# **🎯 מטרות השיעור**

1. להרים שרת Flask בסיסי עם CORS ולחבר אותו ללקוח React
2. ליישם רישום/התחברות עם הצפנת סיסמא (Hash)
3. להנפיק JWT בשרת ולהחזיר ללקוח, כולל שמירה ובדיקת הרשאות

# **חלק 0 – מבנה פרויקט מומלץ**

project/

server/

app.py

models.py

db.py

.env

requirements.txt

client/

(React app via Vite/CRA)

# **חלק 1 – הקמה והרצה (ENV, הפעלה, התקנות, CORS)**

## **1.1 יצירת סביבת עבודה וירטואלית (venv)**

**Windows (PowerShell):**

cd server

python -m venv venv

.\venv\Scripts\Activate.ps1

**macOS / Linux:**

cd server

python3 -m venv venv

source venv/bin/activate

## **1.2 קובץ requirements.txt**

flask==3.0.3

flask-cors==4.0.1

python-dotenv==1.0.1

PyJWT==2.9.0

Werkzeug==3.0.3

התקנה:

pip install -r requirements.txt

## **1.3 קובץ .env**

FLASK\_ENV=development

SECRET\_KEY=super\_secret\_change\_me

JWT\_EXPIRES\_MIN=60

CLIENT\_ORIGIN=http://localhost:5173

## **1.4 שרת בסיסי עם CORS – app.py**

from flask import Flask, request, jsonify

from flask\_cors import CORS

from dotenv import load\_dotenv

import os

load\_dotenv()

app = Flask(\_\_name\_\_)

app.config["SECRET\_KEY"] = os.getenv("SECRET\_KEY")

# CORS – הגבלת מקור הלקוח

CORS(app, resources={r"/\*": {"origins": os.getenv("CLIENT\_ORIGIN")}}, supports\_credentials=True)

@app.get("/health")

def health():

return {"status": "ok"}

if \_\_name\_\_ == "\_\_main\_\_":

app.run(port=5003)

**הרצה:** Windows/macOS/Linux: python app.py  
 בדיקה: גלשו ל־http://localhost:5003/health → אמור להחזיר {"status":"ok"}.

* להבין למה משתמשים ב-Hash (ולא שומרים סיסמא גולמית).
* ליישם רישום/התחברות עם generate\_password\_hash ו־check\_password\_hash.
* לבדוק את הזרימה עם curl/Postman/React קטן (רשות).

# **דרישות מוקדמות**

כבר מותקן אצלך: Flask, flask-cors, Werkzeug.  
 לא צריך ספריות נוספות.

# **שלב 1 – “מסד” זמני בזיכרון**

server/db.py

* USERS = {} # email -> {email, name, password\_hash}

# **שלב 2 – פונקציות משתמשים עם Hash**

server/models.py

from werkzeug.security import generate\_password\_hash, check\_password\_hash

from db import USERS

def create\_user(email: str, name: str, password: str):

email = email.strip().lower()

if not email or not name or not password:

return None, "missing fields"

if email in USERS:

return None, "user exists"

pw\_hash = generate\_password\_hash(password) # יוצר hash עם salt פנימי

USERS[email] = {"email": email, "name": name, "password\_hash": pw\_hash}

return {"email": email, "name": name}, None

def verify\_user(email: str, password: str):

email = email.strip().lower()

u = USERS.get(email)

if not u:

return None, "user not found"

if not check\_password\_hash(u["password\_hash"], password):

return None, "invalid credentials"

return {"email": u["email"], "name": u["name"]}, None

# **שלב 3 – מסלולי Register/Login ב־Flask**

server/app.py

from flask import Flask, request, jsonify

from flask\_cors import CORS

from models import create\_user, verify\_user

app = Flask(\_\_name\_\_)

CORS(app, resources={r"/\*": {"origins": "http://localhost:5173"}})

@app.get("/health")

def health():

return {"status": "ok"}

@app.post("/auth/register")

def register():

data = request.get\_json() or {}

user, err = create\_user(data.get("email",""), data.get("name",""), data.get("password",""))

if err:

return jsonify({"error": err}), 400

return jsonify({"message": "registered", "user": user}), 201

@app.post("/auth/login")

def login():

data = request.get\_json() or {}

user, err = verify\_user(data.get("email",""), data.get("password",""))

if err:

return jsonify({"error": err}), 401

return jsonify({"message": "ok", "user": user}), 200

if \_\_name\_\_ == "\_\_main\_\_":

app.run(port=5003)

