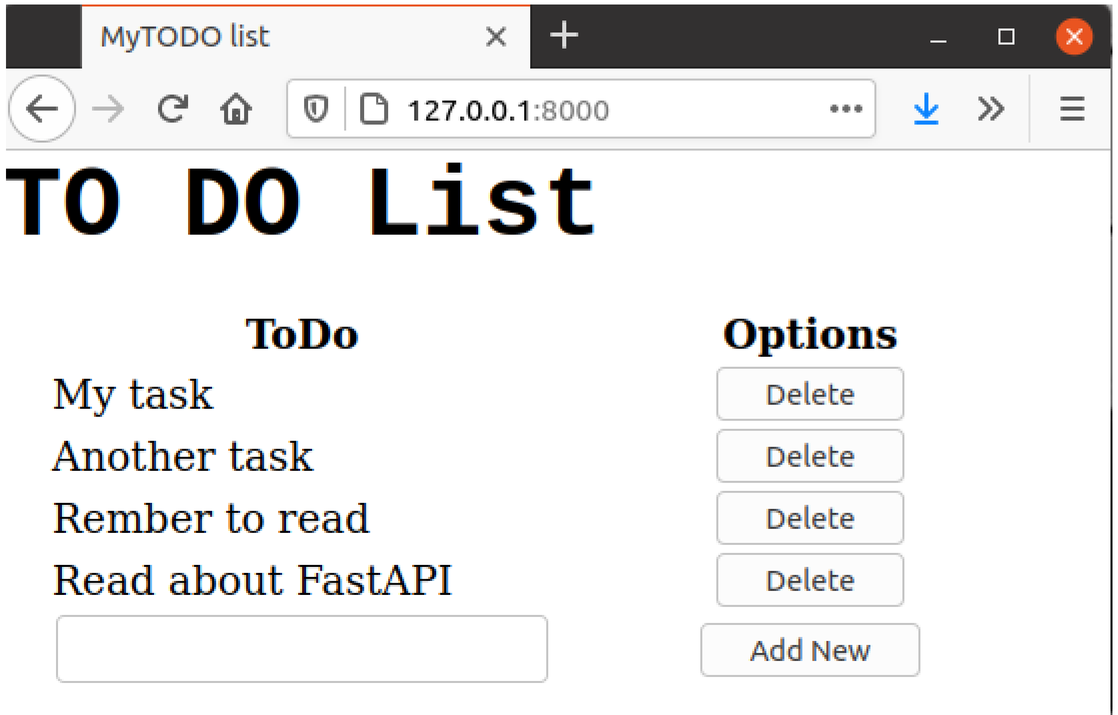
software developers for making web apps easily. It provides a backbone to the app due it’s speed, efficiency and easy-to-use-and-understand nature.

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.

