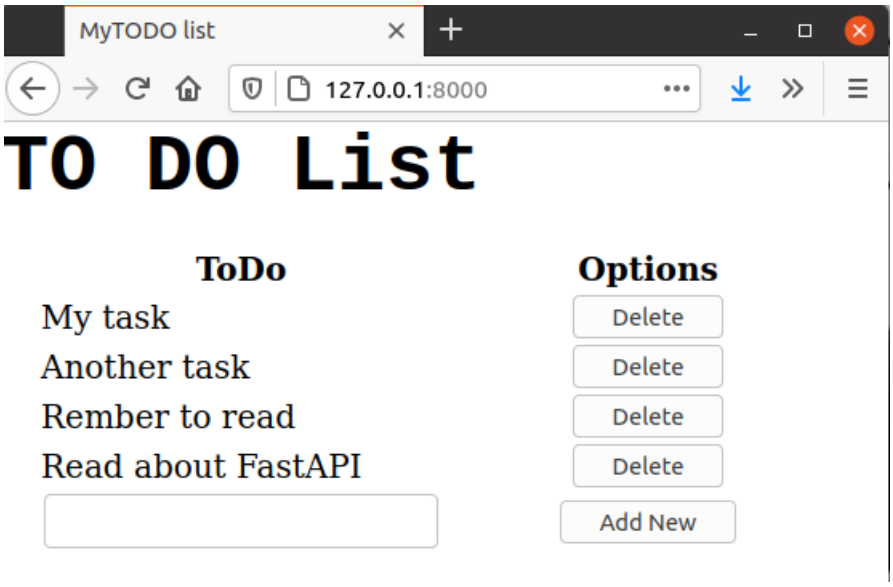
FastApi - To Do List

FastAPI is one of the **many popular** web frameworks of python used by professional software developers for making web apps easily. It provides a backbone to the app due it's speed, efficiency and easy-to-use-and-understand nature.

It is based on Starlette and ASGI which are essential for the speed of FastAPI.

We are going to make an ToDo api *application*. We are going to make a simple ✔ todo list *application*. We will cover all basics of starting a **FastAPI application** from scratch. The application will include **routing**, **storing** data, **reading** the data and showing it in template (HTML) and **adding** and **deleting** todo tasks.