# **שלב 4 – בדיקות מהטרמינל**

* # רישום
* curl -X POST http://localhost:5003/auth/register -H "Content-Type: application/json" \

-d '{"email":"a@a.com","name":"A","password":"secret123"}'

# התחברות

curl -X POST http://localhost:5003/auth/login -H "Content-Type: application/json" \

-d '{"email":"a@a.com","password":"secret123"}'

# ניסיון עם סיסמא שגויה

curl -X POST http://localhost:5003/auth/login -H "Content-Type: application/json" \

-d '{"email":"a@a.com","password":"wrong"}'

# **הסבר קצר לתלמיד**

* **Hash ≠ הצפנה**: אי-אפשר “לשחזר” את הסיסמא; משווים hash(candidate) ל־hash שאוחסן.
* generate\_password\_hash יוצר Hash עם **salt** ומחזק בשיטה בטוחה (ברירת מחדל טובה).
* check\_password\_hash מבצע את ההשוואה בצורה בטוחה (כולל תזמונים).

# **🎯 מטרות השיעור**

1. להבין למה משתמשים ב־Token (לעומת Session קלאסי) ומה היתרונות/חסרונות.
2. ליצור **Access Token (JWT)** ב־Flask אחרי התחברות מוצלחת.
3. להגן על ראוטים בעזרת בדיקת טוקן (middleware).
4. צריכת ה־API מהקליינט (React): שמירת טוקן, הוספת Authorization: Bearer, טיפול ב־401.

# **למה בכלל טוקן?**

* **Stateless**: השרת לא צריך לשמור סטייט לכל משתמש (סקיילבילי, קל ל־microservices).
* **נישא ב־Header**: קל להעביר בין שירותים (API Gateway, שירותי צד שלישי).
* **Claims**: אפשר לקודד בתוכו פרטים (זהות/תפקיד/תוקף) – לא סיסמא!
* **חסרונות**: חשוף ל־XSS אם שומרים ב־localStorage; חובה להקפיד על הגנות. לפרודקשן לעתיד—מומלץ **HttpOnly Secure Cookie**.

# **חלק א’ – שרת Python (Flask)**

## **1) התקנות וקונפיגורציה**

הוסיפו לקובץ הדרישות/התקינו:

pip install PyJWT python-dotenv

קובץ .env (אם אין):

SECRET\_KEY=change\_me\_in\_prod

JWT\_EXPIRES\_MIN=60

CLIENT\_ORIGIN=http://localhost:5173

טעינת ENV (אם עדיין לא טוענים):

# למשל בקובץ app/\_\_init\_\_.py או בתחילת הקונטרולר לפני השימוש

from dotenv import load\_dotenv

load\_dotenv()

## **2) עיקרון חשוב – אל תחזירו סיסמא/Hash ללקוח**

בקוד שלך, בפונקציית login\_user—אסור להחזיר את user['password'] בתגובה. נחזיר רק שדות בטוחים + ה־token.

## **3) יצירת טוקן (Service קטן)**

צרו מודול עזר: services/jwt\_service.py

import os, datetime, jwt

SECRET\_KEY = os.getenv("SECRET\_KEY", "dev\_secret")

EXPIRES\_MIN = int(os.getenv("JWT\_EXPIRES\_MIN", "60"))

def create\_access\_token(user\_dict: dict) -> str:

"""

user\_dict מכיל לפחות: user\_id, email, role\_id, first\_name, last\_name

"""

payload = {

"sub": str(user\_dict["user\_id"]),

"email": user\_dict["email"],

"role": user\_dict["role\_id"],

"name": f'{user\_dict.get("first\_name","")} {user\_dict.get("last\_name","")}'.strip(),

"exp": datetime.datetime.utcnow() + datetime.timedelta(minutes=EXPIRES\_MIN),

"iat": datetime.datetime.utcnow(),

"iss": "your-app"

