

label {

font-weight: bold;

}

input[type="email"] {

width: 100%;

padding: 10px;

margin-top: 5px;

margin-bottom: 20px;

border: 1px solid #ccc;

border-radius: 3px;

}

.btn {

background-color: #333;

color: #fff;

padding: 10px 15px;

border: none;

border-radius: 3px;

cursor: pointer;

}

</style>

</head>

<body>

<div class="container">

<h2>Password Reset Request</h2>

<form id="reset-request-form">

<div class="form-group">

<label for="email">Email</label>

<input type="email" id="email" name="email" required />

</div>

<button class="btn" type="submit">Request Password Reset</button>

</form>

</div>

<script>

const form = document.getElementById("reset-request-form");

form.addEventListener("submit", (event) => {

event.preventDefault();

fetch("/api/password-reset/request", {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({

email: form.elements.email.value,

}),

})

.then((response) => response.json())

.then((data) => {

alert(data?.message || data?.detail);

})

.catch((error) => console.error(error));

});

// Your JavaScript code for password reset request

</script>

</body>

</html>

Step 5:

Run the FastAPI Application

uvicorn main:app --reload

3. Set and access the secondary authentications with account recovery mechanisms

Step 1:

Set Up a Python Virtual Environment First, make sure you have Python installed on your system. Then, follow these steps to set up a virtual environment:

Create a virtual environment (you might need to install the venv module if it's not already installed):

python -m venv venv

venv\Scripts\activate

Step 2:

Install FastAPI and Uvicorn

pip install fastapi[“all”]

Step 3:

Create a FastAPI Application with file name as `main.py` and write the code below.

from fastapi import FastAPI, HTTPException

from fastapi.responses import HTMLResponse

from pydantic import BaseModel

import random

import smtplib

from email.mime.text import MIMEText

app = FastAPI()

# Sample user data as a list (replace with a database in a real application)

users\_db = []

# Temporary storage for password reset tokens (replace with a database in a real application)

password\_reset\_tokens = {}

# Model for user registration

class User(BaseModel):

email: str

password: str

# Model for requesting a password reset

class PasswordResetRequest(BaseModel):

email: str

# Model for password reset

class PasswordReset(BaseModel):

email: str

token: str

new\_password: str

# Route for user registration

@app.post("/api/signup", response\_model=dict)

def register\_user(user: User):

# Check if the email already exists

if any(existing\_user.email == user.email for existing\_user in users\_db):

raise HTTPException(status\_code=400, detail="Email already exists")

users\_db.append(user)

return {"message": "User registered successfully"}

# Route for user login

@app.post("/api/signin", response\_model=dict)

def login\_user(user: User):

# Check if the email and password match

if any(existing\_user.email == user.email and existing\_user.password == user.password for existing\_user in users\_db):

return {"message": "User login successful"}

else:

raise HTTPException(status\_code=400, detail="Invalid credentials")

# Route to serve the HTML form for signup

@app.get("/signup", response\_class=HTMLResponse)

async def signup\_form():

with open("./templates/signup.html", "r") as html:

return HTMLResponse(content=html.read())

# Route to serve the HTML form for signup

@app.get("/signin", response\_class=HTMLResponse)

async def signup\_form():

with open("./templates/signin.html", "r") as html:

return HTMLResponse(content=html.read())

# Password Reset Routes

@app.post("/api/password-reset/request", response\_model=dict)

def request\_password\_reset(request: PasswordResetRequest):

user\_email = request.email

# Check if the email exists in the user database

if any(existing\_user.email == user\_email for existing\_user in users\_db):

# Generate a password reset token (for simplicity, we're using random here)

reset\_token = str(random.randint(1000, 9999))

# Store the token in temporary storage (replace with a database)

password\_reset\_tokens[user\_email] = reset\_token

# Send a password reset email (you should replace this with your email sending logic)

send\_password\_reset\_email(user\_email, reset\_token)

return {"message": "Password reset email sent successfully"}

else:

raise HTTPException(status\_code=400, detail="Email not found")

def send\_password\_reset\_email(email, token):