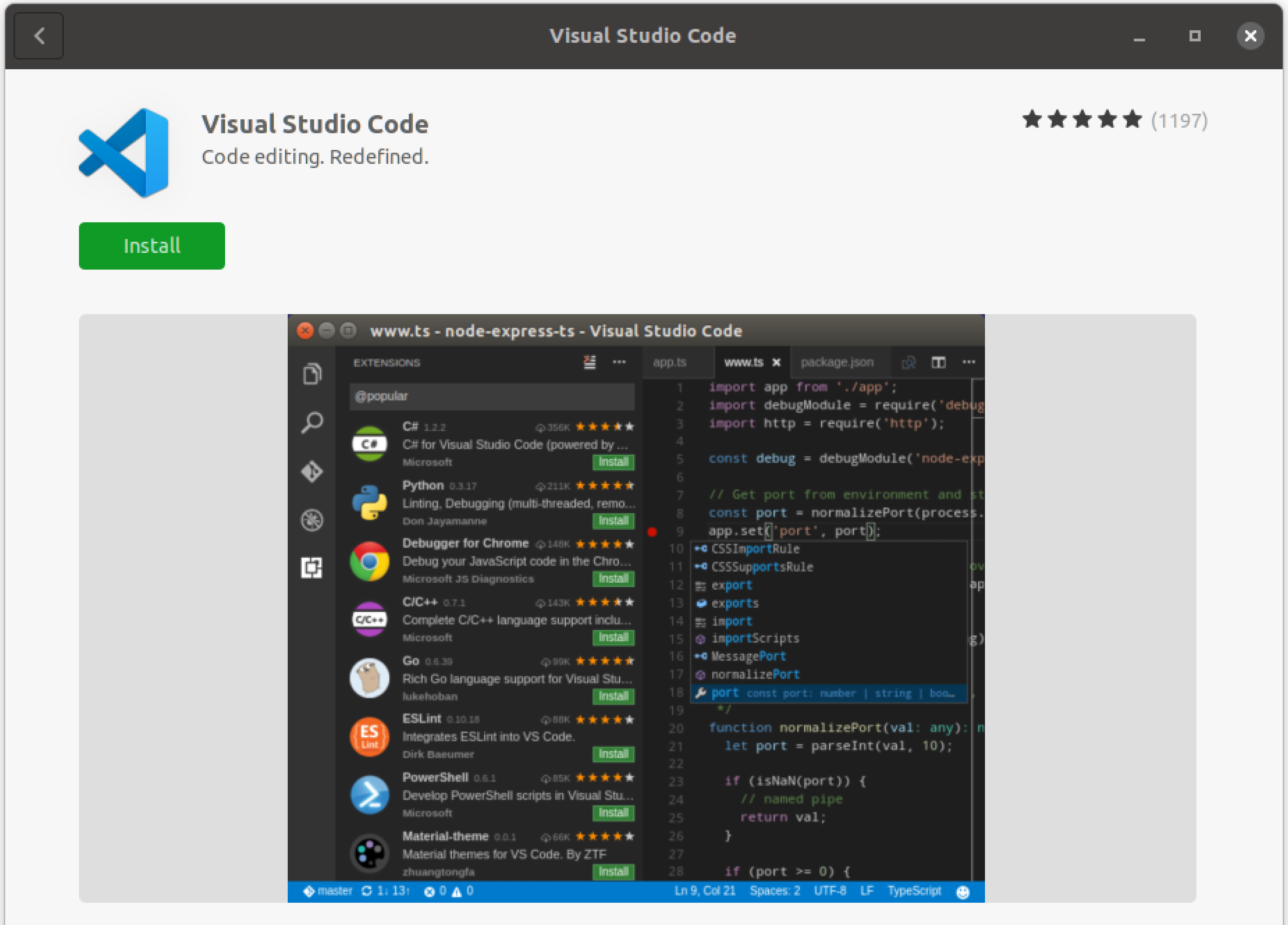

MyTODO list

127.0.0.1:8000

TO DO List

ToDo	Options
My task	Delete
Another task	Delete
Rember to read	Delete
Read about FastAPI	Delete
	Add New

Use the Ubuntu Software installer.



Visual Studio Code

Code editing. Redefined.

@popular

C# 1.2.2 — 356K — C# for Visual Studio Code (powered by … — Microsoft — Install

Python 0.3.17 — 211K — Linting, Debugging (multi-threaded, remo… — Don Jayamanne — Install

Debugger for Chrome — 148K — Debug your JavaScript code in the Chro… — Microsoft JS Diagnostics — Install

C/C++ 0.7.1 — 143K — Complete C/C++ language support inclu… — Microsoft — Install

Go 0.6.39 — 99K — Rich Go language support for Visual Stu… — lukehoban — Install

ESLint 0.10.18 — 88K — Integrates ESLint into VS Code. — Dirk Baeumer — Install

PowerShell 0.6.1 — 85K — Develop PowerShell scripts in Visual Stu… — Microsoft — Install

Material-theme 0.0.1 — 66K — Material themes for VS Code. By ZTF — zhuangtongfa — Install

```
import app from './app';
import debugModule = require('debug
import http = require('http');

const debug = debugModule('node-exp

// Get port from environment and st
const port = normalizePort(process.
app.set('port', port);
  CSSImportRule
  CSSSupportsRule
  export
  exports
  import
  importScripts
  MessagePort
  normalizePort
  port  const port: number | string | boo…
 */
function normalizePort(val: any): n
  let port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
```

master 1↓ 13↑ 0 0 Ln 9, Col 21 Spaces: 2 UTF-8 LF TypeScript

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.

<div style='page-break-after: always'></div>

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.