}

return jwt.encode(payload, SECRET\_KEY, algorithm="HS256")

: services/auth\_middleware.py

import os, jwt

from functools import wraps

from flask import request, jsonify, g

SECRET\_KEY = os.getenv("SECRET\_KEY", "dev\_secret")

def require\_auth(f):

@wraps(f)

def wrapper(\*args, \*\*kwargs):

auth = request.headers.get("Authorization", "")

if not auth.startswith("Bearer "):

return jsonify({"error": "Missing Bearer token"}), 401

token = auth.split(" ", 1)[1]

try:

payload = jwt.decode(token, SECRET\_KEY, algorithms=["HS256"])

# שמור זהות ביקשה למהלך הבקשה

g.user = {

"user\_id": payload.get("sub"),

"email": payload.get("email"),

"role\_id": payload.get("role"),

"name": payload.get("name")

}

except jwt.ExpiredSignatureError:

return jsonify({"error": "Token expired"}), 401

except jwt.InvalidTokenError:

return jsonify({"error": "Invalid token"}), 401

return f(\*args, \*\*kwargs)

return wrapper

## **4) עדכון ה־Controller שלך: יצירת טוקן בהתחברות**

להלן גרסה מעודכנת ל־login\_user + תיקון החזרת הפרטים (ללא סיסמא) וסטטוסים:

# controllers/user\_controller.py

import re, os

from flask import jsonify, request

from werkzeug.security import generate\_password\_hash, check\_password\_hash

from models.user\_model import User

from services.jwt\_service import create\_access\_token

class UserController:

@staticmethod

def register\_user(data):

required\_fields = ['first\_name', 'last\_name', 'email', 'password']

if not all(field in data for field in required\_fields):

return jsonify({'error': 'Missing required fields'}), 400

first\_name = data['first\_name'].strip()

last\_name = data['last\_name'].strip()

email = data['email'].strip().lower()

password = data['password'].strip()

role\_id = 2 # Force regular user

if not all([first\_name, last\_name, email, password]):

return jsonify({'error': 'All fields are required'}), 400

email\_pattern = r'^[\w\.-]+@[\w\.-]+\.\w+$'

if not re.match(email\_pattern, email):

return jsonify({'error': 'Invalid email format'}), 400

existing\_user = User.get\_user\_by\_email(email)

if existing\_user:

return jsonify({'error': 'Email already exists'}), 409

hashed\_password = generate\_password\_hash(password)

try:

new\_user = User.create\_user(first\_name, last\_name, email, hashed\_password, role\_id)

# אל תחזיר hashed password

safe\_user = {

'user\_id': new\_user['user\_id'],

'first\_name': new\_user['first\_name'],

'last\_name': new\_user['last\_name'],

'email': new\_user['email'],

'role\_id': new\_user['role\_id']

}

return jsonify({'user': safe\_user}), 201

except Exception as e:

return jsonify({'error': str(e)}), 500

@staticmethod

def login\_user(data):

if not all(k in data for k in ('email', 'password')):

return jsonify({'error': 'Missing email or password'}), 400

email = data['email'].strip().lower()

password = data['password'].strip()

if not email or not password:

return jsonify({'error': 'Email and password are required'}), 400

user = User.get\_user\_by\_email(email)

if not user:

return jsonify({'error': 'User not found'}), 404

if not check\_password\_hash(user['password'], password):

return jsonify({'error': 'Incorrect password'}), 401

# יצירת טוקן אחרי אימות

token = create\_access\_token(user)

safe\_user = {

'user\_id': user['user\_id'],

'first\_name': user['first\_name'],

'last\_name': user['last\_name'],

'email': user['email'],

'role\_id': user['role\_id']

}

# 200 (OK) בלוגאין, לא 201

return jsonify({'user': safe\_user, 'token': token}), 200

## **5) ראוט מוגן לדוגמה**

בקובץ ה־routes (או ה־app) הוסיפו ראוט שמוגן בעזרת הדקורטור:

from flask import Blueprint, jsonify, g

from services.auth\_middleware import require\_auth

protected\_bp = Blueprint('protected', \_\_name\_\_)

@protected\_bp.get("/me")

@require\_auth

def me():

