الجمهورية العربية السورية

وزارة التعليم العالي والبحث العلمي

جامعة تشرين

هندسة الاتصالات والإلكترونيات

-------------------------------------------------------------------------------------------

**برمجة الشبكات**

**الوظيفة الثانية**

إعداد الطالب:

زين العابدين حسن بربهان

(2680)

المشرف:

د.مهند عيسى

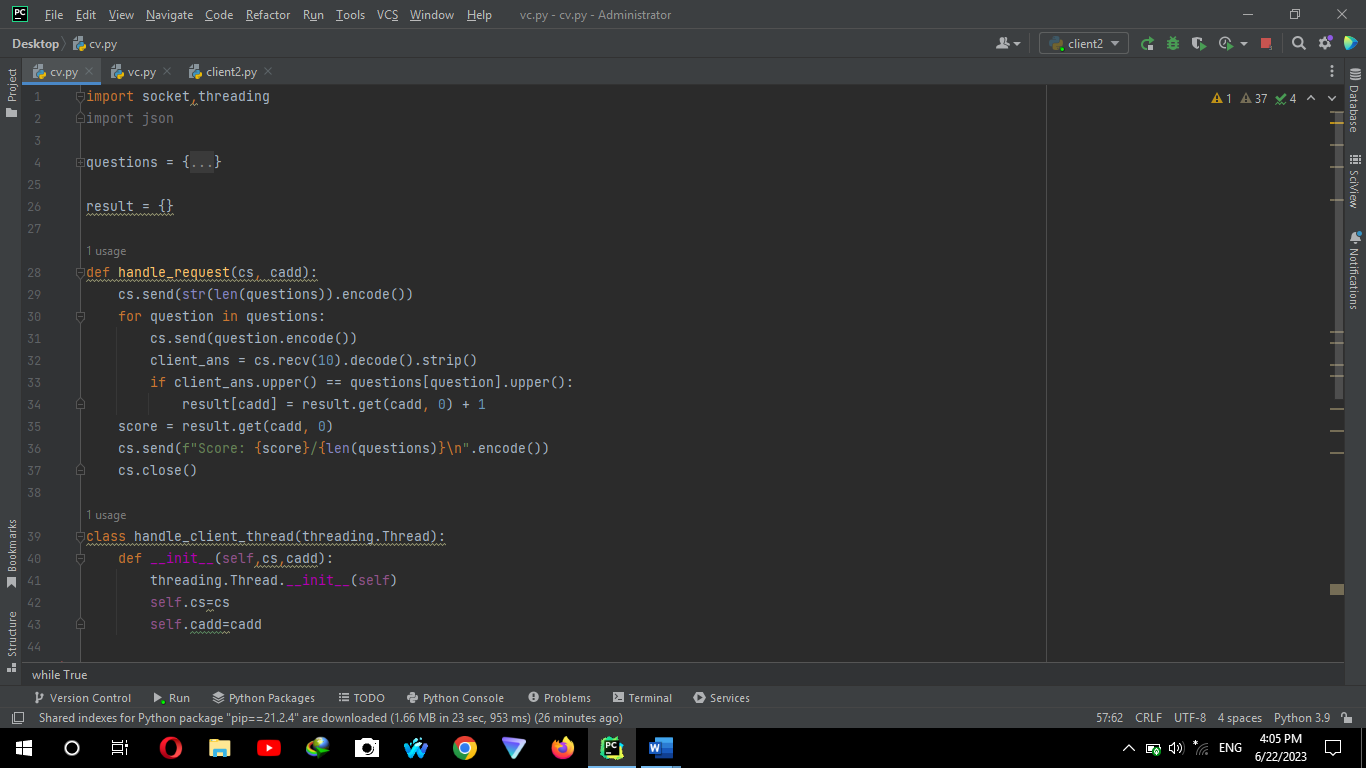
**Question 1:** **TCP Server/Client Quiz App with Multi-threading?**

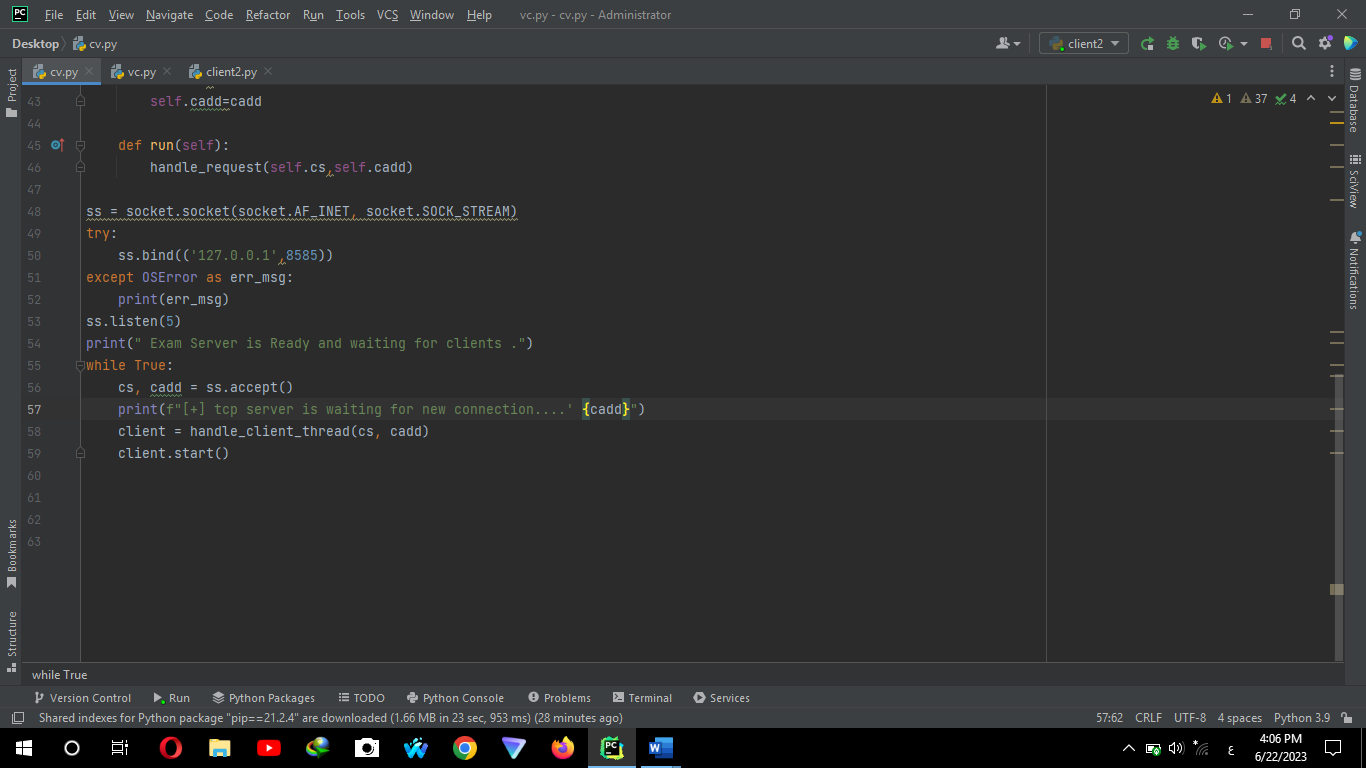
As an improvement to previous first homework, build a TCP server and client quiz application using Python. The server should handle multiple client connections simultaneously using multi-threading. The application should allow clients to connect, participate in a quiz, and receive their quiz scores upon completion.

