Introduction to Web App Development with Python and Flask

Developing and deploying a simple web app in Python

Seyed Parsa Neshaei, PhD Student, EPFL

LauzHack Workshops — April 24, 2024

Before we get in…

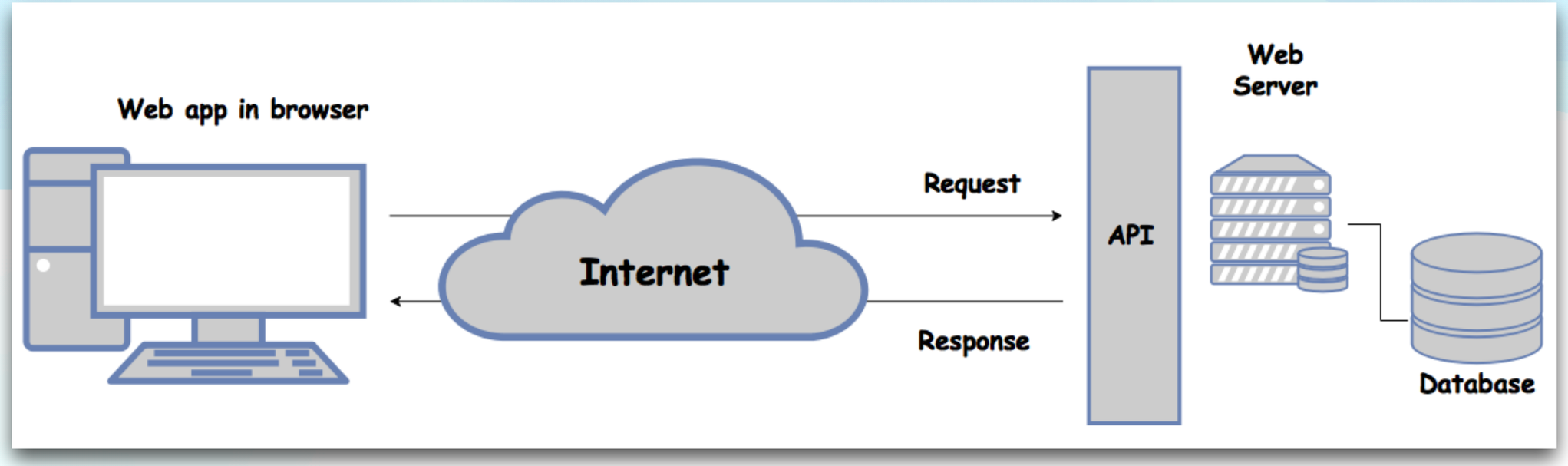
- For beginners in web and API dev (even useful for mobile apps)
- We learn to:
  - Deploy Python code as a service, from *ground up* (we don't use many frameworks for educational purposes)
  - Build a simple todo app from scratch + a bit of styling + saving in database
- There won't be a finished product (to just modify and use), but you learn the building blocks
- We won't cover front-end or DB (SQL) in details

=> Install Postman (or another API Tester app if not on PC/Mac)

=> Get the OpenAI API key

Web API

Front-end, back-end

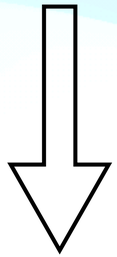


Source: Medium and https://www.analyticsvidhya.com/blog/2022/10/most-commonly-asked-web-api-interview-questions/

Front-end

- Defines how the website appears in the browser
- You “describe” how the website looks in a certain format: HTML
- You can “style” the components of the HTML code using another specific format: CSS
- To write the logic and behavior, you write JS and the browser executes it
  - Most other options convert to JS at the end

```
<h1>Welcome</h1>

<p style="color: red">I hope you
enjoy the workshop!</p>
```



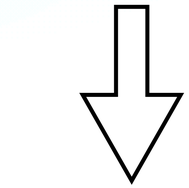
Welcome

I hope you enjoy the workshop!

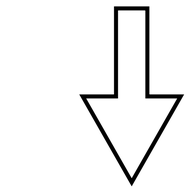
Front-end

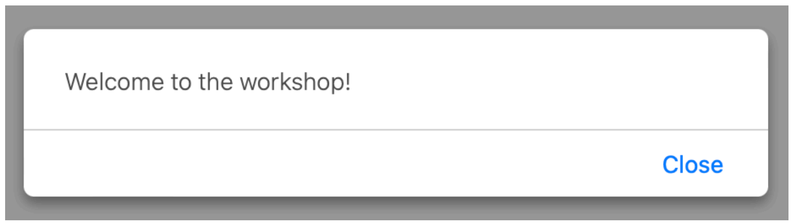
- Defines how the website appears in the browser
- You “describe” how the website looks in a certain format: HTML
- You can “style” the components of the HTML code using another specific format: CSS
- To write the logic and behavior, you write JS and the browser executes it
  - Most other options convert to JS at the end

```
<script>

    function buttonClick() {

        alert('Welcome to the workshop!');

    }

</script>

<button onclick="buttonClick()">Click me!</
button>
```



Click me!





Why not only front-end?

- All front-end code runs locally on the browser (Chrome, Safari, …)
- You have no remote access to any data
- No possible to share information or continue working on different devices
- No data-interactive website, no server
  - The “server” just returns **static** code (HTML/CSS/JS) which is **rendered** in users’ browsers
- We want the same code to serve both browsers and mobile apps

Back-end

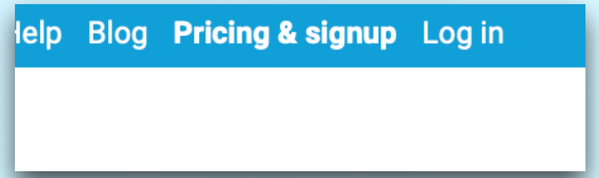
- Is just a piece of code running on a (possibly) always-on server: Python, Go, Node.JS, …
- **Listens** for incoming **requests** from the users (in the form of an **API**)
  - The source of requests can be a browser, a mobile app, …
- Some possible examples
  - WhatsApp / Telegram: send a request to the server (back-end) to retrieve the latest messages
  - TWINT: send a request to the server to send X CHF from A to B
  - Each has its own front-end interface, as well as a server that handles data centrally