# g.user הוזרק ע"י הדקורטור

return jsonify({"user": g.user}), 200

## **6) בדיקות מהירות (curl)**

# Login – שמור את ה-token מהמענה

curl -X POST http://localhost:5003/auth/login -H "Content-Type: application/json" \

-d '{"email":"a@a.com","password":"secret123"}'

# ראוט מוגן

curl http://localhost:5003/me -H "Authorization: Bearer <TOKEN>"

אני רוצה להטמיע token בפייתון בפרויקט שלי. כלומר שהניתוב יהיה מוגן באמצעות token אני רוצה להחזיר ללקוח שביצע לוגאין toen אני רוצה שתתמיע בקונטרול את ה token בלוגאין ו צריך להתקין pyjwt ו צריך קובץ הכנה env . לפני שאתה יוצר ספר לי מה אתה הולך ליצור בשלבים. לאחר מכן נתמיע. בלי קוד בינתיים

## 

## **מה אני הולך ליצור:**

### **1. התקנת pyjwt**

* נוסיף את pyjwt ל-requirements.txt
* נוודא שהספרייה מותקנת

### **2. קובץ .env**

* ניצור קובץ .env עם משתני סביבה
* נגדיר SECRET\_KEY חזק ליצירת JWT tokens
* נגדיר JWT\_EXPIRATION\_TIME (זמן תפוגה של ה-token)

### **3. עדכון user\_controller.py**

* נוסיף פונקציה ליצירת JWT token בפונקציית login\_user
* נחזיר את ה-token ללקוח לאחר לוגין מוצלח
* נוסיף פונקציה לוולידציה של token

### **4. יצירת decorator להגנה על routes**

* ניצור פונקציה שתבדוק אם ה-token תקין
* נשתמש בה כדי להגן על routes שדורשים authentication

### **5. עדכון routes**

* נגן על routes שדורשים authentication
* נוסיף את ה-token ל-headers של בקשות מוגנות

### **6. עדכון seed\_data.py (אופציונלי)**

* נוודא שיש משתמשים לבדיקה

## **סדר הפעולות:**

1. התקנת pyjwt
2. יצירת .env
3. עדכון user\_controller
4. יצירת decorator להגנה
5. עדכון routes
6. בדיקה ובדיקות

האם זה נשמע הגיוני? האם יש לך העדפות מיוחדות לגבי:

* זמן תפוגה של ה-token?
* איזה routes צריכים להיות מוגנים?
* איך תרצה שהלקוח ישלח את ה-token (headers, cookies, או query params)?

אחרי שתאשר, נתחיל ליישם!

3%

1 Tab

# **חלק ב’ – צד לקוח React**

## **1) שירות API בסיסי**

src/api.js

const API\_URL = "http://localhost:5003";

async function handle(res) {

const data = await res.json().catch(() => ({}));

if (!res.ok) throw new Error(data.error || "Request failed");

return data;

}

export async function login({ email, password }) {

const res = await fetch(`${API\_URL}/auth/login`, {

method: "POST",

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

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

});

return handle(res); // { user, token }

}

export async function getMe(token) {

const res = await fetch(`${API\_URL}/me`, {

headers: { Authorization: `Bearer ${token}` },

});

return handle(res); // { user }

}

## **2) ניהול טוקן ב־React (פשוט לשיעור)**

src/App.jsx

import { useState } from "react";

import { login, getMe } from "./api";

export default function App() {

const [form, setForm] = useState({ email: "", password: "" });

const [token, setToken] = useState(localStorage.getItem("token") || "");

const [me, setMe] = useState(null);

const [error, setError] = useState("");

const onChange = (e) => setForm({ ...form, [e.target.name]: e.target.value });

const doLogin = async () => {

setError("");

try {

const { user, token } = await login(form);

setToken(token);

localStorage.setItem("token", token);

setMe(user);

} catch (e) {

setError(e.message);

}

};

const fetchMe = async () => {

setError("");

try {

const data = await getMe(token);

setMe(data.user);

} catch (e) {

setError(e.message);

if (e.message.toLowerCase().includes("expired") || e.message === "Unauthorized") {

localStorage.removeItem("token");

setToken("");

setMe(null);