**Requirements**:

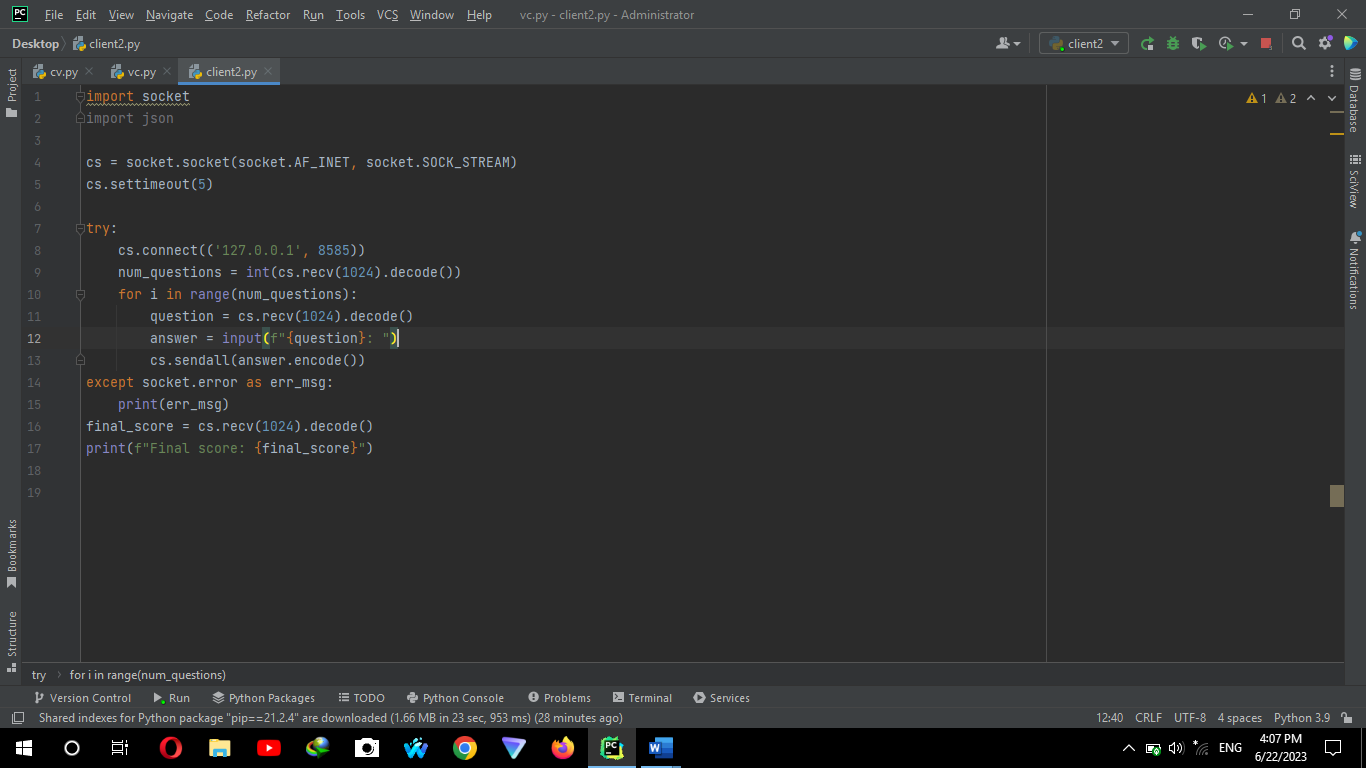
1. The server should be able to handle multiple client connections concurrently.
2. The quiz should consist of a set of pre-defined questions stored on the server.
3. Each client should connect to the server and receive the quiz questions.
4. Clients should send their answers to the server.
5. The server should keep track of the scores for each client.
6. At the end of the quiz, the server should send the final scores to each client.