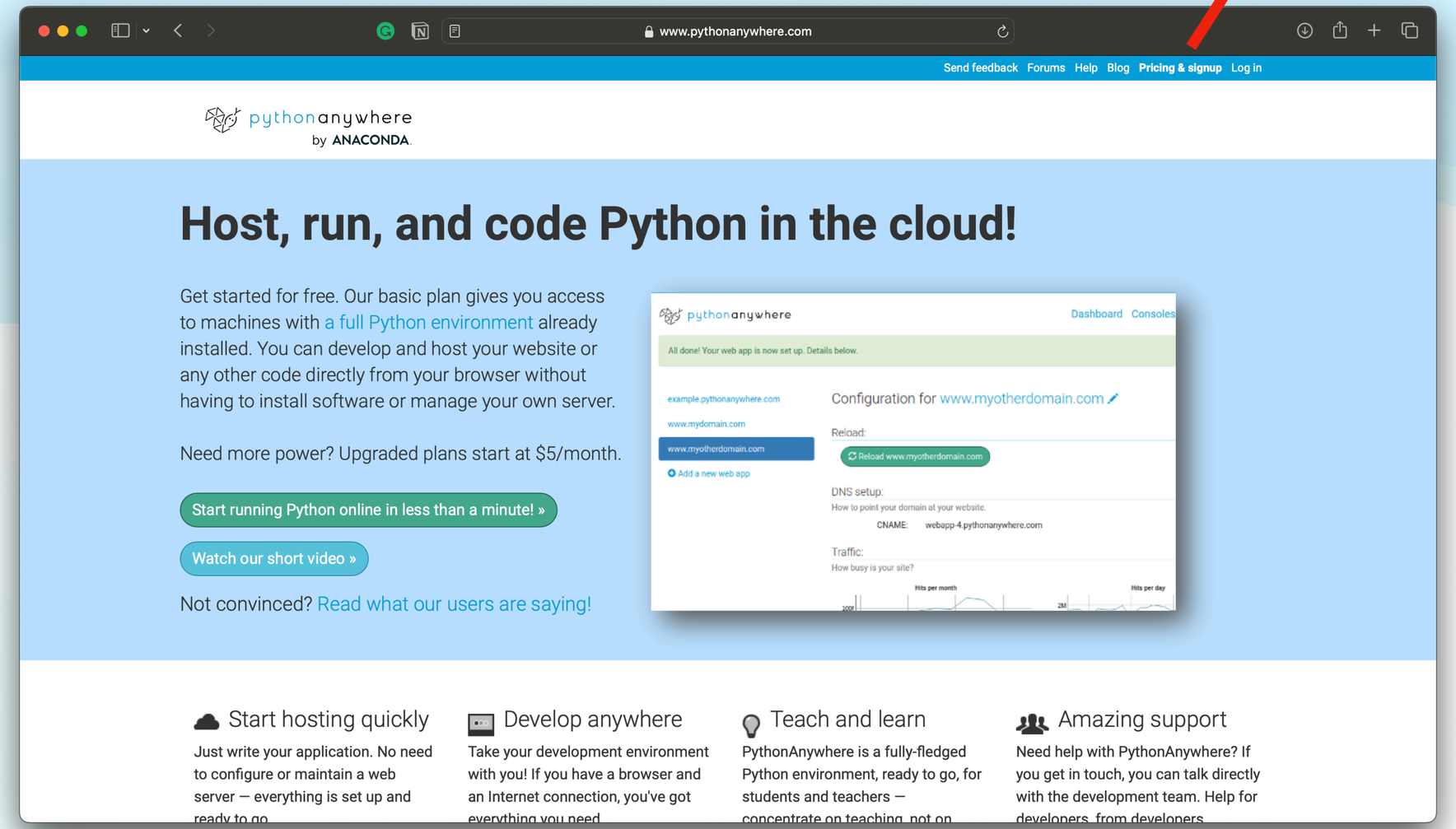
Back-end

- Almost any modern programming language can make a back-end!
- We use Python with the easy-to-use Flask framework
- We need a server: can use PythonAnywhere for free!
  - https://www.pythonanywhere.com/
- Goal: build a todo app: people can add tasks to do for the day
  - What is the front-end and back-end here?

Pythonanywhere

Create an account





Plans and pricing

Beginner: Free!

A limited account with one web app at *your-username*.pythonanywhere.com, restricted outbound Internet access from your apps, low CPU/bandwidth, no IPython/Jupyter notebook support.
It works and it's a great way to get started!

Create a Beginner account

Register with a username (will be a part of the URL) and email

Stay in the browser window to avoid “you are a bot”

Warning You have not confirmed your email address yet. T
anymore, send yourself a new one here.

Dashboard Consoles Files Web Tasks Databases

Add a new web app

Next »

Select a Python Web framework

...or select "Manual configuration" if you want detailed control.

- » **Django**
- » **web2py**
- » **Flask**
- » **Bottle**
- » **Manual configuration** (including virtualenvs)

Flask

What other frameworks should we have here? Send us some feedback using the link at the top of the page!

» **Python 3.10** (Flask 2.1.2)

Next »

Exercises!

1) Change the returning text to be bold. Learn to re-launch the code!

2) Introduce a compilation error. What happens?

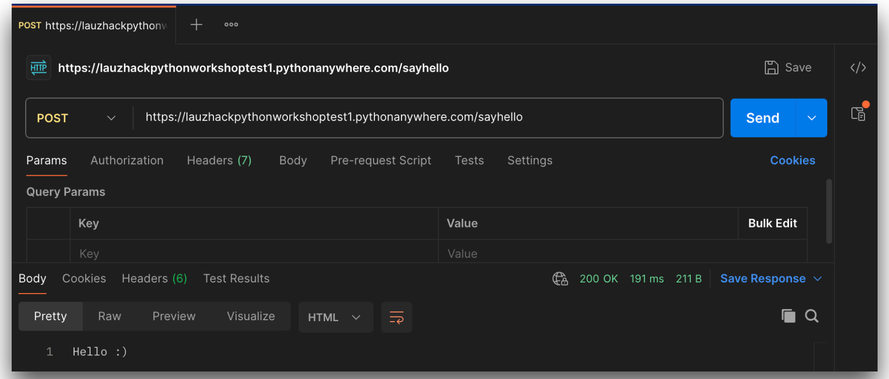
```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello from Flask!'
```

Exercises!