# Replace this with your actual email sending logic

# Example using smtplib (you may need to configure your email server settings)

server = smtplib.SMTP("smtp.outlook.com", 587)

server.starttls()

server.login("your\_username", "your\_password")

message = f"Click this link to reset your password: http://localhost:8000/reset?email={email}&token={token}"

msg = MIMEText(message)

server.sendmail(email, email, msg.as\_string())

server.quit()

@app.post("/api/password-reset/reset", response\_model=dict)

def reset\_password(reset\_data: PasswordReset):

user\_email = reset\_data.email

token = reset\_data.token

new\_password = reset\_data.new\_password

# Check if the token matches the one in temporary storage

if user\_email in password\_reset\_tokens and password\_reset\_tokens[user\_email] == token:

# Reset the user's password (you should update this to store in your database)

for user in users\_db:

if user.email == user\_email:

user.password = new\_password

break

# Clear the token from temporary storage

del password\_reset\_tokens[user\_email]

return {"message": "Password reset successful"}

else:

raise HTTPException(status\_code=400, detail="Invalid token")

# Route to serve the HTML form for password reset

@app.get("/password-reset", response\_class=HTMLResponse)

async def password\_reset\_form():

with open("./templates/passwordreset.html", "r") as html:

return HTMLResponse(content=html.read())

Step 4:

Create templates folder for web page render

Signin.html

<!DOCTYPE html>

<html>

<head>

<title>User singin</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

}

.container {

max-width: 500px;

margin: 0 auto;

padding: 20px;

background-color: #fff;

border-radius: 5px;

box-shadow: 0 2px 5px #ccc;

}

h2 {

text-align: center;

color: #333;

}

.form-group {

margin: 10px 0;

}

label {

font-weight: bold;

}

input[type="email"],

input[type="password"] {

width: 100%;

padding: 10px;

margin-top: 5px;

margin-bottom: 20px;

border: 1px solid #ccc;

border-radius: 3px;

}

.btn {

background-color: #333;

color: #fff;

padding: 10px 15px;

border: none;

border-radius: 3px;

cursor: pointer;

}

</style>

</head>

<body>

<div class="container">

<h2>User Login</h2>

<form id="signin-form" method="post">

<div class="form-group">

<label for="email">email</label>

<input type="email" id="email" name="email" required />

</div>

<div class="form-group">

<label for="password">Password</label>

<input type="password" id="password" name="password" required />

</div>

<div>

<button class="btn" type="submit">Sing in</button>

<a style="cursor: pointer;" href="/signup">create new account?</a>

<br />

<a style="cursor: pointer;" href="/password-reset">forgot password?</a>

</div>

</form>

</div>

<script>

// Your JavaScript code for user registration

const form = document.getElementById("signin-form");

form.addEventListener("submit", (event) => {

event.preventDefault();

fetch("/api/signin", {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({

email: form.elements.email.value,

password: form.elements.password.value,

}),

})

.then((response) => response.json())

.then((data) => {

alert(data?.message || data?.detail);

})

.catch((error) => console.error(error));

});

</script>

</body>

</html>

Signup.html

<!DOCTYPE html>

<html>

<head>

<title>User Registration</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

}

.container {

max-width: 500px;

margin: 0 auto;

padding: 20px;

background-color: #fff;

border-radius: 5px;

box-shadow: 0 2px 5px #ccc;

}

h2 {

text-align: center;

color: #333;

}

.form-group {

margin: 10px 0;

}

label {

font-weight: bold;

}

input[type="email"],

input[type="password"] {

width: 100%;

padding: 10px;

margin-top: 5px;

margin-bottom: 20px;

border: 1px solid #ccc;

border-radius: 3px;

}

.btn {

background-color: #333;

color: #fff;

padding: 10px 15px;

border: none;

border-radius: 3px;

cursor: pointer;

}

</style>

</head>

<body>

<div class="container">

<h2>User Registration</h2>

<form id="registration-form" method="post">

<div class="form-group">

<label for="email">email</label>

<input type="email" id="email" name="email" required />

</div>

<div class="form-group">

<label for="password">Password</label>

<input type="password" id="password" name="password" required />

</div>

<div>

<button class="btn" type="submit">Register</button>

<a style="cursor: pointer" href="/signin">already have an account?</a>

</div>

</form>

</div>

<script>

// Your JavaScript code for user registration

const form = document.getElementById("registration-form");

form.addEventListener("submit", (event) => {

event.preventDefault();

fetch("/api/signup", {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({

email: form.elements.email.value,

password: form.elements.password.value,

}),

})