Server

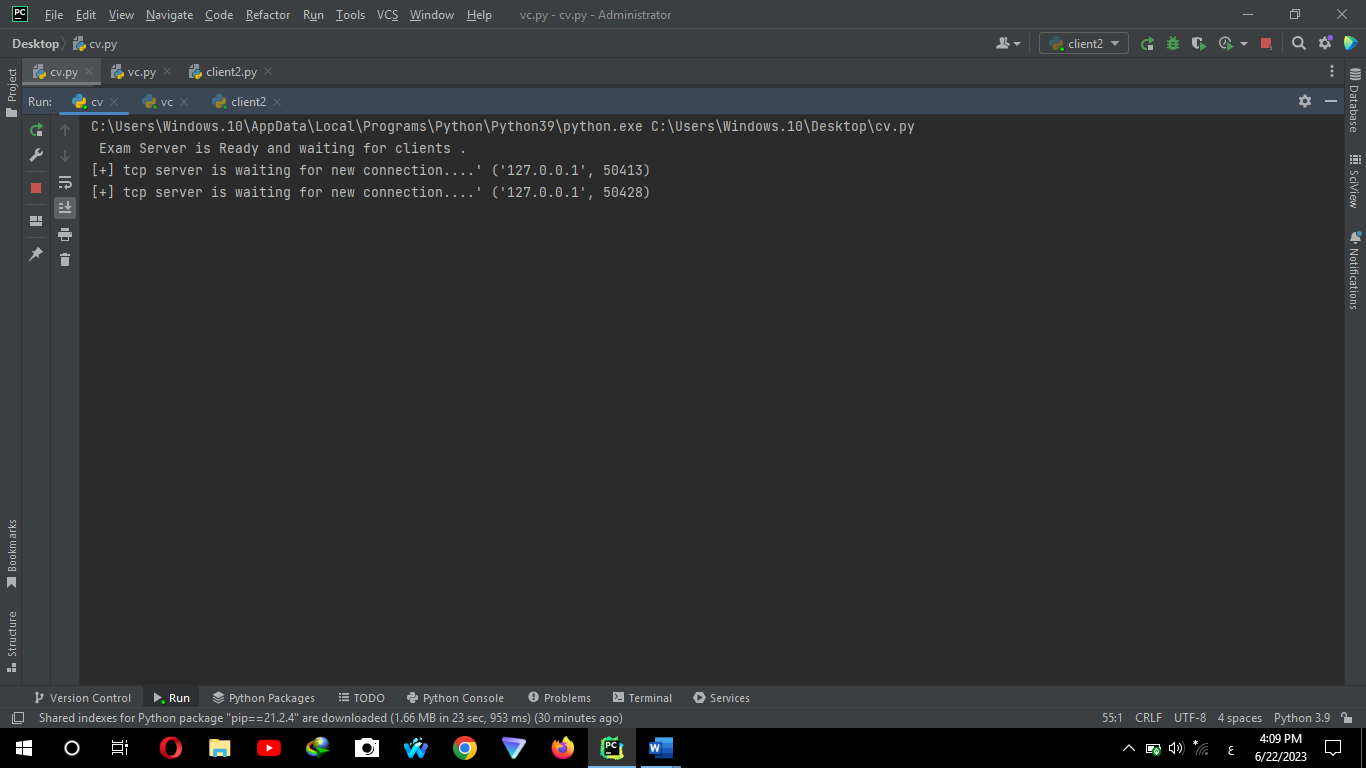


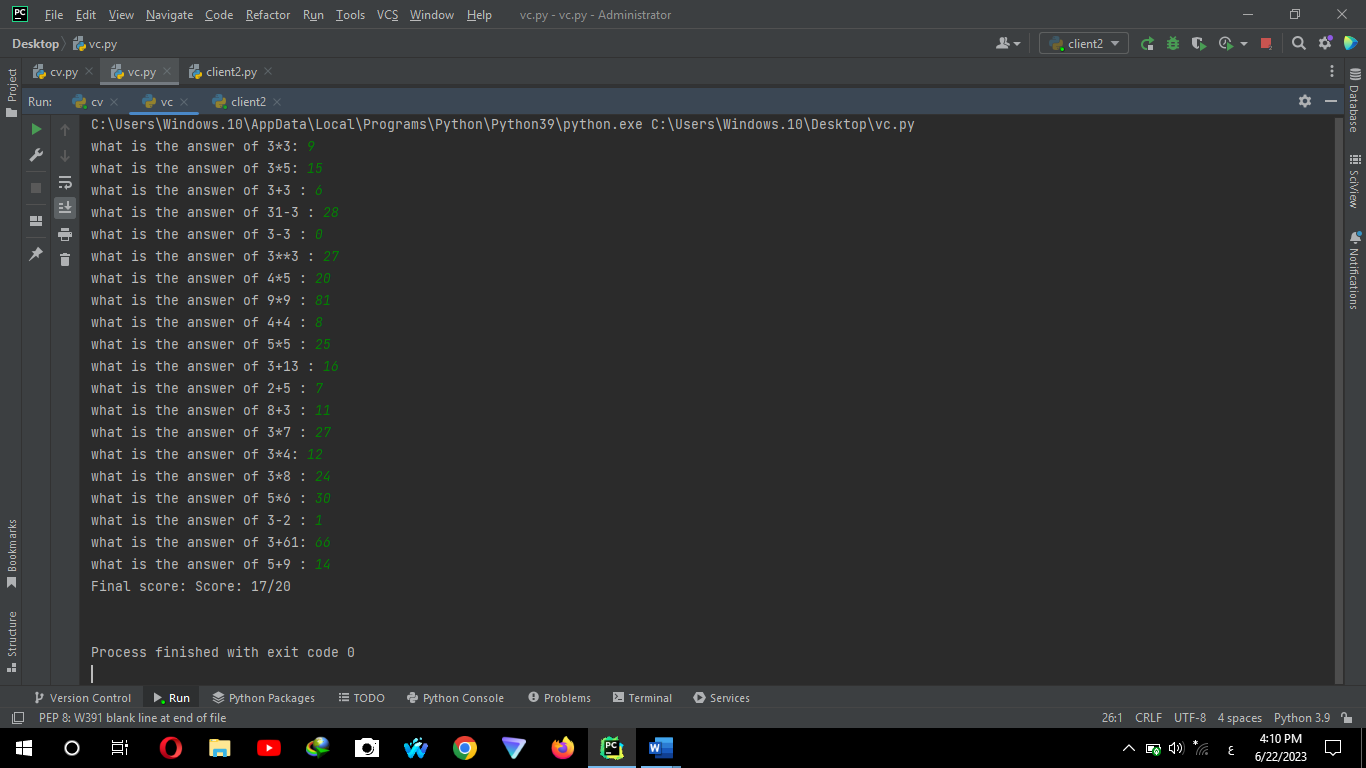


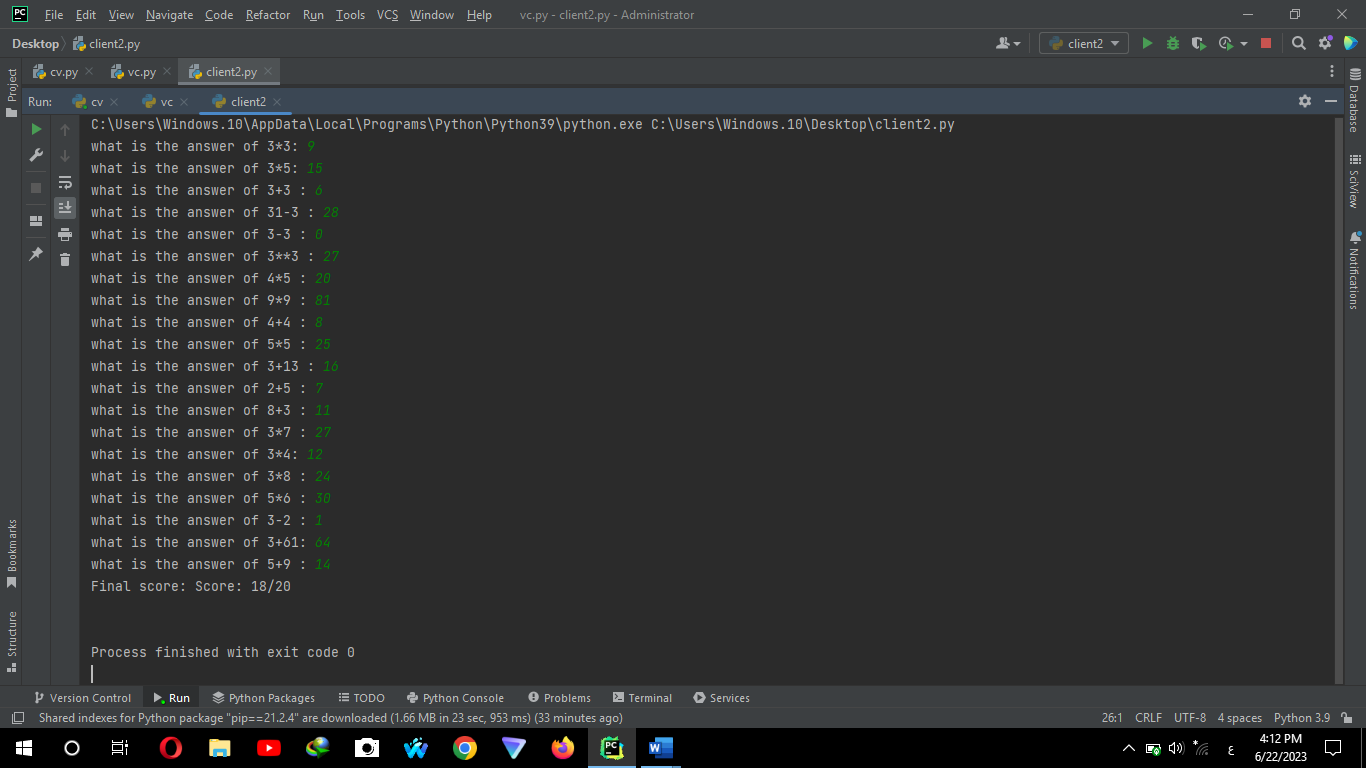
Client



عند تشغيل السيرفر



 الزبون الأول الزبون الثاني



شرح الكود الأول:

هذا الكود يمثل خادمًا (Server) يقوم بإجراء اختبار (Exam) على المتصلين به. يستخدم البروتوكول TCP/IP لإرسال واستقبال البيانات من وإلى المتصلين. يتم فتح المنفذ 8585 للاستماع على الاتصالات الواردة.

تم تعريف قائمة الأسئلة (Questions) التي سيتم استخدامها في الاختبار، مع الإجابات الصحيحة المقابلة لكل سؤال. يتم إرسال الأسئلة إلى المتصلين ويُطلب منهم إرسال الإجابات. يتم حساب النتيجة النهائية لكل متصل عن طريق عدد الإجابات الصحيحة التي قدمها.

تم إنشاء فئتين، الأولى `handle\_request` تستخدم لإجراء عملية التحقق من الإجابات المقدمة من المتصل، وتحسب النتيجة النهائية. الثانية `handle\_client\_thread` تستخدم لإنشاء خيوط (Threads) جديدة لكل متصل يتصل بالخادم.