3) Return a HTML file: render_template in the templates folder next to the Python code

4) Try with Postman! See the error code, number of bytes, and time

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/sayhello', methods=['POST'])
def say_hello():
    return 'Hello :)'
@app.route('/')
def hello_world():
    return render_template('test.html')
```



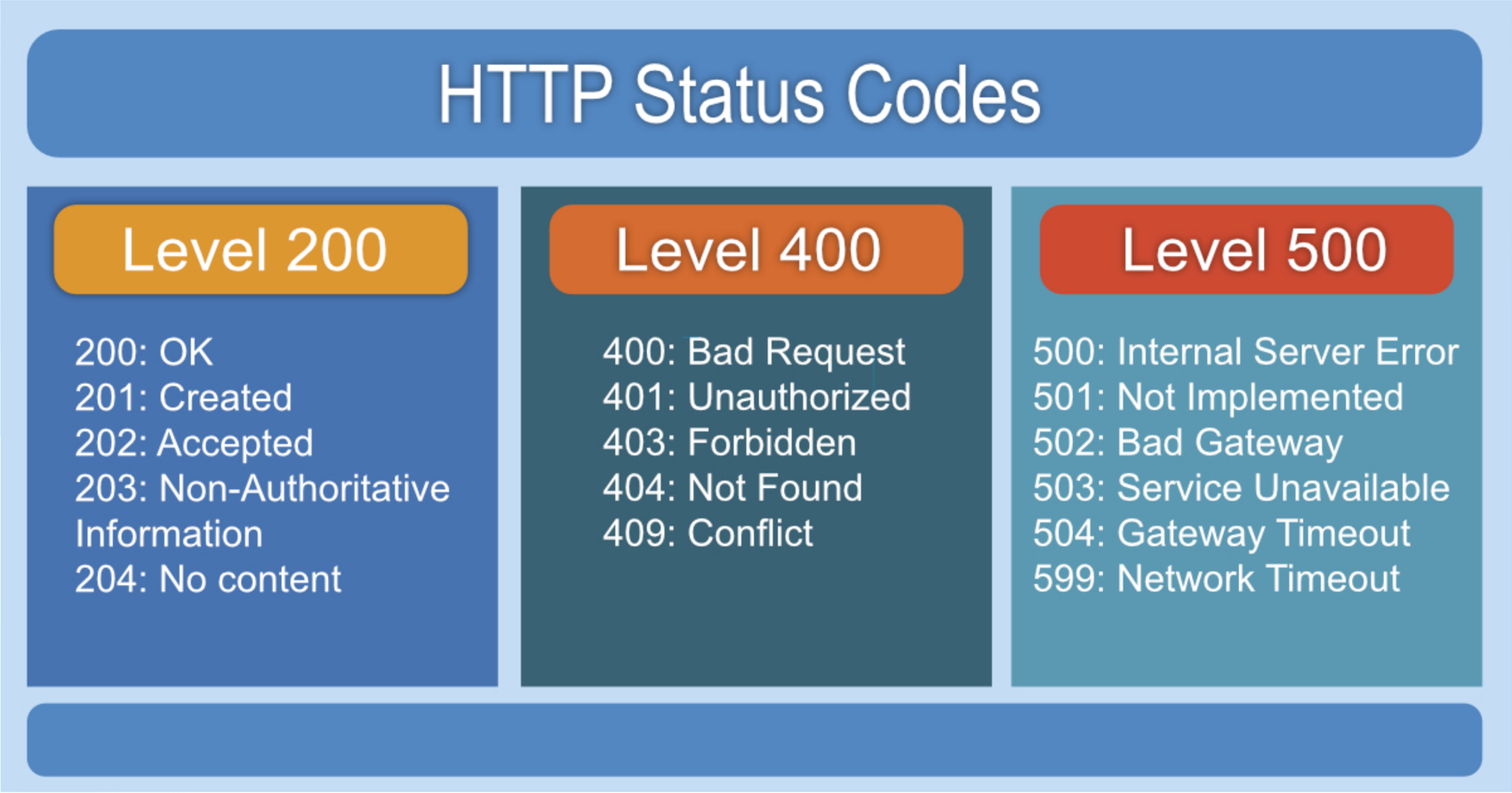
HTTP Methods

HTTP Methods and Their Meaning

Method	Meaning
GET	Read data
POST	Insert data
PUT or PATCH	Update data, or insert if a new id
DELETE	Delete data

Source: https://www.devopsschool.com/blog/understanding-rest-http-method-get-post-put-head-delete/

Error Codes

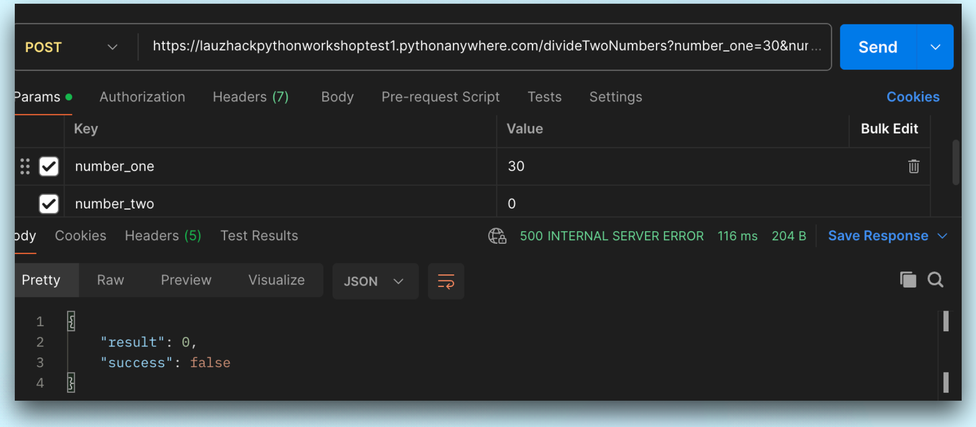


Source: https://restfulapi.net/http-status-codes/

Also see https://http.cat/ !

Exercises!

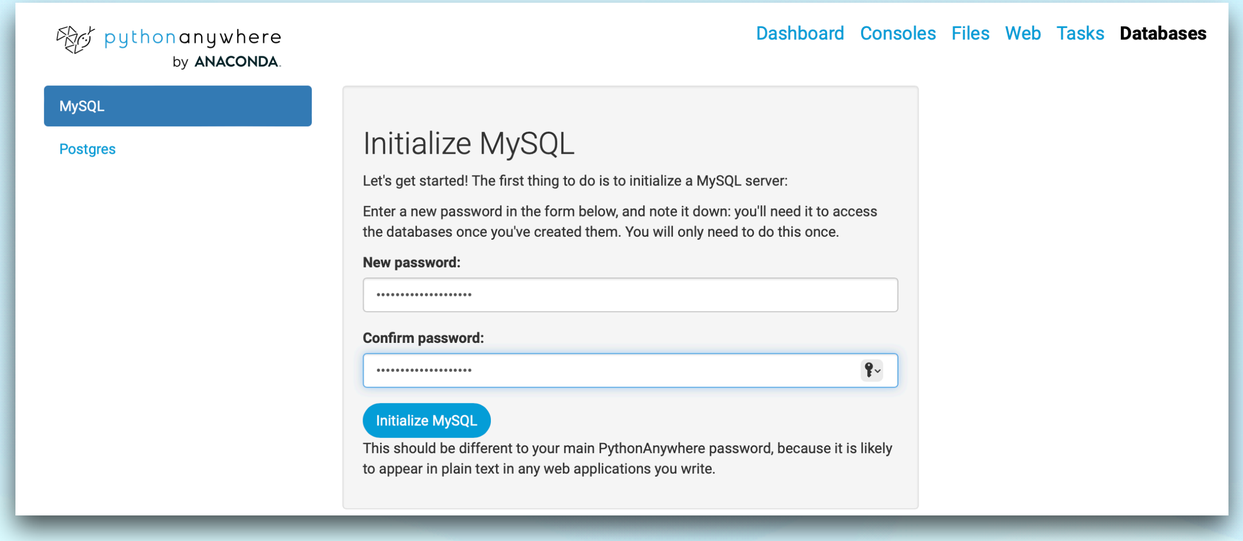
5) Get parameters, return a JSON

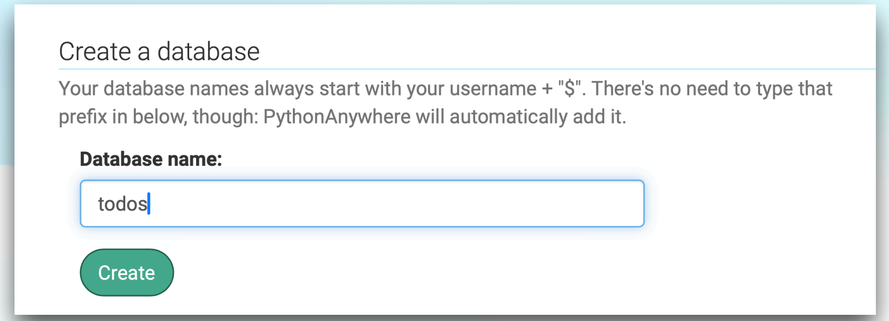


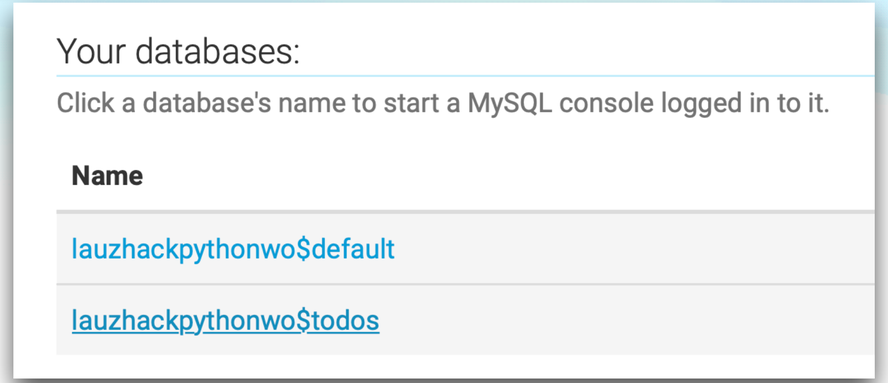