.then((response) => response.json())

.then((data) => {

alert(data?.message || data?.detail);

})

.catch((error) => console.error(error));

});

</script>

</body>

</html>

Passwordreset.html

<!DOCTYPE html>

<html>

<head>

<title>Password Reset Request</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

}

.container {

max-width: 500px;

margin: 0 auto;

padding: 20px;

background-color: #fff;

border-radius: 5px;

box-shadow: 0 2px 5px #ccc;

}

h2 {

text-align: center;

color: #333;

}

.form-group {

margin: 10px 0;

}

label {

font-weight: bold;

}

input[type="email"] {

width: 100%;

padding: 10px;

margin-top: 5px;

margin-bottom: 20px;

border: 1px solid #ccc;

border-radius: 3px;

}

.btn {

background-color: #333;

color: #fff;

padding: 10px 15px;

border: none;

border-radius: 3px;

cursor: pointer;

}

</style>

</head>

<body>

<div class="container">

<h2>Password Reset Request</h2>

<form id="reset-request-form">

<div class="form-group">

<label for="email">Email</label>

<input type="email" id="email" name="email" required />

</div>

<button class="btn" type="submit">Request Password Reset</button>

</form>

</div>

<script>

const form = document.getElementById("reset-request-form");

form.addEventListener("submit", (event) => {

event.preventDefault();

fetch("/api/password-reset/request", {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({

email: form.elements.email.value,

}),

})

.then((response) => response.json())

.then((data) => {

alert(data?.message || data?.detail);

})

.catch((error) => console.error(error));

});

// Your JavaScript code for password reset request

</script>

</body>

</html>

Step 5:

Run the FastAPI Application

uvicorn main:app --reload

4. Design authorization and web access control

Step 1:

Set Up a Python Virtual Environment First, make sure you have Python installed on your system. Then, follow these steps to set up a virtual environment:

Create a virtual environment (you might need to install the venv module if it's not already installed):

python -m venv venv

venv\Scripts\activate

Step 2:

Install FastAPI and Uvicorn

pip install fastapi[“all”]

Step 3:

Create a FastAPI Application with file name as `main.py` and write the code below.

from fastapi import FastAPI, Form, Depends, HTTPException, Cookie

from fastapi.security import OAuth2PasswordBearer

from fastapi.responses import HTMLResponse

from pydantic import BaseModel

app = FastAPI()

# Demo user credentials (you should use a database in a production environment).

demo\_users = [

{"username": "demo", "password": "password",

"token": "vxcmsdfuh34508uglkm34kjvfkjofg9845jnfgb45"},

]

# OAuth2PasswordBearer is a helper class to get the token from the request headers.

def oauth2\_scheme(token):

for u in demo\_users:

if u["token"] == token:

return True

else:

return False

@app.get("/")

async def get\_index():

with open("./templates/index.html", "r") as html:

return HTMLResponse(content=html.read())

# FastAPI routes for login and home pages.

class User(BaseModel):

username: str

password: str

@app.post("/login")

async def login(data: User):

user = next((u for u in demo\_users if u["username"]

== data.username and u["password"] == data.password), None)

print(user)

if user:

return {"success": True, "token": user["token"]}

else:

return {"success": False}

@app.get("/home")

async def home(token: str = Depends(oauth2\_scheme)):

if token:

return HTMLResponse(content=HomeHtml)

else:

return HTTPException(status\_code=401, detail="Not Authorized")

HomeHtml = '''

<!DOCTYPE html>

<html>

<head>

<title>

home page

</title>

</head>

<body>

<h1>

welcome to the home page

</h1>

</body>

</html>

'''

Step 4:

Create templates folder for web page render

index.html

<!DOCTYPE html>

<html>

<head>

<title>Login Page</title>

</head>

<body>

<h1>Login</h1>

<form id="loginForm">

<input type="text" id="username" placeholder="Username" />

<input type="password" id="password" placeholder="Password" />

<button type="submit">Login</button>

</form>

<p id="message"></p>

<script>

document

.getElementById("loginForm")

.addEventListener("submit", function (event) {

event.preventDefault();

const username = document.getElementById("username").value;

const password = document.getElementById("password").value;

// Make an AJAX request to the FastAPI backend for authentication.

fetch("/login", {

method: "POST",

body: JSON.stringify({ username, password }),

headers: {

"Content-Type": "application/json",

},

})

.then((response) => response.json())

.then((data) => {

if (data.success) {

window.location.href = "/home?token=" + data?.token;

} else {

document.getElementById("message").textContent =

"Invalid credentials";

}

});

});

</script>

</body>

</html>

Step 5:

Run the FastAPI Application

uvicorn main:app --reload

5. Find the content representation

Step 1:

Set Up a Python Virtual Environment First, make sure you have Python installed on your system. Then, follow these steps to set up a virtual environment:

Create a virtual environment (you might need to install the venv module if it's not already installed):

python -m venv venv

venv\Scripts\activate

Step 2:

Install FastAPI and Uvicorn

pip install fastapi[“all”]

Step 3:

Create a FastAPI Application with file name as `main.py` and write the code below.

from fastapi import FastAPI

from fastapi.responses import HTMLResponse

app = FastAPI()

@app.get("/", response\_class=HTMLResponse)

async def read\_root():

html\_content = """

<!DOCTYPE html>

<html>

<head>

<title>FastAPI Demo</title>

<style>

body {

background-color: #f0f0f0;

text-align: center;

}

h1 {

color: #0074D9;

}

</style>

</head>

<body>

<h1>Hello, World!</h1>

<p>This is a FastAPI demo.</p>

</body>

</html>

"""

return HTMLResponse(content=html\_content)

Step 4:

Run the FastAPI Application

uvicorn main:app --reload

6. Reading resources from HTTP REST API and WebSockets API

Step 1:

Set Up a Python Virtual Environment First, make sure you have Python installed on your system. Then, follow these steps to set up a virtual environment:

Create a virtual environment (you might need to install the venv module if it's not already installed):

python -m venv venv

venv\Scripts\activate

Step 2:

Install FastAPI and Uvicorn

pip install fastapi[“all”]

Step 3:

Create a FastAPI Application with file name as `main.py` and write the code below.

from fastapi import FastAPI, WebSocket

from fastapi.responses import HTMLResponse

app = FastAPI()

@app.get("/")

async def get\_index():

with open("./templates/index.html", "r") as html:

return HTMLResponse(content=html.read())

@app.get("/api/data")

async def read\_data():

return {"data": "Data from REST API"}

# WebSocket endpoint

@app.websocket("/ws")

async def websocket\_endpoint(websocket: WebSocket):

await websocket.accept()

while True:

data = await websocket.receive\_text()

await websocket.send\_text(f"Data from WebSocket: {data}")

Step 4:

Create templates folder for web page render

index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>FastAPI with WebSocket and REST API</title>

</head>

<body>

<h1>FastAPI with WebSocket and REST API</h1>

<div id="data-container"></div>

<script>

const dataContainer = document.getElementById("data-container");

const ws = new WebSocket("ws://localhost:8000/ws");

ws.onmessage = function (event) {

dataContainer.innerHTML = `<p>${event.data}</p>`;

};

function sendData() {

const inputElement = document.getElementById("data-input");

const data = inputElement.value;

ws.send(data);

}

function getRESTAPI() {

fetch("/api/data", {

method: "GET",

})

.then((response) => response.json())

.then((data) => {

alert(data?.data || data?.detail);

})

.catch((error) => console.error(error));

}

</script>

<input

type="text"

id="data-input"

placeholder="Enter data"

onkeyup="sendData()"

/>

<button onclick="getRESTAPI()">get REST Api</button>

</body>

</html>

Step 5:

Run the FastAPI Application

uvicorn main:app --reload

7. Writing resources from HTTP REST API and WebSockets API

Step 1:

Set Up a Python Virtual Environment First, make sure you have Python installed on your system. Then, follow these steps to set up a virtual environment:

Create a virtual environment (you might need to install the venv module if it's not already installed):

python -m venv venv

venv\Scripts\activate

Step 2:

Install FastAPI and Uvicorn

pip install fastapi[“all”]

Step 3:

Create a FastAPI Application with file name as `main.py` and write the code below.

from fastapi import FastAPI, WebSocket

from fastapi.responses import HTMLResponse

app = FastAPI()

# Simulated REST API data store

data\_store = []

# Simulated WebSocket connections

websocket\_connections = []

# REST API endpoint for writing data

@app.get("/api/data")

async def get\_date():

return {"message": data\_store}

# REST API endpoint for writing data

@app.post("/api/data")

async def write\_data(data: str):

data\_store.append(data)

return {"message": "Data added to REST API"}

# WebSocket endpoint for writing data

@app.websocket("/ws")

async def websocket\_endpoint(websocket: WebSocket):

await websocket.accept()

websocket\_connections.append(websocket)

try:

while True:

data = await websocket.receive\_text()

data\_store.append(data)

await websocket.send\_text(f"Data written to WebSocket: {data}")

except Exception:

websocket\_connections.remove(websocket)

@app.get("/")

async def get\_index():

with open("./templates/index.html", "r") as html:

return HTMLResponse(content=html.read())

Step 4:

Create templates folder for web page render

index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>FastAPI with WebSocket and REST API</title>

</head>

<body>

<h1>FastAPI with WebSocket and REST API</h1>

<div id="data-container">

<h2>Stored Data:</h2>

<ul id="data\_list"></ul>

</div>

<h2>Write Data:</h2>

<input type="text" id="data-input" placeholder="Enter data" />

<button onclick="writeToWebSocket()">Write to WebSocket</button>

<button onclick="writeToAPI()">Write to REST API</button>

<script>

const dataContainer = document.getElementById("data-container");

const ws = new WebSocket("ws://localhost:8000/ws");

const data\_list = document.getElementById("data\_list");

window.onload = () => {

fetch("/api/data")

.then((res) => res.json())

.then((res) => {

res?.message?.map((re) => {

data\_list.innerHTML += `

<li>${re}</li>

`;

});

});

};

ws.onmessage = function (event) {

dataContainer.innerHTML = `<p>${event.data}</p>`;

};

function writeToWebSocket() {

const inputElement = document.getElementById("data-input");

const data = inputElement.value;

ws.send(data);

inputElement.value = "";

}

function writeToAPI() {

const inputElement = document.getElementById("data-input");

const data = inputElement.value;

fetch("/api/data?data=" + data, {

method: "POST",

})

.then((response) => response.json())

.then((data) => {

console.log(data.message);

inputElement.value = "";

});

}

</script>

</body>

</html>

Step 5:

Run the FastAPI Application

uvicorn main:app --reload

8. Data notification using Social Web App protocol

Step 1:

Create index.html file

index.html

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" type="text/css" href="styles.css" />

</head>

<body>

<h1>web notification code</h1>

<button id="showNotificationButton">Show Notification</button>

<script>

document.addEventListener("DOMContentLoaded", () => {

const showNotificationButton = document.getElementById(

"showNotificationButton"

);

if (!("Notification" in window)) {

console.log("This browser does not support system notifications.");

} else {

Notification.requestPermission().then((permission) => {

if (permission === "granted") {

showNotificationButton.addEventListener("click", () => {

showNotification(

"New Message",

"You have a new message from web browser."

);

});

}

});

}

function showNotification(title, message) {

const notification = new Notification(title, {

body: message,

});

notification.onclick = () => {

// Handle the notification click event (e.g., open a related page).