يتم إنشاء كائن `socket` باستخدام `AF\_INET` و `SOCK\_STREAM`، والذي يتم استخدامه لإنشاء الاتصالات بين المتصلين والخادم. تم استخدام استثناء الـ `OSError` للتحقق من وجود أية أخطاء في عملية الربط.

أخيرًا، يتم استدعاء `accept` للانتظار على المتصلين الجدد وقبول الاتصالات الواردة. عند وصول متصل جديد، تم إنشاء خيط جديد للتعامل مع المتصل.

شرح الكود الثاني:

هذا الكود يمثل عميل (Client) يتصل بالخادم السابق ويتفاعل معه لإجراء الاختبار.

تم استخدام مكتبة `socket` لإنشاء كائن Socket، وهو ما سيستخدم لإنشاء الاتصال بالخادم. تم استخدام `AF\_INET` و `SOCK\_STREAM` للاتصال باستخدام بروتوكول TCP/IP.

يتم تعيين قيمة الـ `settimeout` إلى 5 ثوانٍ، وهو ما يعني أنه إذا لم يتم الرد على الاتصال خلال هذه الفترة، سيتم إثارة استثناء `socket.timeout` وإيقاف الاتصال.

بعد الاتصال بالخادم، يتم استقبال عدد الأسئلة الذي سيتم طرحها في الاختبار، ثم يتم طرح الأسئلة على المستخدم من خلال استخدام `input` وإرسال الإجابات المقدمة من المستخدم إلى الخادم باستخدام `sendall`.