```
from flask import Flask, render_template, request, jsonify
app = Flask(__name__)
@app.route('/divideTwoNumbers', methods=['POST'])
def divide_two_numbers():
    number_one = float(request.args['number_one'])
    number_two = float(request.args['number_two'])
    if number_two == 0:
        return jsonify({"success": False, "result": 0}), 500
    else:
        return jsonify({"success": True, "result": number_one /
number_two}), 200
```

Database

Web Tasks Databases





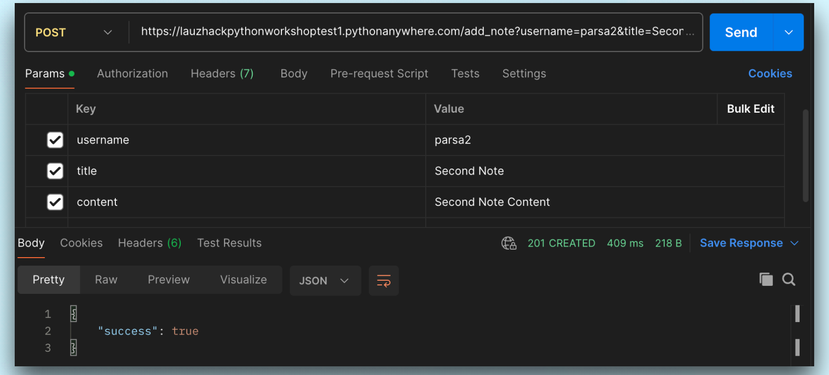