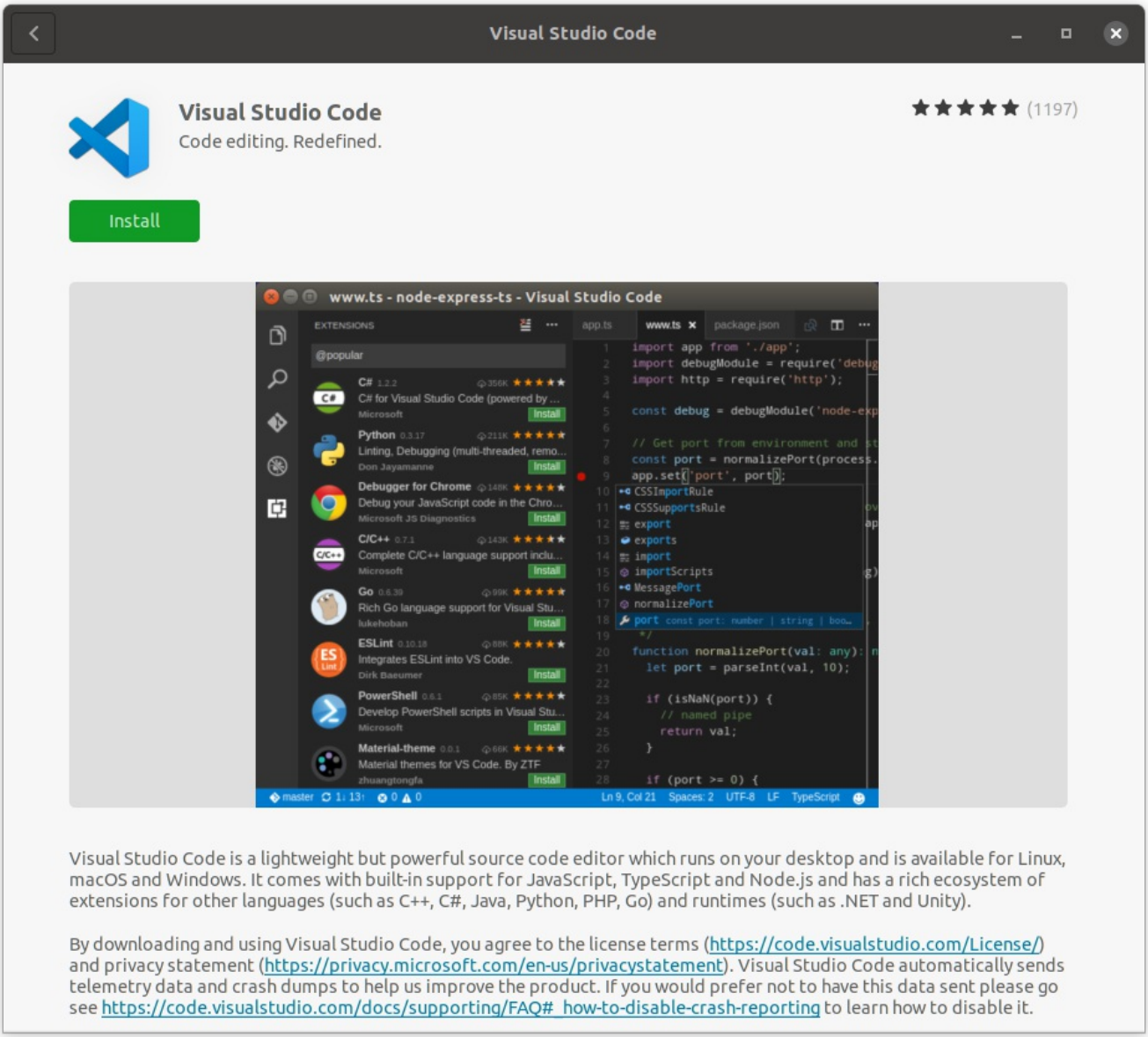
Step 1 - Visual Studio Code

We are going to use *Visual Studio Code* for this. You have to create tre files in *Visual Studio Code*

- main.py
- todolist.html
- database.json

Install Visual Studio Code on your Ubuntu virtual machine.

Use the Ubuntu Software installer.



Step 2 - Install

You need to create a virtual environment for this project. For that we are using **virtualenv**.

Linux - Ubuntu

In a terminal:

```
sudo pip install virtualenv
sudo apt-get install python3-venv
mkdir mytodo
cd mytodo
python3 -m venv todoenv
source todoenv/bin/activate
sudo apt install uvicorn
sudo apt-get install -y python3-uvloop
sudo pip3 install httptools
sudo pip install fastapi uvicorn jinja2 python-multipart
```

FastAPI doesn't have it's server like Django and Flask, so **Uvicorn** is an ASGI server which will be used for production and serving of a FastAPI.

Step 3 - main.py

Now we are building our basic API route and run it, using **uvicorn**.

We will start by running

```
code .
```

in the terminal it opens **Visual Studio Code** for you

Make a new file named **main.py** in the **mytodo** folder. (**Do not touch the todoenv folder!!**) and write the below code in it:

```
from fastapi import FastAPI
app = FastAPI()
@app.get("/")
async def root():
    return {"message": "Hello World"}
```

Step 4 - Run

Go back to the terminal and run:

```
uvicorn main:app --reload
```

Open **http://localhost:8000/**

You should see:

```
{"message":"Hello World"}
```

Congratulations! You have successfully made an API!

Explanation

In the **main.py** we have first imported the required **FastAPI()** function and used it to declare the app.

Then, we use a **decorator** to define the routing of the root function. In the decorator, the important bits are the function **get()** and the parameter passed in the same.

Here, **get** refers to the type of request the url should accept to run the function and the parameter in the function is the url itself.

A **/** url also means that even if nothing is typed after **localhost:8000**, still the function will run i.e. **/** is an optional url if nothing is typed after it.

Step 5 - Create HTML

We installed **jinja2** during pip installs. It is a template engine used for many tasks related to templates. We are now going to use it for rendering our templates.

To create the template, create the file **todolist.html** in the folder **/mytodo/templates/**.

Use *Visual Studio Code* for this.