};

}

});

</script>

</body>

</html>

Step 5:

Open the html with browser you will handle the notification button there

9. Managing subscriptions and friends list using Social Web App protocol

Step 1:

Set Up a Python Virtual Environment First, make sure you have Python installed on your system. Then, follow these steps to set up a virtual environment:

Create a virtual environment (you might need to install the venv module if it's not already installed):

python -m venv venv

venv\Scripts\activate

Step 2:

Install FastAPI and Uvicorn

pip install fastapi[“all”]

Step 3:

Create a FastAPI Application with file name as `main.py` and write the code below.

from fastapi import FastAPI

from fastapi.responses import HTMLResponse

from pydantic import BaseModel

from typing import List

app = FastAPI()

# Sample data for subscriptions and friends list

subscriptions = []

friends\_list = []

class SubscriptionCreate(BaseModel):

user\_id: int

content: str

@app.get("/")

def Homepage():

with open("./templates/index.html") as html:

return HTMLResponse(content=html.read())

@app.post("/subscriptions/", response\_model=SubscriptionCreate)

async def create\_subscription(subscription: SubscriptionCreate):

subscriptions.append(subscription)

return subscription

@app.get("/subscriptions/", response\_model=List[SubscriptionCreate])

async def get\_subscriptions():

return subscriptions

Step 4:

Create templates folder for web page render

index.html

<!DOCTYPE html>

<html>

<head>

<title>Subscription and Friends Manager</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 20px;

}

h1 {

text-align: center;

}

h2 {

margin-top: 20px;

}

form {

margin: 10px;

}

button {

padding: 5px 10px;

background-color: #3498db;

color: #fff;

border: none;

cursor: pointer;

}

button:hover {

background-color: #2980b9;

}

</style>

</head>

<body>

<h1>Subscription Manager</h1>

<h2>Subscriptions</h2>

<div id="subscription-list"></div>

<h2>Friends</h2>

<div id="friend-list"></div>

<h2>Add Subscription</h2>

<form id="subscription-form">

<label for="user\_id">User ID:</label>

<input type="number" id="user\_id" name="user\_id" required />

<label for="content">Content:</label>

<input type="text" id="content" name="content" required />

<button type="submit">Add</button>

</form>

<script>

const subscriptionForm = document.getElementById("subscription-form");

const subscriptionList = document.getElementById("subscription-list");

const friendList = document.getElementById("friend-list");

subscriptionForm.addEventListener("submit", async (e) => {

e.preventDefault();

const user\_id = document.getElementById("user\_id").value;

const content = document.getElementById("content").value;

const response = await fetch("/subscriptions/", {

method: "POST",

body: JSON.stringify({ user\_id: user\_id, content: content }),

headers: { "Content-Type": "application/json" },

});

const subscription = await response.json();

addSubscriptionToList(subscription);

});

async function displaySubscriptions() {

const response = await fetch("/subscriptions/");

const subscriptions = await response.json();

subscriptions.forEach(addSubscriptionToList);

}

function addSubscriptionToList(subscription) {

const listItem = document.createElement("div");

listItem.textContent = `User ID: ${subscription.user\_id}, Content: ${subscription.content}`;

subscriptionList.appendChild(listItem);

}

displaySubscriptions();

</script>

</body>

</html>

Step 5:

Run the FastAPI Application

uvicorn main:app --reload

10. Managing list of followers and following list using Social Web App protocol

Step 1:

Set Up a Python Virtual Environment First, make sure you have Python installed on your system. Then, follow these steps to set up a virtual environment:

Create a virtual environment (you might need to install the venv module if it's not already installed):

python -m venv venv

venv\Scripts\activate

Step 2:

Install FastAPI and Uvicorn

pip install fastapi[“all”]

Step 3:

Create a FastAPI Application with file name as `main.py` and write the code below.

from fastapi import FastAPI, HTTPException

from fastapi.responses import HTMLResponse

app = FastAPI()

# Sample user data

users = {

1: {"id": 1, "username": "user1", "followers": [], "following": [2, 3]},

2: {"id": 2, "username": "user2", "followers": [1], "following": [1, 3]},

3: {"id": 3, "username": "user3", "followers": [1, 2], "following": [1, 2]},

4: {"id": 4, "username": "user4", "followers": [], "following": []},

5: {"id": 5, "username": "user5", "followers": [], "following": []},

}

def get\_user(user\_id):

if user\_id in users:

return users[user\_id]

return None

@app.get("/")

def Homepage():

with open("./templates/index.html") as html:

return HTMLResponse(content=html.read())