```
mysql> create table TodoNote (id int auto_increment, username text, title text, content text, primary key (id));
Query OK, 0 rows affected (0.02 sec)

mysql> insert into TodoNote (username, title, content) values ('parsa', 'Welcome', 'This is my note body');
Query OK, 1 row affected (0.01 sec)

mysql> select * from TodoNote;
+----+----------+---------+----------------------+
| id | username | title   | content              |
+----+----------+---------+----------------------+
|  1 | parsa    | Welcome | This is my note body |
+----+----------+---------+----------------------+
1 row in set (0.00 sec)
```

SQL...
A future workshop?

Exercises!

6) Add an endpoint for inserting a note



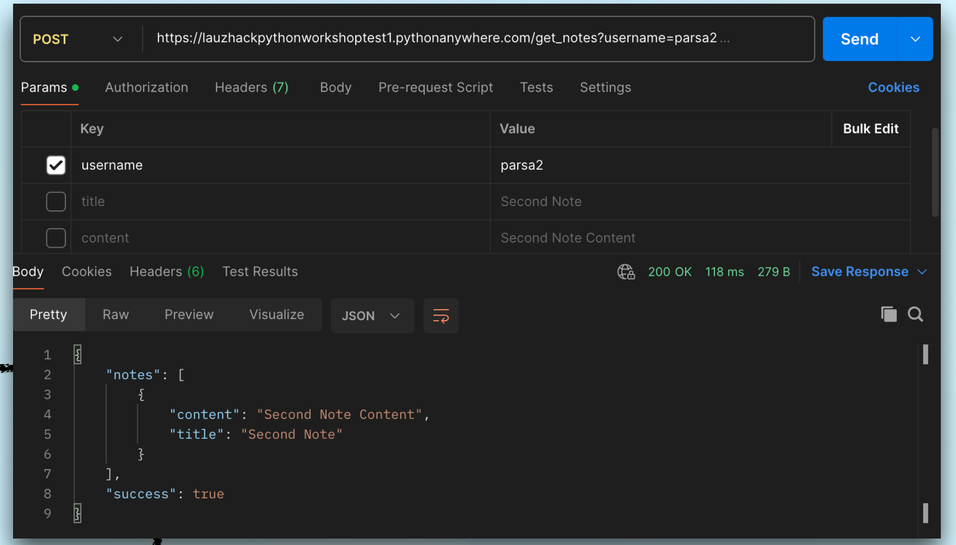

```
from flask import Flask, render_template, request, jsonify
import mysql.connector
db = mysql.connector.connect(host="lauzhackpythonworkshoptest1.mysql.pythonanywhere-services.com",
user="lauzhackpythonwo", password="**********", database="lauzhackpythonwo$todos")
cursor = db.cursor()
app = Flask(__name__)
@app.route('/add_note', methods=['POST'])
def add_note():
    username = str(request.args['username'])
    title = str(request.args['title'])
    content = str(request.args['content'])
    sql_command = "insert into TodoNote (username, title, content) values (%s, %s, %s);"
    values = (username, title, content)
    try:
        cursor.execute(sql_command, values)
        db.commit()
        return jsonify({"success": True}), 201
    except:
        return jsonify({"success": False}), 500
```

```
mysql> select * from TodoNote;
+----+----------+-------------+----------------------+
| id | username | title       | content              |
+----+----------+-------------+----------------------+
|  1 | parsa    | Welcome     | This is my note body |
|  2 | parsa2   | Second Note | Second Note Content  |
+----+----------+-------------+----------------------+
2 rows in set (0.00 sec)
```

Exercises!

7) Return all notes of a user

```
@app.route('/get_notes', methods=['POST'])
def get_notes():
    username = str(request.args['username'])
    sql_command = "select username, title, content from TodoNote
where (username = %s);"
    values = (username,)
    try:
        cursor.execute(sql_command, values)
        results_from_db = cursor.fetchall()
        results_to_return = []
        for item in results_from_db:
            username, title, content = item
            results_to_return.append({"title": title, "content":
content})
        return jsonify({"success": True, "notes":
results_to_return}), 200
    except:
        return jsonify({"success": False, "notes": []}), 500
```

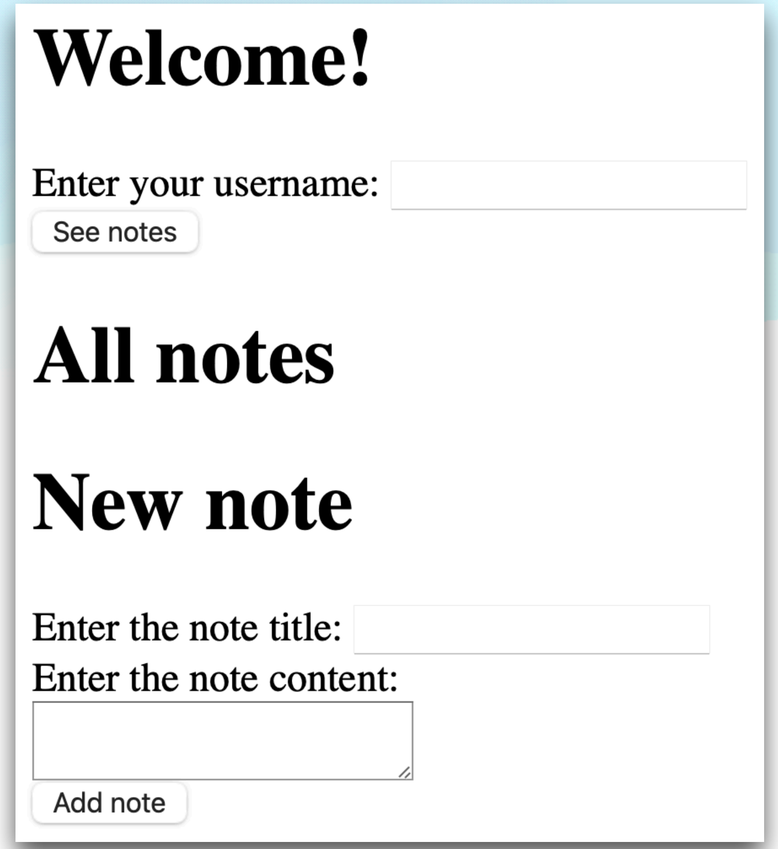


Exercises!

8) Add the front-end UI for seeing and adding notes