}

}

};

const logout = () => {

localStorage.removeItem("token");

setToken("");

setMe(null);

};

return (

<div style={{ maxWidth: 420, margin: "2rem auto", fontFamily: "sans-serif" }}>

<h1>JWT Demo</h1>

<h3>Login</h3>

<input name="email" placeholder="email" onChange={onChange} />

<input name="password" type="password" placeholder="password" onChange={onChange} />

<button onClick={doLogin}>Login</button>

{error && <p style={{color:"crimson"}}>{error}</p>}

<h3>Token</h3>

<textarea value={token} readOnly rows={3} style={{width:"100%"}} />

<h3>Protected</h3>

<button onClick={fetchMe} disabled={!token}>GET /me</button>

<pre>{me ? JSON.stringify(me, null, 2) : "No user loaded"}</pre>

<button onClick={logout} disabled={!token}>Logout</button>

</div>

);

}

לשיעור זה—localStorage בסדר. לפרודקשן: שקלו לעבור ל־**HttpOnly Secure Cookie** בשרת + CORS(supports\_credentials=True) בצד Flask ו־credentials: "include" ב־fetch.

# **דגשי אבטחה חשובים**

* **לעולם לא** מחזירים סיסמא/Hash בתגובת ה־API.
* **תוקף קצר** לטוקן (30–120 דק’).
* טיפול ב־401 בקליינט → יציאה/רענון התחברות.
* **XSS**: אם נשארים עם localStorage להדגמה — לא להזרים HTML לא מאובטח ולשמור על קוד נקי.
* לפרודקשן: Cookie HttpOnly + SameSite=None + Secure, ושקילת Refresh Token.

הסבר פשוט לתלמידים על:

def require\_auth(f):

@wraps(f)

def wrapper(\*args, \*\*kwargs):

...

return f(\*args, \*\*kwargs)

return wrapper

## **מה זה דקורטור? (במילים פשוטות)**

**דקורטור** = פונקציה שמקבלת פונקציה אחרת, ועוטפת אותה בשכבה נוספת של התנהגות — בלי לשנות את הקוד של הפונקציה המקורית.  
 תחשבו על זה כעל “שומר בכניסה”: לפני שנכנסים לפונקציה האמיתית, עוברים דרך שומר שמוודא שיש לכם אישור.

## **פירוק השורות**

1. def require\_auth(f):  
   * require\_auth היא פונקציה שמקבלת את הפונקציה המקורית (f) – למשל ה־route של Flask.
2. @wraps(f)  
   * קישוט קטן ששומר את השם/תיעוד/מטא־דאטה של f.
   * בלי זה, הפונקציה תיקרא “wrapper”, וזה מבלבל דיבאגר/Flask/דוקומנטציה.
3. def wrapper(\*args, \*\*kwargs):  
   * זו הפונקציה **החדשה** שתופעל במקום f.
   * למה \*args, \*\*kwargs? כדי לקבל **כל** פרמטרים שאי פעם f מקבלת (חתימה גמישה), ואז להעביר אותם הלאה.
4. בתוך wrapper  
   * עושים בדיקות “לפני”: למשל לקרוא כותרת Authorization, לפרש JWT, לשים את המשתמש ב־g.user.
   * אם אין הרשאה → מחזירים 401 מיד.
   * אם הכל תקין → קוראים לפונקציה המקורית: return f(\*args, \*\*kwargs).
5. return wrapper  
   * מחזירים את ה”גרסה החדשה” של הפונקציה. מעכשיו, כשפלסק יקרא לפונקציה — הוא בעצם יקרא ל־wrapper שלנו.

## **איך זה נראה בשימוש?**

@app.get("/me")

@require\_auth

def me():

# נרוץ רק אם עברנו את השומר

return jsonify({"msg": "hello, authorized user"})

קריאה ל־/me → קודם wrapper (השומר) → אם אישור תקין, מפעילים את me.

## **דימוי לתלמידים**

* **מתנה + עטיפה**: f היא המתנה, require\_auth מוסיף לה **עטיפה**. פותחים את העטיפה (בדיקות), ואם הכל טוב — מגיעים למתנה.
* **שומר בכניסה**: לפני שאפשר להיכנס למסיבה (הפונקציה), עובר דרך שומר שבודק כרטיס (הטוקן).