<div style='page-break-after: always'></div>

Step 5 - Create HTML

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:

```python
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

<div style='page-break-after: always'></div>

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

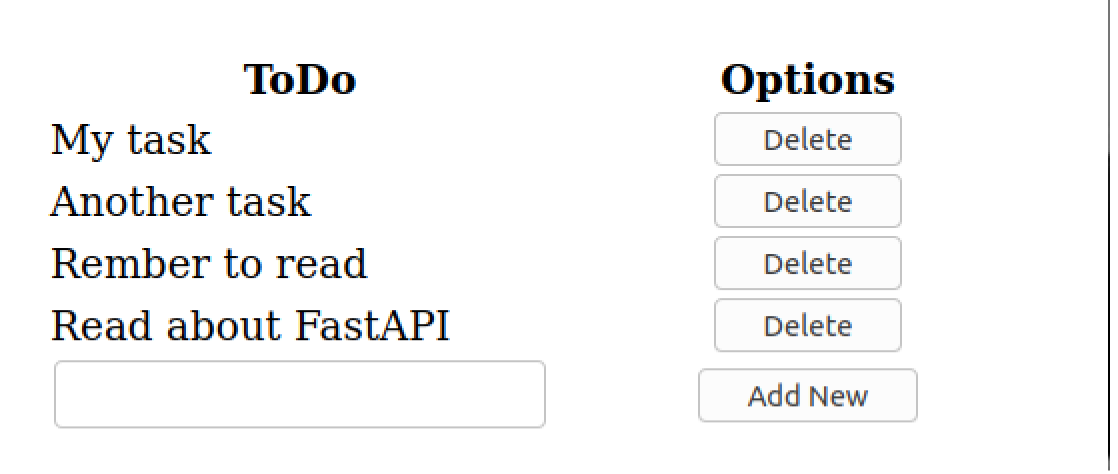
Save the file in the folder **/mytodo/**, same place as the **main.py** file.

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



ToDo	Options
My task	Delete
Another task	Delete
Rember to read	Delete
Read about FastAPI	Delete
	Add New

<div style='page-break-after: always'></div>

Directory

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

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

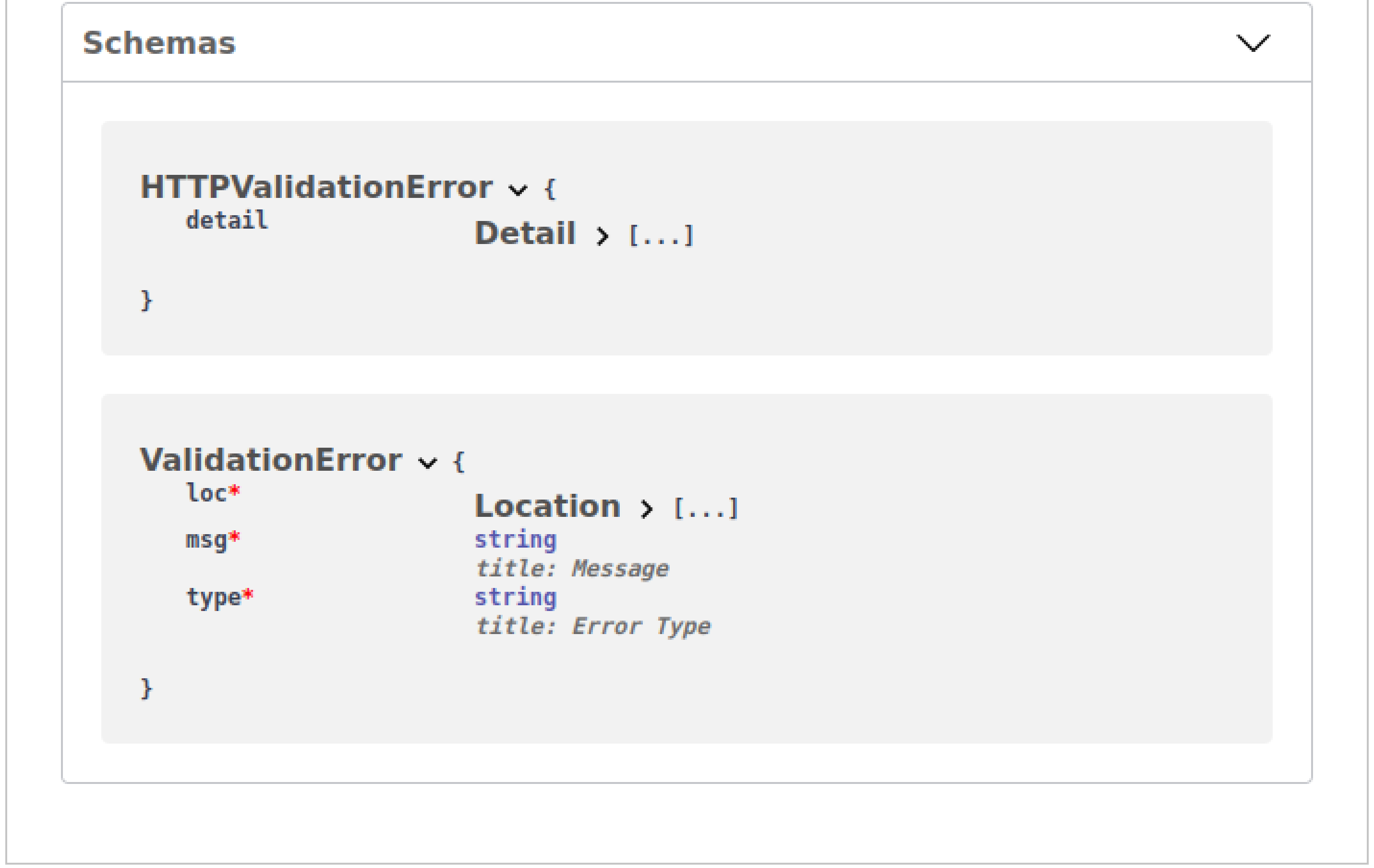
```
{% for id in tododict %}
<tr>
    <td>{{ tododict[id] }}</td>
    <td><a href="/delete/{{ id }}"><button>Delete</button></a></td>
</tr>
{% endfor %}
```

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.



```
HTTPValidationError ⌄ {
    detail                    Detail › [...]
}

ValidationError ⌄ {
    loc*                      Location › [...]
    msg*                      string
                              title: Message
    type*                     string
                              title: Error Type
}
```

<div style='page-break-after: always'></div>

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