```
<html>
    <head>
        <title>The best Todo app!</title>
    </head>
    <body>
        <h1>Welcome!</h1>
        Enter your username: <input type="text" id="username" /> <br />
        <button onclick="seeNotes()">See notes</button>
        <h1>All notes</h1>
        <div id="all-notes"></div>
        <h1>New note</h1>
        Enter the note title: <input type="text" id="username" /> <br />
        Enter the note content: <br />
        <textarea id="note-content"></textarea> <br />
        <button onclick="addNote()">Add note</button>
    </body>
</html>
```

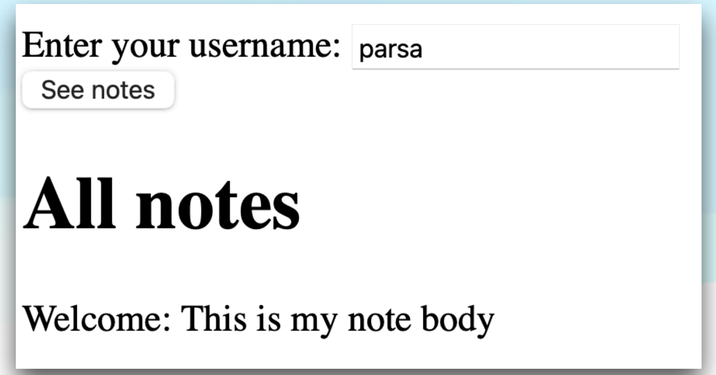
```
@app.route('/')
def main_route():
    return render_template('index.html')
```



Exercises!

9) Send a request to the back-end API to retrieve the notes

```
<script>
    async function seeNotes() {
        let username = document.getElementById("username-field").value;
        let response = await fetch("get_notes?" + new URLSearchParams({
            username: username,
        }), {
            method: 'POST'
        });
        let json = await response.json();
        let outputText = "";
        for (let i = 0; i < json.notes.length; i++) {
            let item = json.notes[i];
            outputText += item.title + ": " + item.content + "<br />";
        }
        document.getElementById("all-notes").innerHTML = outputText;
    }
</script>
```



Many better ways exist to integrate the results of the API in the front-end user interface!

React, Angular, etc.

Exercises!

10) Send a request to add a new note, then check if it is added

```
async function addNote() {
    let username = document.getElementById("username-field").value;
    let title = document.getElementById("note-title-field").value;
    let content = document.getElementById("note-content-field").value;
    let response = await fetch("add_note?" + new URLSearchParams({
        username: username,
        title: title,
        content: content
    }), {
        method: 'POST'
    });
    let status = response.status;
    if (status != 201) alert("Failed");
    else alert("Success");
}
```

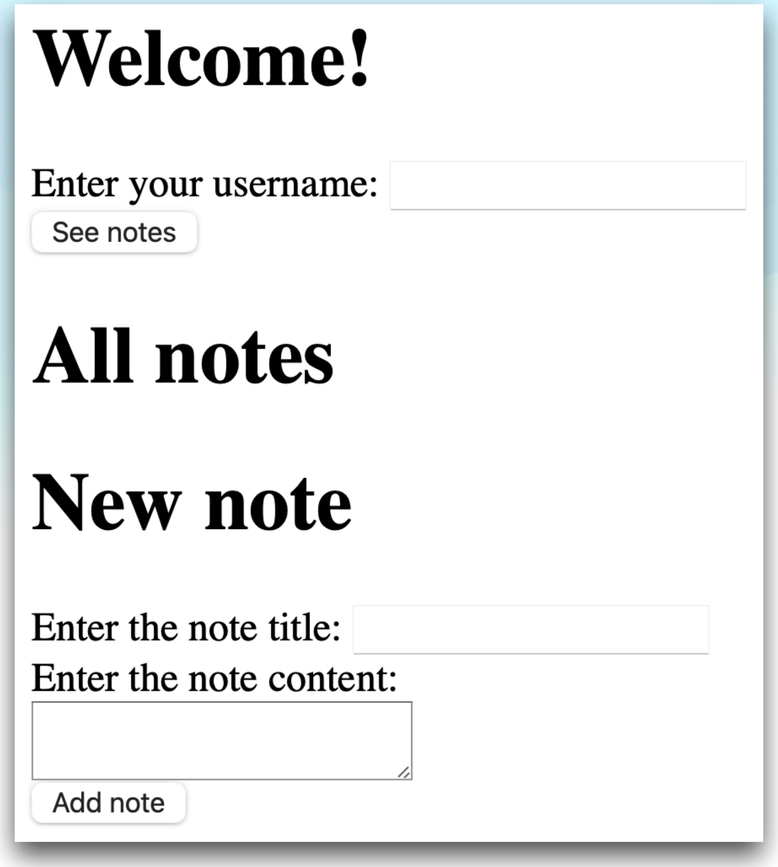
Sending a request, almost similar everywhere!

```swift
func addNote() {
    guard let username = usernameTextField.text, let title = noteTitleTextField.text, let content = noteContentTextField.text,  let url = URL(string: "add_note") else { return }
    var request = URLRequest(url: url)
    request.httpMethod = "POST"
    let parameters = ["username": username, "title": title, "content": content]
    request.httpBody = parameters.percentEncoded()
    let task = URLSession.shared.dataTask(with: request) { data, response, error in
        guard let response = response as? HTTPURLResponse else { return }
        DispatchQueue.main.async {
            if response.statusCode == 201 {
                showAlert(withTitle: "Success", andMessage: "Note added successfully")
            } else {
                showAlert(withTitle: "Failed", andMessage: "Failed to add note")
            }
        }
    }
    task.resume()
}
```

```go
func addNote() {
      username := "username-value"
      title := "note-title-value"
      content := "note-content-value"
      url := "add_note"
      data := map[string]string{"username": username, "title":    title, "content":  content,}
      payload, _ := json.Marshal(data)
      req, err := http.NewRequest("POST", url, bytes.NewBuffer(payload))
      req.Header.Set("Content-Type", "application/json")
      client := &http.Client{}
      resp, err := client.Do(req)
      defer resp.Body.Close()
      if resp.StatusCode == http.StatusCreated {
                                fmt.Println("Success")
      } else {
                                fmt.Println("Failed")
      }
}
```

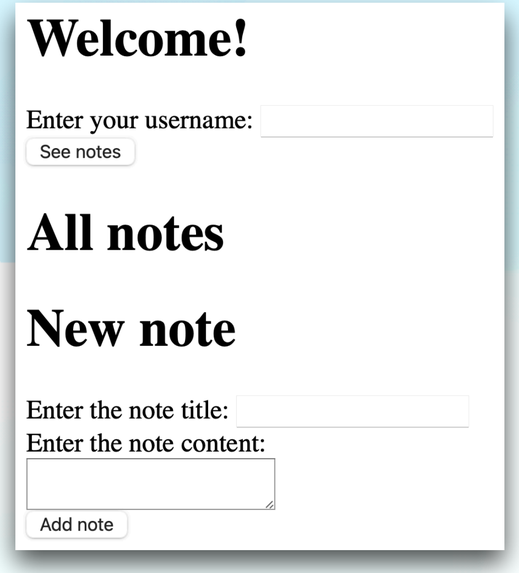
```kotlin
fun addNote() {
    val username = usernameField.text.toString()
    val title = noteTitleField.text.toString()
    val content = noteContentField.text.toString()
    val url = URL("add_note")
    val urlParameters = "username=" + URLEncoder.encode(username, "UTF-8") + "&title=" + URLEncoder.encode(title, "UTF-8") + "&content=" + URLEncoder.encode(content, "UTF-8")
    AddNoteTask().execute(url, urlParameters)
}
class AddNoteTask : AsyncTask<Any, Void, String>() {
    override fun doInBackground(vararg params: Any): String {
        val url = params[0] as URL
        val urlParameters = params[1] as String
        lateinit var connection: HttpURLConnection
        try {
            connection = url.openConnection() as HttpURLConnection
            connection.requestMethod = "POST"
            connection.doOutput = true
            val wr = DataOutputStream(connection.outputStream)
            wr.writeBytes(urlParameters)
            wr.flush()
            wr.close()
            val responseCode = connection.responseCode
            if (responseCode == HttpURLConnection.HTTP_CREATED) {
                return "Success"
            } else {
                return "Failed"
            }
        } catch (e: Exception) {
            Log.e("Error", "Exception: ${e.message}")
            return "Failed"
        } finally {
            connection.disconnect()
        }
    }
```

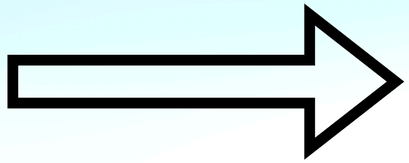
Stop — Front-end is ugly!

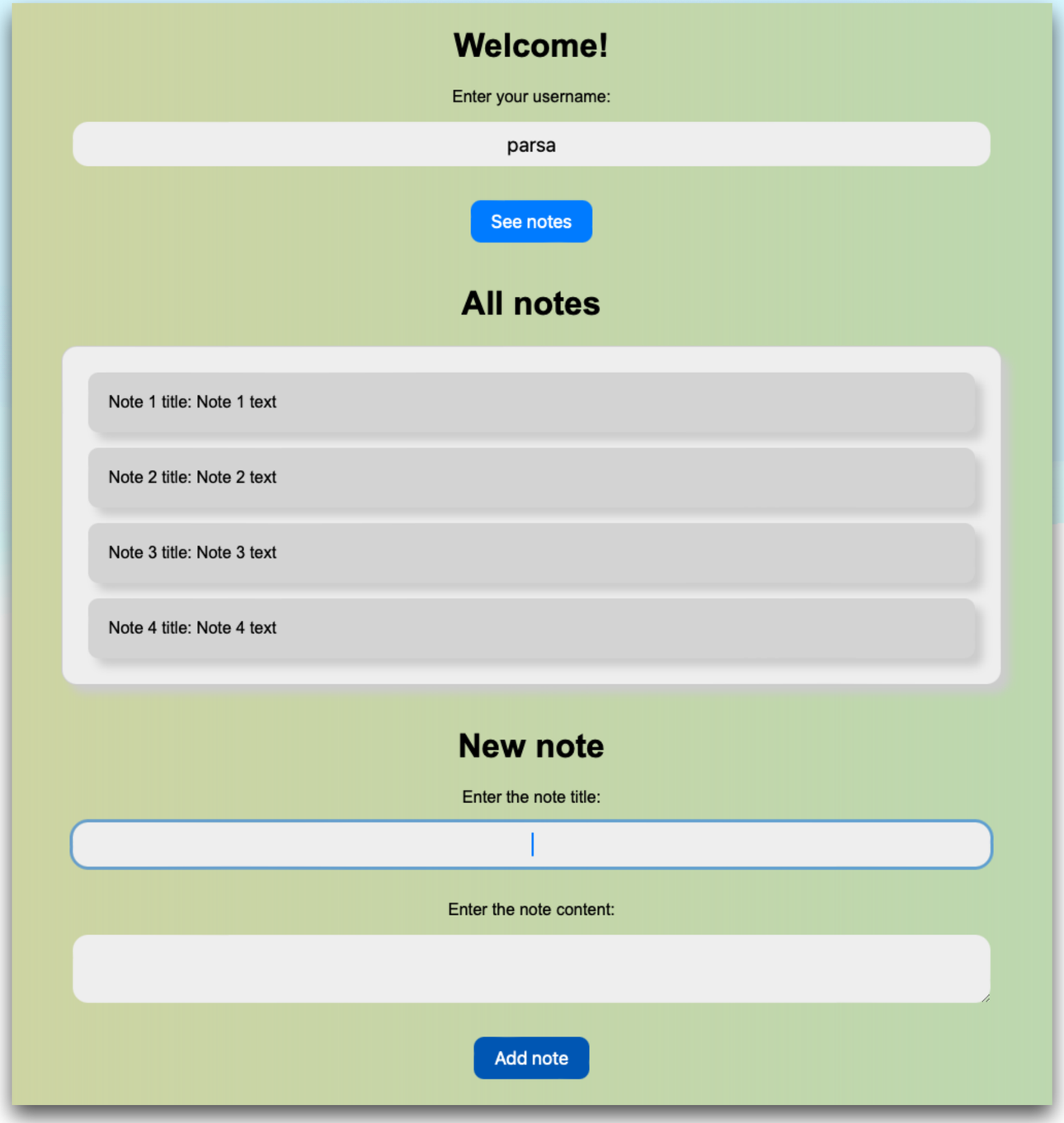


No one will use such an app

What about now?