The content of the **HMTL** file has to be:

```
<html>
    <head>
        <title>MyTODO list</title>
    </head>
    <style>
        *{
            margin: 0;
        }
        table {
            align-items: center;
            margin-right: auto;
            margin-left: auto;
        }
        h1 {
            width: fit-content;
            font-family: 'Courier New', Courier, monospace;
            margin-left: auto;
            margin-right: auto;
            font-size: 50px;
        }
        th,td {
            width: 250px;
            justify-content: center;
            font-size: 20px;
            font-family: 'Lucida Sans';
        }
        td:nth-child(2) {
            text-align: center;
        }
    </style>
    <body>
        <h1>My TO DO list</h1>
        <br/>
        <table>
            <tr>
                <th>ToDo</th>
                <th>Options</th>
            </tr>
            {% for id in tododict %}
            <tr>
                <td>{{ tododict[id] }}</td>
                <td><a href="/delete/{{ id }}"><button>Delete</button></a></td>
            </tr>
            {% endfor %}
            <tr>
                <form method="POST" action="/add">
                <td><input type="text" name="newtodo" required></td>
                <td style="text-align: center;"><button type="submit">Add New</button></td>
                </form>
            </tr>
        </table>
    </body>
</html>
```

You can get the code at this link:
https://gist.github.com/officegeek/bd889aa501f78211129d5c3d72918801

Step 6 - main.py api

In *main.py make the flowing changes to the code:

```
from fastapi import FastAPI, Request
from fastapi.responses import RedirectResponse
from fastapi.templating import Jinja2Templates
import json

app = FastAPI()
templates = Jinja2Templates(directory="templates")

@app.get("/")
async def root(request: Request):
    with open('database.json') as f:
        data = json.load(f)
    return templates.TemplateResponse("todolist.html",{"request":request,"tododict":data})

@app.get("/delete/{id}")
async def delete_todo(request: Request, id: str):
    with open('database.json') as f:
        data = json.load(f)
    del data[id]
    with open('database.json','w') as f:
        json.dump(data,f)
    return RedirectResponse("/", 303)

@app.post("/add")
async def add_todo(request: Request):
    with open('database.json') as f:
        data = json.load(f)
    formdata = await request.form()
    newdata = {}
    i=1
    for id in data:
        newdata[str(i)] = data[id]
        i+=1
    newdata[str(i)] = formdata["newtodo"]
    print(newdata)
    with open('database.json','w') as f:
        json.dump(newdata,f)
    return RedirectResponse("/", 303)
```

You can get the code at this link:
https://gist.github.com/officegeek/deb8b8996e30ee16c2e9e6415b17d326

Step 7 - Database

Make a new file in *Visual Studio Code* - **database.json**.

Save the file in the folder **/mytodo/**, same place as the **main.py** file.

The content of the file has to be:

```
{"1": "My task", "2": "Another task", "3": "Rember to read", "4": "Read about FastAPI"}
```

You can get the code at this link:
https://gist.github.com/officegeek/4396b3c3b40a41b7544700997dcafe14

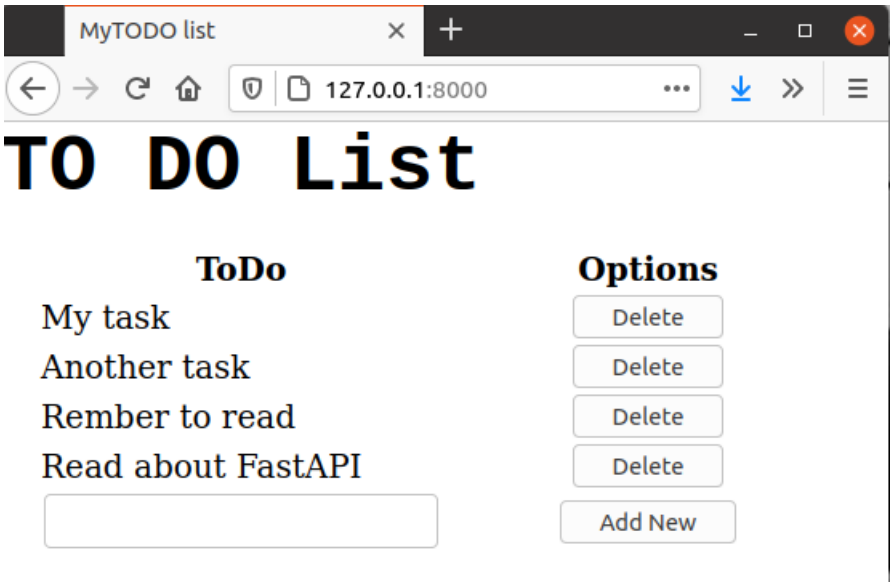
Step 8 - Run the final API

If you still have the webpage open you can just refresh the page or you have to go back to the terminal and run:

```
uvicorn main:app --reload
```

Open **http://localhost:8000/**

Try **adding** and **deleting** tasks and checkout the **database.json** file in between to get some idea about the process.