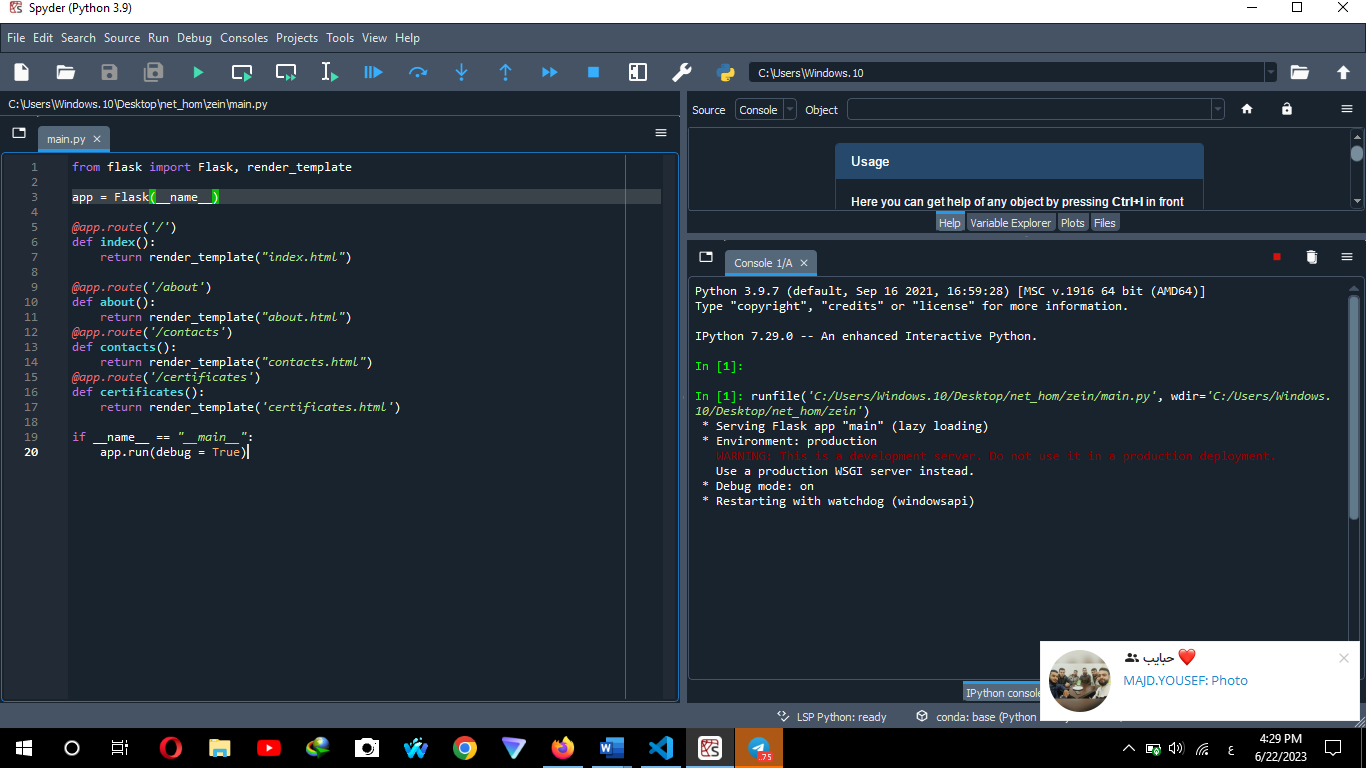
يتم استقبال النتيجة النهائية للمتصل من الخادم باستخدام `recv` ويتم طباعة النتيجة النهائية للمستخدم. يتم التحقق من وجود أية أخطاء في عملية الاتصال باستخدام استثناء `socket.error` وإظهار رسالة الخطأ المناسبة إذا حدث خطأ.

**Question 2:** **Simple Website with Python Flask Framework**

Create a simple website with multiple pages using Flask, HTML, CSS, and Bootstrap. The website should demonstrate your understanding of web design principles.

**Requirements**:

1. Set up a local web server using XAMPP, IIS, or Python's built-in server (using Flask).
2. Apply CSS and Bootstrap to style the website and make it visually appealing.
3. Ensure that the website is responsive and displays correctly on different screen sizes.
4. Implement basic server-side functionality using Flask to handle website features.



شرح الكود :

هذا الكود يمثل تطبيق Flask الذي يقدم عددًا من الصفحات التي يمكن الوصول إليها عن طريق العناوين المحددة.

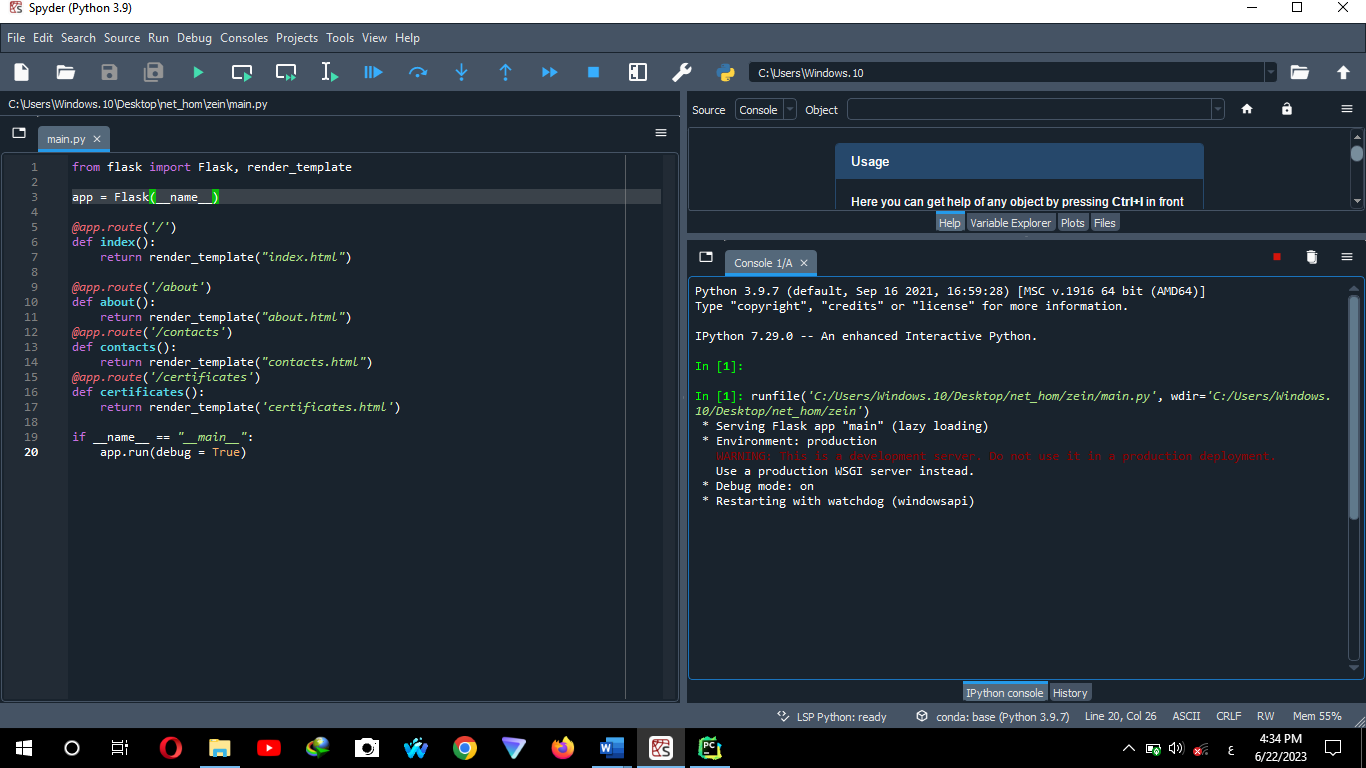
تم استيراد الوحدات `Flask` و `render\_template` من إطار عمل Flask. تم إنشاء كائن Flask باستخدام اسم التطبيق `\_\_name\_\_`، ويتم تعيين كائن `app` إلى هذا الكائن.