```
input[type="text"], textarea {
    width: 35%;
    padding: 10px;
    margin: 15px;
    border: none;
    border-radius: 15px;
    background-color: #eeeeee;
    font-size: 120%;
    text-align: center;
}

button {
    padding-top: 10px;
    padding-bottom: 10px;
    padding-left: 20px;
    padding-right: 20px;
    margin-bottom: 20px;
    background-color: #007bff;
    color: #fff;
    border: none;
    border-radius: 10px;
    cursor: pointer;
    font-size: 110%;
}
```

```
body {
    font-family: Arial;
    background: linear-gradient(to right, #dad299, #b0dab9);
    display: flex;
    flex-direction: column;
    align-items: center;
}

button:hover {
    background-color: #0056b3;
}

#all-notes {
    border: 1px solid #ccc;
    width: 35%;
    min-height: 100px;
    border-radius: 15px;
    padding: 10px;
    background-color: #eeeeee;
    margin-bottom: 20px;
}

#all-notes p {
    padding: 20px;
    margin: 15px;
    border-radius: 10px;
    background-color: lightgrey;
}
```

Exercises!

11) Use OpenAI API to summarize notes

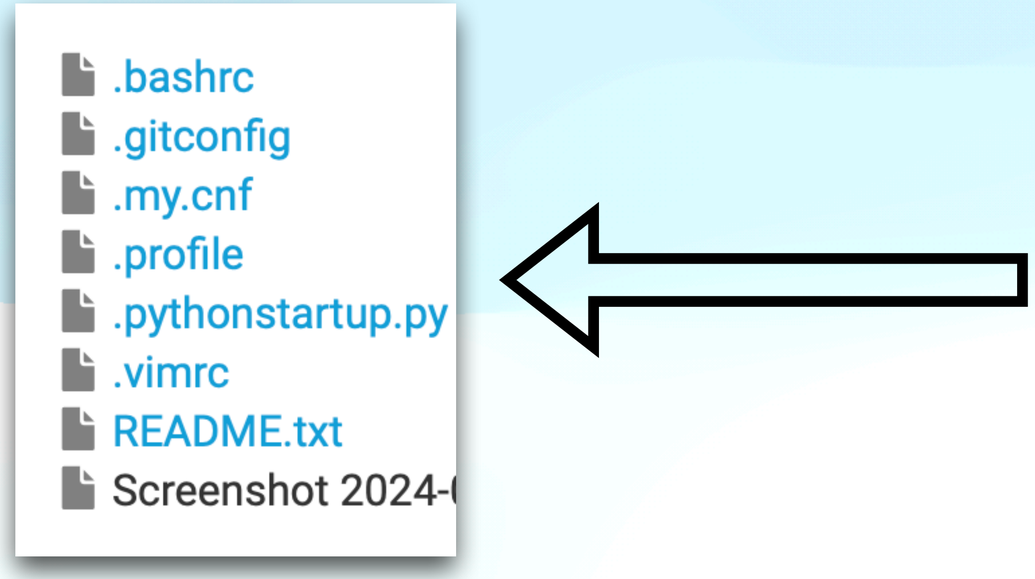
```
!pip install openai
```

```
from openai import OpenAI
client = OpenAI(api_key='XXX')
def askGPTToSummarize(notes):
    user_message = ""
    for note in notes:
        user_message += "Title: " + note['title'] +
"\nContent: " + note['content'] + "\n======\n"
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "Please
summarize the notes of the user."},
            {"role": "user", "content": user_message}
        ],
        temperature=0.5
    )
    return response.choices[0].message.content
```

```
@app.route('/summarize_notes', methods=['POST'])
def summarize_notes():
    username = str(request.args['username'])
    sql_command = "select username, title, content from TodoNote
where (username = %s);"
    values = (username,)
    try:
        cursor.execute(sql_command, values)
        results_from_db = cursor.fetchall()
        notes = []
        for item in results_from_db:
            username, title, content = item
            notes.append({"title": title, "content": content})
        response = askGPTToSummarize(notes)
        return jsonify({"success": True, "summary": response}), 200
    except:
        return jsonify({"success": False, "summary": ""}), 500
```

Exercises!

12) Upload an image




```
@app.route('/upload_photo', methods=['POST'])
def upload_photo():
    try:
        f = request.files['photo']
        f.save(f.filename)
        return jsonify({"success": True}), 200
    except:
        return jsonify({"success": False}), 500
```

```
<br />
<input id="photo-field" type="file" /> <br />
<button onclick="uploadPhoto()">Upload photo</button>
```

```
async function uploadPhoto() {
                let photo = document.getElementById("photo-
field").files[0];
                let formData = new FormData();
                formData.append("photo", photo);
                let response = await fetch("upload_photo", {
                    method: 'POST',
                    body: formData
                });
                let status = response.status;
                if (status != 200) alert("Failed");
                else alert("Success");
            }
```

Exercise — *for you!*

Delete notes

- A text field (`input`) for entering note title
- Confirm with the user: *“Are you sure?”* -> hint: use the ‘`prompt`’ function in JS
- The back-end checks if there is any note with this username and title
  - If no, return 404 and show relevant error in the front-end
  - If yes, delete it and return 200
    - `delete from TodoNote where (username = %s);`
    - Then, fetch the list of notes and update the list automatically in the front-end
- Ask for help if necessary!

Roadmap to be a web developer

https://roadmap.sh/backend

Thanks a lot!

Quiz time :)