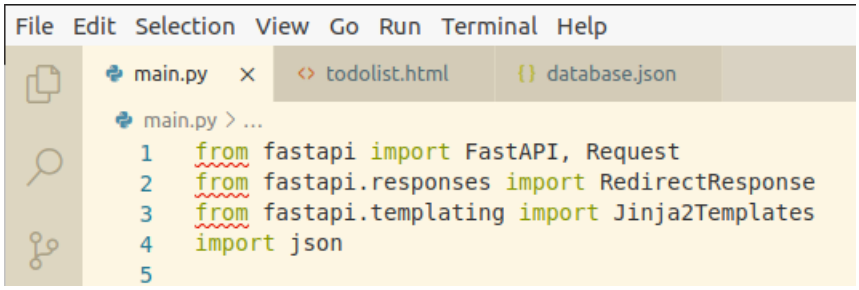


Directory

The working directory of the project should look like this for the project to work correctly:

- /mytodo
    - /templates
        - todolist.html
    - /todoenv
- database.json
- main.py

The 3 files in **Visual Studio Code**



Understanding the code

Lest get a better understanding of the code

main.py

In **main.py** line 1 to 4 says:

```
from fastapi import FastAPI, Request
from fastapi.responses import RedirectResponse
from fastapi.templating import Jinja2Templates
import json
```

We have imported the required functions from **FastAPI** and imported **json**. The **RedirectResponse** function will help us redirect to the main-page after going to **add or delete API**.

The **Jinja2Templates** will be useful for using templates while parsing data and variables in templates. We are using **json library** to **store** and **read** data from a json file which we will use as a database to store our To-do's.

We have defined a templates variable by providing a valid templates directory for us to directly use the templates in it.

```
templates = Jinja2Templates(directory="templates")
```

In the code snippet below:

```
@app.get("/")
async def root(request: Request):
    with open('database.json') as f:
        data = json.load(f)
    return templates.TemplateResponse("todolist.html",
"request":request,"tododict":data})
```

we open **database.json** and read it's content using the loads function from json library. The data is passed to the **template** using the dictionary object in the **TemplateResponse** function.

This function renders the template for us, whose **name** we provide as **first parameter** and parses the **variables**, we pass as dictionary object in the template.

It is necessary to provide the request variable to the template for rendering

Jump to the **todolist.html** template

```
{% for id in tododict %}
<tr>
    <td>{{ tododict[id] }}</td>
    <td><a href="/delete/{{ id }}"><button>Delete</button></a></td>
</tr>
{% endfor %}
```

This is the most *confusing/interesting/important* part. Here, we are using template formatting to use the variables that were passed and also using Python inside our template.

The for **loop**, loops over the to-do's and using **{{ variable_name }}** as a format we are making a new row for every todo and also making a **button** along with the todo specifically hyperlinked to the **"/delete/(id of the todo)"** which we have defined in **main.py** for deleting the todo.

The **{% endfor %}** provides the template a limit from where to where it has to repeat in for. You will also find the form to add the todo hyperlinked to **"/add"** to add a **new todo**.

Back in **main.py**, you can now understand the later defined **delete** and **add** API's.

```
@app.get("/delete/{id}")
async def delete_todo(request: Request, id: str):
    with open('database.json') as f:
        data = json.load(f)
    del data[id]
    with open('database.json','w') as f:
        json.dump(data,f)
    return RedirectResponse("/", 303)

@app.post("/add")
async def add_todo(request: Request):
    with open('database.json') as f:
        data = json.load(f)
    formdata = await request.form()
    newdata = {}
    i=1
    for id in data:
        newdata[str(i)] = data[id]
        i+=1
    newdata[str(i)] = formdata["newtodo"]
    print(newdata)
    with open('database.json','w') as f:
        json.dump(newdata,f)
    return RedirectResponse("/", 303)
```

In **delete API**, one new thing we can see is, the way in which the API is defined.

It is defined as **"/delete/{id}"**. Here the curly brackets signify that **any variable** data can be passed as a API route where the id will be passed as the parameter of the API function.

We are again **reading** the current database and deleting the specific todo. But *importantly* we then need to **update the database** with the latest data which we can do by opening the file in write mode this time - **"w"** - and **dumping** the json in the file.

We are then simply **redirecting** the user to our mainpage using our imported **RedirectResponse** function where we pass the redirect_url as **"/"** and the status code as **303** for temporary redirect.

After reading the data, we are getting the form that was posted from the html by awaiting the request.

Then, we give **proper id's** to each task to make sure, we add todo correctly and *THEN* add our new task. The request provides us formdata type object which is a dictionary, with keys corresponding to the names of the inputs in the form.