@app.get("/users")

def UserList():

return users

@app.get("/users/{user\_id}")

def read\_user(user\_id: int):

user = get\_user(user\_id)

if user is None:

raise HTTPException(status\_code=404, detail="User not found")

return user

@app.post("/follow/{user\_id}/{followed\_user\_id}")

def follow\_user(user\_id: int, followed\_user\_id: int):

user = get\_user(user\_id)

followed\_user = get\_user(followed\_user\_id)

if user is None or followed\_user is None:

raise HTTPException(status\_code=404, detail="User not found")

if followed\_user\_id not in user["following"]:

user["following"].append(followed\_user\_id)

followed\_user["followers"].append(user\_id)

return {"message": "Followed successfully", "ok": True}

@app.post("/unfollow/{user\_id}/{followed\_user\_id}")

def unfollow\_user(user\_id: int, followed\_user\_id: int):

user = get\_user(user\_id)

followed\_user = get\_user(followed\_user\_id)

if user is None or followed\_user is None:

raise HTTPException(status\_code=404, detail="User not found")

if followed\_user\_id in user["following"]:

user["following"].remove(followed\_user\_id)

followed\_user["followers"].remove(user\_id)

return {"message": "Unfollowed successfully"}

Step 4:

Create templates folder for web page render

index.html

<!DOCTYPE html>

<html>

<head>

<title>Follow Users</title>

<style>

body {

font-family: Arial, sans-serif;

text-align: center;

background-color: #f2f2f2;

}

h1 {

color: #333;

}

#userList {

max-width: 400px;

margin: 0 auto;

padding: 20px;

background-color: #fff;

border: 1px solid #ccc;

border-radius: 5px;

box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);

display: flex;

flex-direction: column;

gap: 16px;

}

a {

cursor: pointer;

padding: 16px;

font-size: 20px;

color: #333;

text-decoration: none;

}

</style>

</head>

<body>

<h1 id="header">user :</h1>

<div>

<span id="followers">followers : </span>

<br />

<span id="following">following : </span>

</div>

<ul id="userList">

<!-- User data will be displayed here -->

</ul>

<script>

const header = document.getElementById("header");

const userList = document.getElementById("userList");

const followErs = document.getElementById("followers");

const followIng = document.getElementById("following");

let currentUser = "";

window.onload = () => {

currentUser =

window.location.search.slice(1).split("=")[0] == "user" &&

window.location.search.slice(1).split("=")[1];

header.innerHTML += currentUser;

// Initialize the page

fetchUsers();

};

// Fetch and display user data

function fetchUsers() {

fetch("/users")

.then((res) => res.json())

.then((res) => {

let userId = undefined;

Object.keys(res).forEach((i) => {

if (res[i].username == currentUser) {

userId = res[i]?.id;

followErs.innerHTML += res[i].followers.length;

followIng.innerHTML += res[i].following.length;

}

});

Object.keys(res).forEach((i) => {

if (res[i].username != currentUser) {

let followers = res[userId]?.following?.filter((a) => a == i);

userList.innerHTML += `

<a href="/?user=${res[i]?.username}">

<span>${res[i].username}</span>

${

followers?.length > 0

? followers.map(

(a) =>

`<button onclick=UnfollowUser(${userId},${i})>unfollow</button>`

)

: `<button onclick=followUser(${userId},${i})>follow</button>`

}

</a>

`;

}

});

});

}

// Follow a user

async function followUser(userId, followed\_user\_id) {

const response = await fetch(`/follow/${userId}/${followed\_user\_id}`, {

method: "POST",

});

if (response.ok) {

alert("You are now following this user.");

} else {

alert("Failed to follow the user.");

}

window.location.reload();

}

// Follow a user

async function UnfollowUser(userId, followed\_user\_id) {

const response = await fetch(

`/unfollow/${userId}/${followed\_user\_id}`,

{

method: "POST",

}

);

if (response.ok) {

alert("You are now following this user.");

} else {

alert("Failed to follow the user.");

}

window.location.reload();

}

</script>

</body>

</html>

Step 5:

Run the FastAPI Application

uvicorn main:app --reload