يتم استخدام الزخم `@app.route` لتحديد عناوين URL المرتبطة بكل صفحة. على سبيل المثال، يتم استخدام `@app.route` لتحديد عنوان URL للصفحة الرئيسية، و `@app.route('/about')` لصفحة المعلومات، وهكذا.

تم إنشاء دالة `index` للصفحة الرئيسية و `about` و `contacts` و `certificates` لصفحات الأخرى. يتم استخدام `render\_template` للعثور على ملفات HTML المرتبطة بكل صفحة وإرجاعها كمحتوى استجابة.

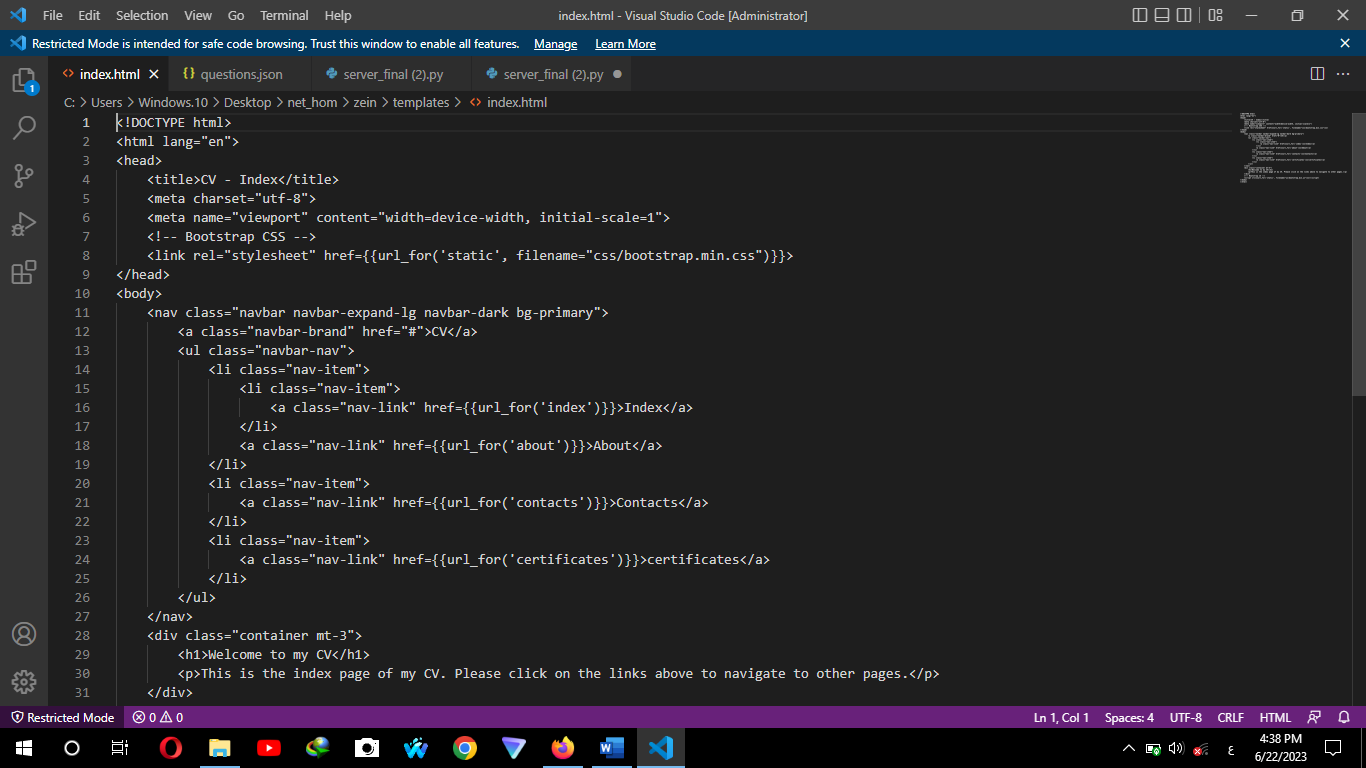
أخيرًا، يتم استخدام `if \_\_name\_\_ == "\_\_main\_\_":` للتأكد من أن الكود يتم تنفيذه فقط عند تشغيل الملف كبرنامج مستقل، ويتم تشغيل الخادم باستخدام `app.run`، مع تمكين وضع التصحيح `debug=True` لتشغيل الخادم في وضع التصحيح.

ويعطي عند التشغيل :