שלב 1

שימוש ב token צד קליינט

1. את ה token שמתקבל לאחר login נשמור ב localstorage או sessionStorage
2. נשלח token מעכשיו לאתר והוא יוודא שאנחנו אכן לוגין

אני מקבל token מהשרת. לשמור אותו בפונקצית לוגאין ב api ל localstorage C:\Users\jbt\Desktop\17.8.2025-hashing\react-app\my-app-v3-token\src\api\api.js או שעדיף לשמור ל localstorage ב context C:\Users\jbt\Desktop\17.8.2025-hashing\react-app\my-app-v3-token\src\contexts\Context.jsx?

שלב 2

דוגמא לבניית לשימוש ב token

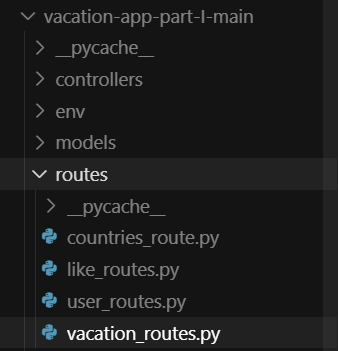
נבנה ארכיטקטורה לבניית vacations

תזכרו ש - ה token שמור ברמת ה context אין צורך לגשת ל localstorage

Api - manage create read update delete vacations

כדי לנהל crud אנו נשלח את ה token עבור הפונקציות בצד השרת שדורשות זאת

ניגש לצד השרת ונביט בקובץ ה route של vaction



אני מקבל token מהשרת. לשמור אותו בפונקצית לוגאין ב api ל localstorage C:\Users\jbt\Desktop\17.8.2025-hashing\react-app\my-app-v3-token\src\api\api.js או שעדיף לשמור ל localstorage ב context C:\Users\jbt\Desktop\17.8.2025-hashing\react-app\my-app-v3-token\src\contexts\Context.jsx?

ניצור קובץ apiVacations באמצעות ה ai

אני מעונין בקובץ apiVacations בריאכט שניגש עם token ל url הבאים :

@vacation\_bp.route('/vacations', *methods*=['POST'])

@require\_auth *# Requires authentication to create vacation*

def create\_vacation():

*return* VacationController.create\_vacation()

@vacation\_bp.route('/vacations', *methods*=['GET'])

@require\_auth *# Requires authentication to get vacation*

def get\_vacations(): *# Public route - no authentication required*

*return* VacationController.get\_vacations()

@vacation\_bp.route('/vacations/<int:vacation\_id>', *methods*=['PUT'])

@require\_auth *# Requires authentication to update vacation*

def update\_vacation(*vacation\_id*):

*return* VacationController.update\_vacation(*vacation\_id*)

@vacation\_bp.route('/vacations/<int:vacation\_id>', *methods*=['DELETE'])

@require\_admin *# Requires admin role to delete vacation*

def delete\_vacation(*vacation\_id*):

*return* VacationController.delete\_vacation(*vacation\_id*)

@vacation\_bp.route('/vacations/<int:vacation\_id>', *methods*=['GET'])

def get\_vacation\_by\_id(*vacation\_id*): *# Public route - no authentication required*

*return* VacationController.get\_vacation\_by\_id(*vacation\_id*)

שלב 3

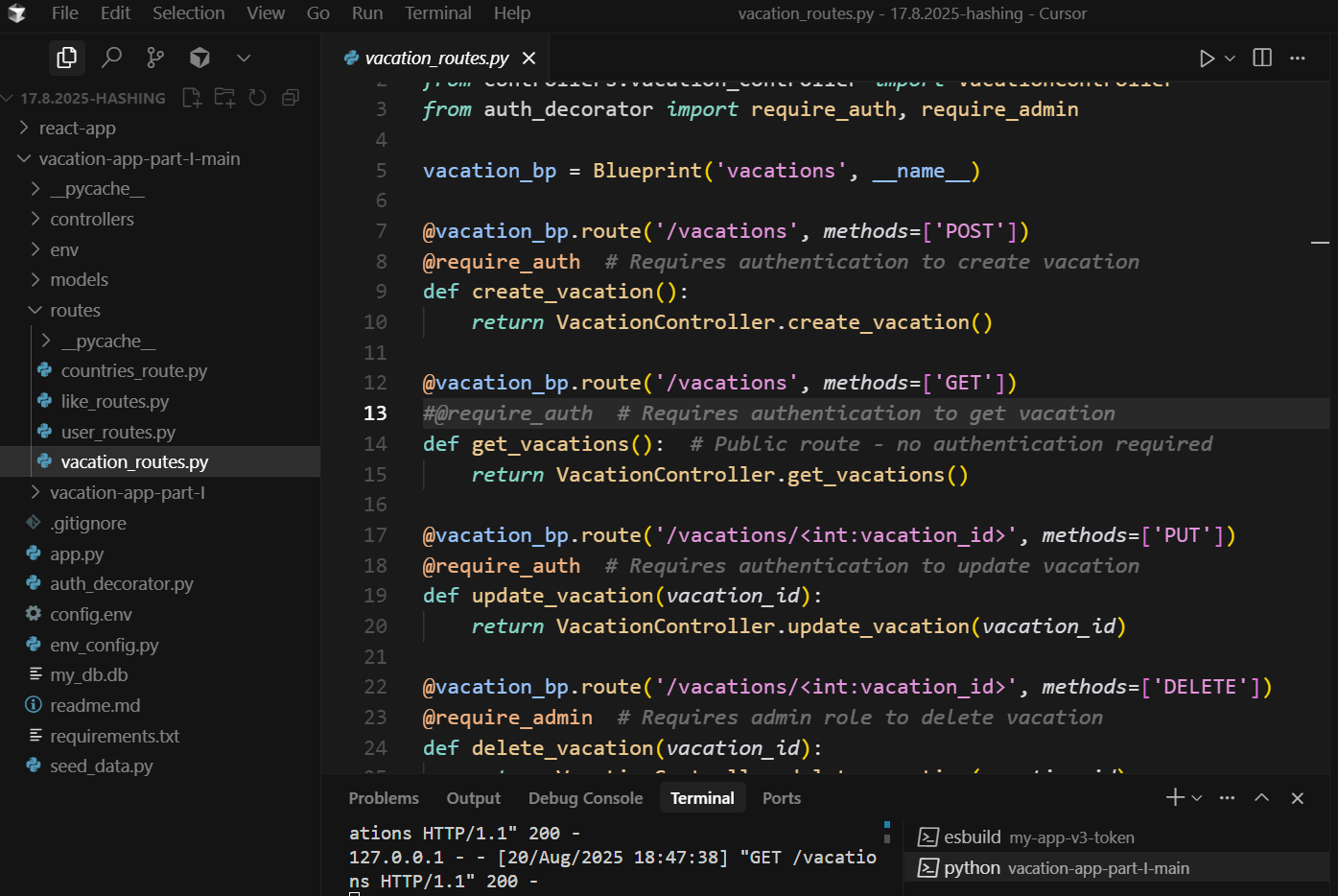
בניית טופס משתמש ב apiVacations

שלב א רק ללוג

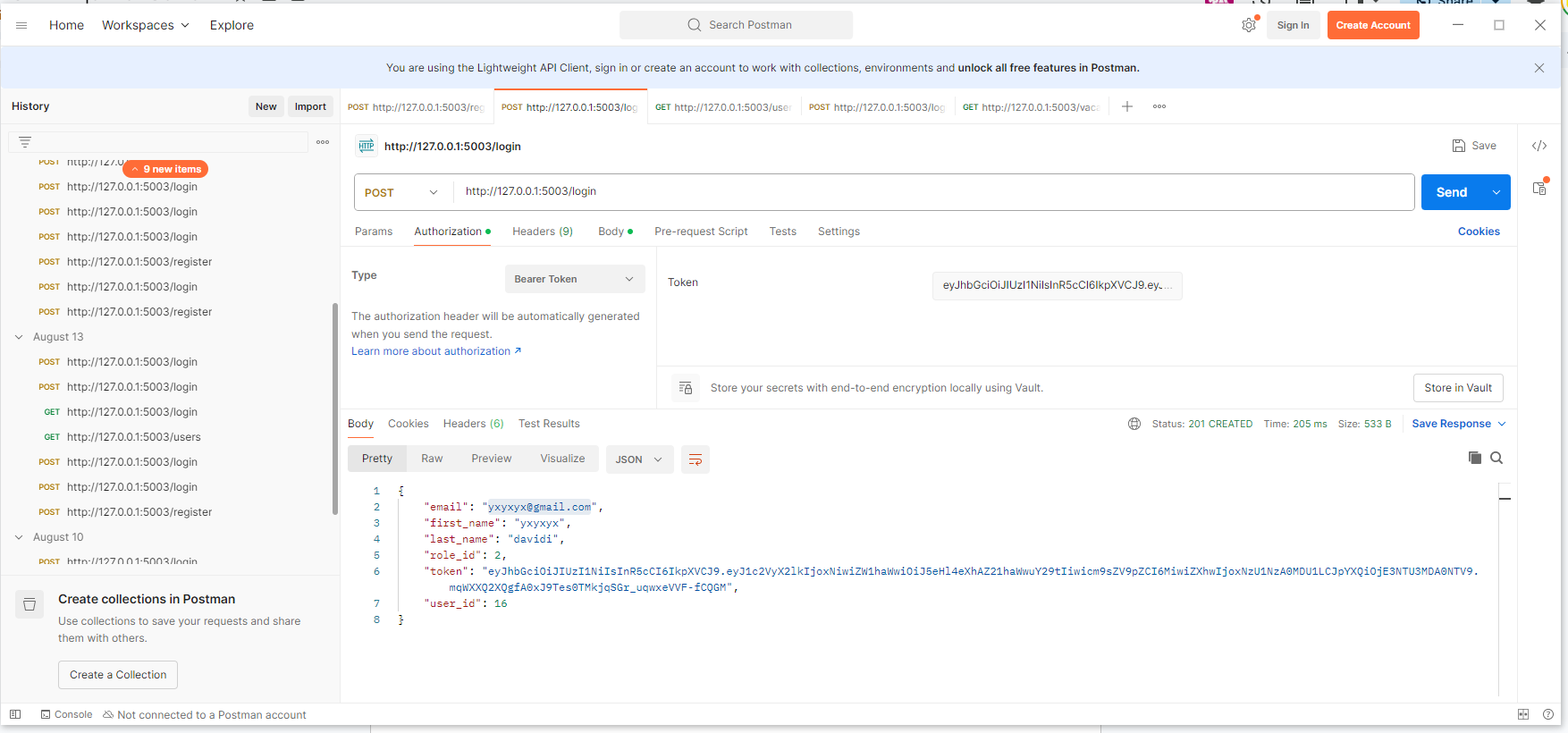
אני מעונין למטרת debug להביא בקומפוננטה vacations את כל החופשות לאחר לחיצה על כפתור. מקם לי כפתור שלחיצה עליו תחזיר את כל החופשות ותדפיס ל console את מערך החןפשות בקופוננטה זו C:\Users\jbt\Desktop\17.8.2025-hashing\react-app\my-app-v3-token\src\pages\Vacations.jsx כמובן אעזר ב apiVacations שבנינו C:\Users\jbt\Desktop\17.8.2025-hashing\react-app\my-app-v3-token\src\api\apiVacations.js

סדר פעולות יצירת הפרויקט

1. טיפול בצד שרת מוקלט בתאריך 17/8
   1. הוספת יכולת הצפנת סיסמא בקונטרולר של user
   2. טיפול בפונקצית login - החזרת token
   3. כתיבת auth\_decorator.py
   4. חסימת routing באמצעות auth\_decorator.py



* 1. בדיקה שהכל עובד מושלם עם postman כולל שליחת autherization



טיפול בקליינט

1. פתיחת פרויקט חדש
2. פתיחת תיקיות contexts, component, page, api
3. פתיחת כל הקומפוננטות הדרושות ריקות
4. התקנת react-router-dom
5. בניית מעטפת כולל navbar שמנווט בין הקומפוננטות
6. יצירת context user
7. מתחילים קומפוננטת register
8. קומפוננטת login
9. בדיקות שהכל עובד
10. ביצוע crud עבור vacation