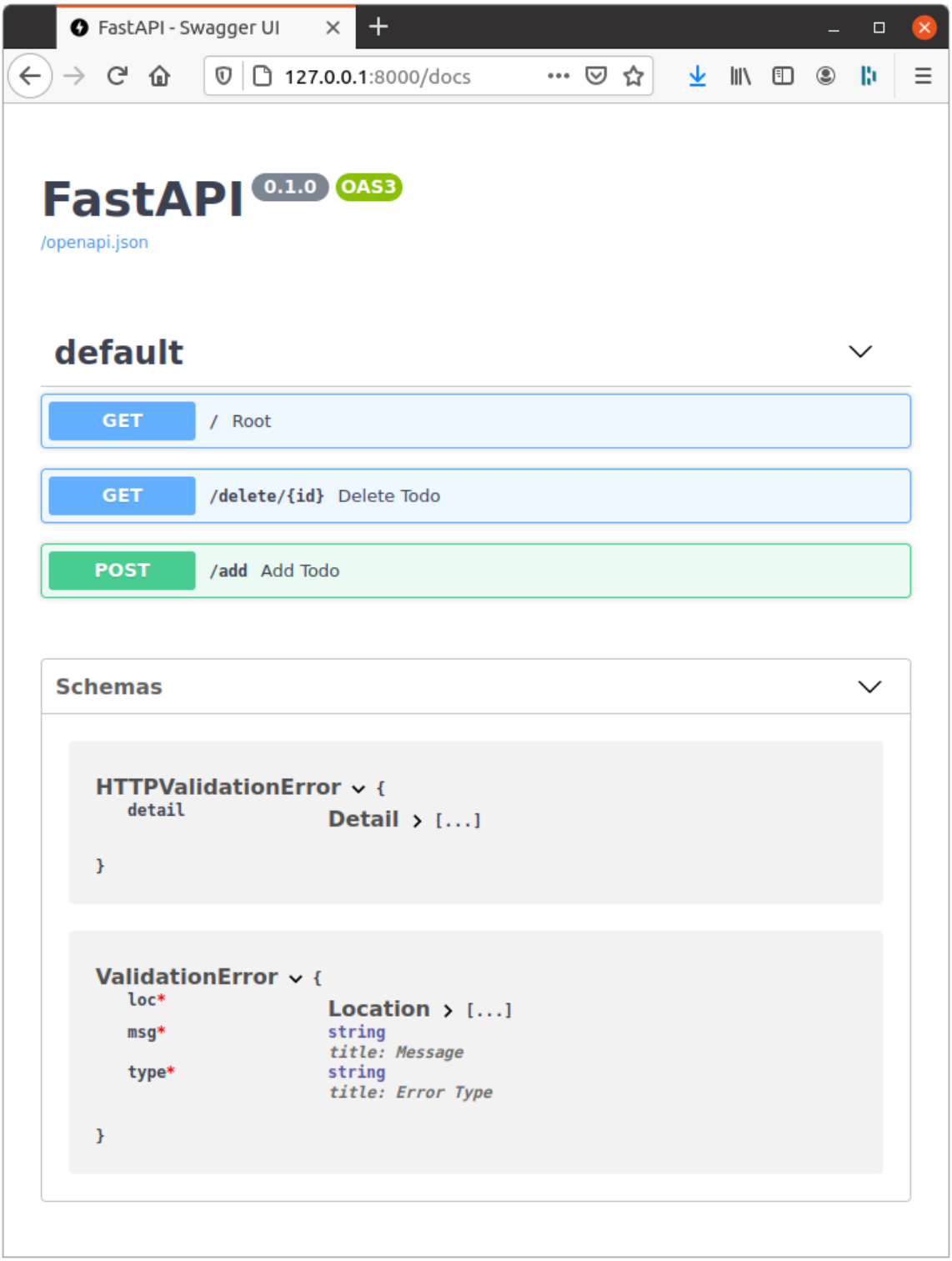
The only step left is to **update** our database and we are done, after **redirecting**.

Documentation

You can get the documentation for the **API** automatic, either by **docs** or **redoc**.

docs

Go to http://localhost:8000/docs (*while the server is running*) and checkout the API's automatic interactive API documentation, provided by **Swagger UI** - **https://github.com/swagger-api/swagger-ui**



redoc

Go to **http://127.0.0.1:8000/redoc** (*while the server is running*) and checkout the API's automatic interactive alternative API documentation, provided by **ReDoc** - **https://github.com/Redocly/redoc**