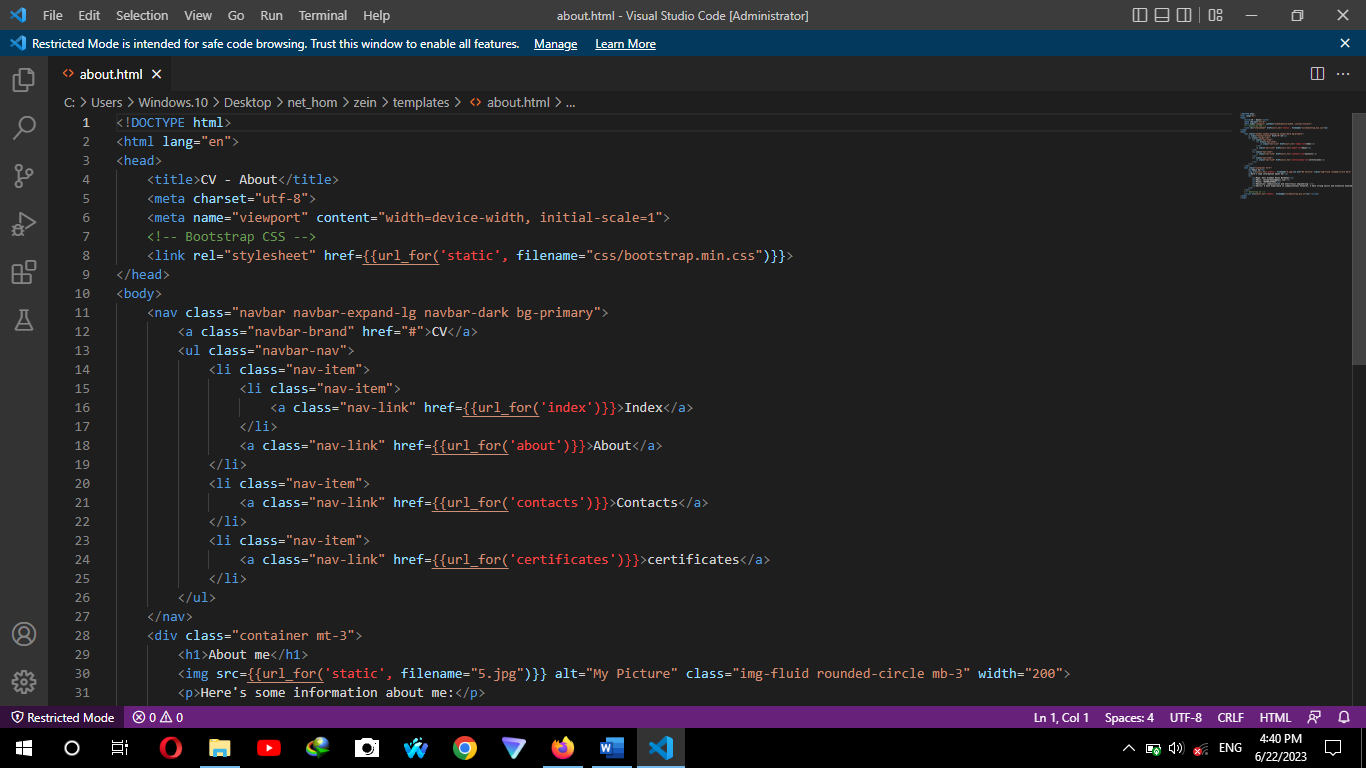


محتوى صفحات الويب:

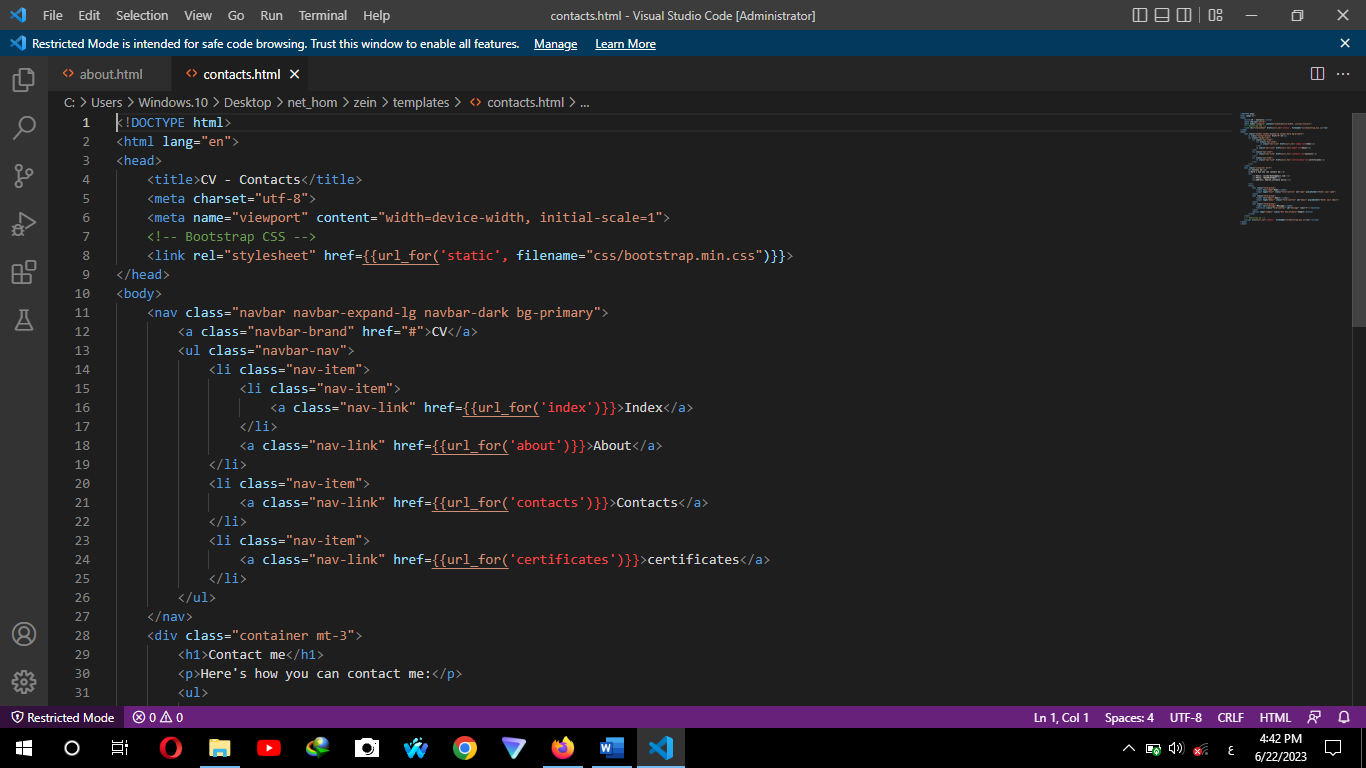
index.html



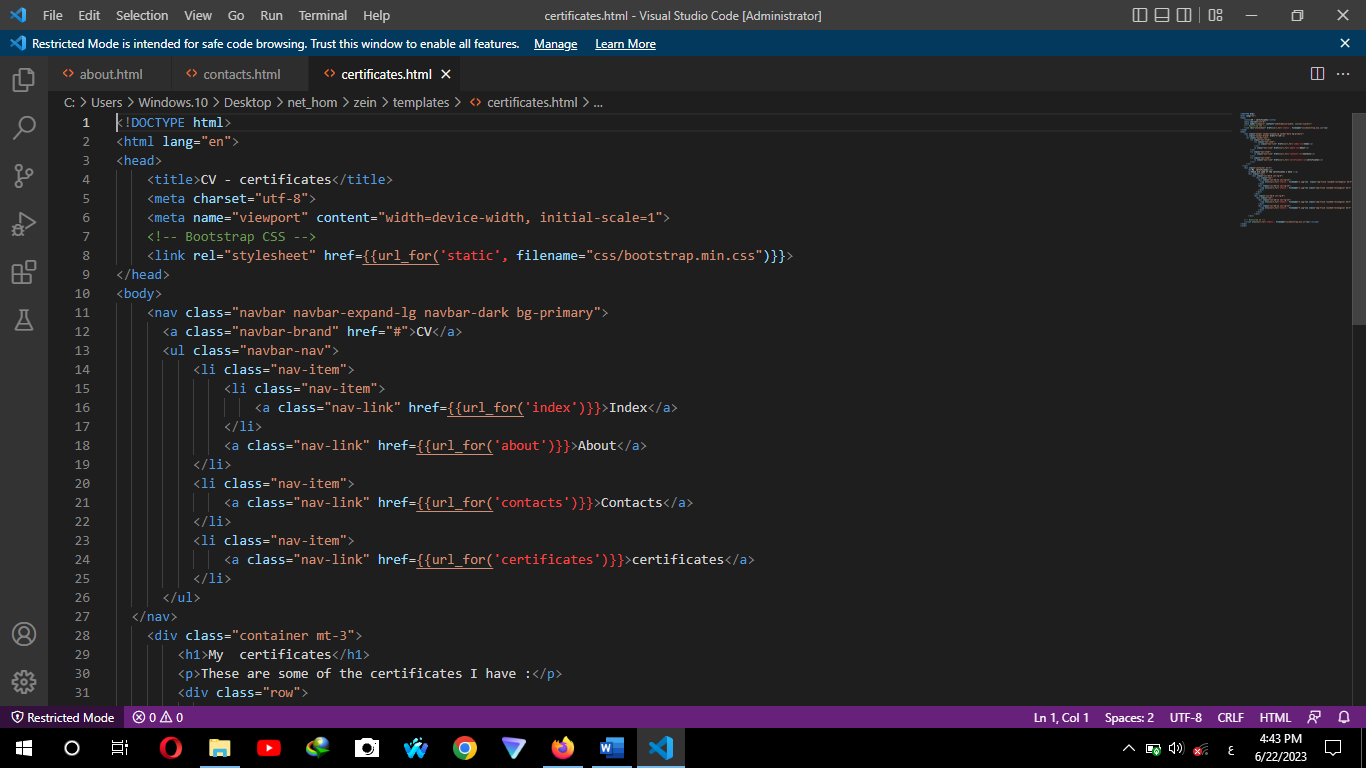
About.html



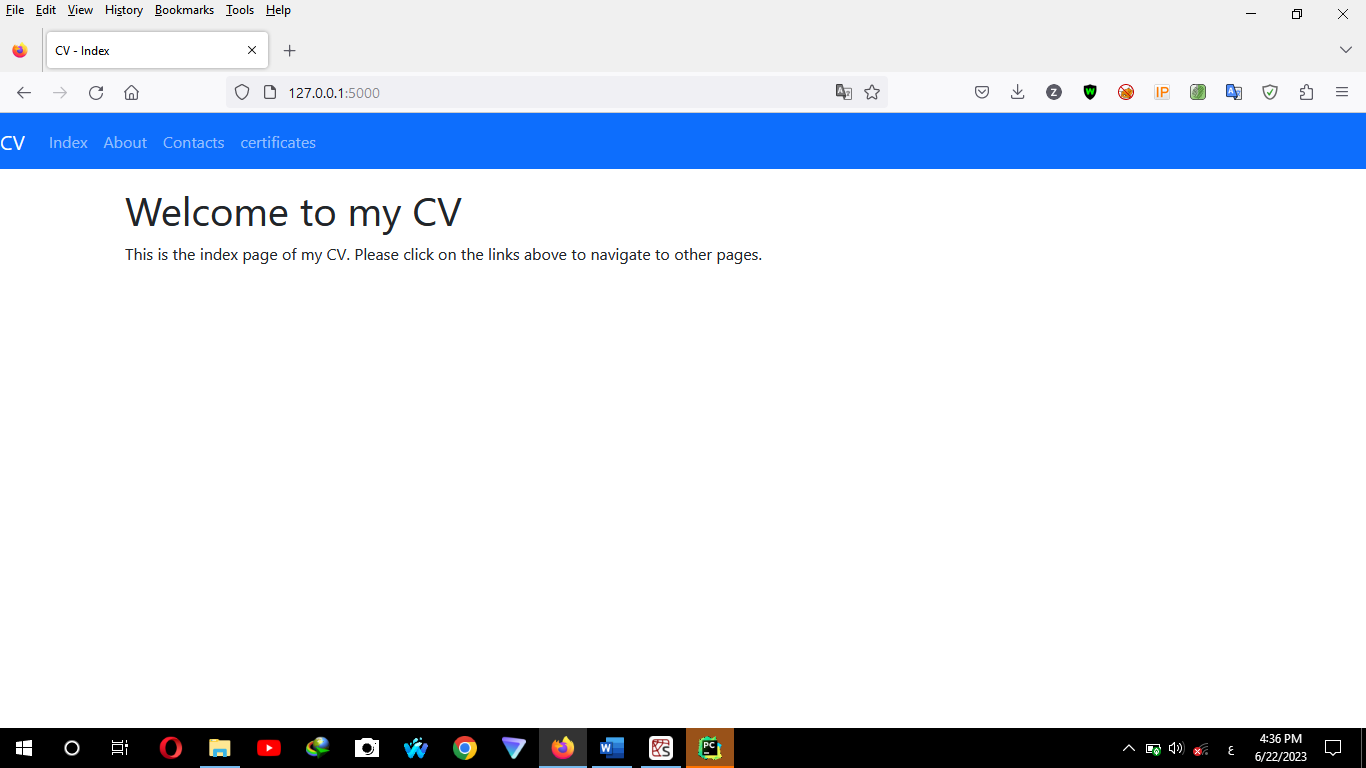
contacts.html



certificates.html



الأن نذهب للمتصفح ونكتب 127.0.0.1:5000

فيظهر:

