

# Εισαγωγή στις τεχνολογίες του διαδικτύου

ECE\_Y210 Εισαγωγή στην Επιστήμη του Ηλεκτρολόγου Μηχανικού - σημειώσεις διάλεξης

Έκδοση 2023

Νικόλαος Αβούρης

## Πίνακας Περιεχομένων

1. Εισαγωγή
2. Το πρωτόκολλο HTTP - Αρχιτεκτονική εφαρμογών
3. Η γλώσσα HTML/CSS
4. Ο εξυπηρετητής Flask/Python
5. Η μηχανή δυναμικών ιστοσελίδων Jinja
6. Παράδειγμα: η εφαρμογή pythonquiz
7. Φιλοξενία της εφαρμογής
8. Εισαγωγή στην Flet και full Python stack
9. Η αρχιτεκτονική της εφαρμογής με Flet/Flask
10. Flet: γραφικά στοιχεία (widgets)
11. Flet: Υποδοχείς (containers)
12. Flet: Δημιουργία στοιχείων από τον χρήστη
13. Flet: Επικοινωνία με τον εξυπηρετητή
14. Επανασχεδίαση της εφαρμογής pythonquiz
15. SQLAlchemy, σύνδεση στη βάση δεδομένων
16. Κρυπτογράφηση μυστικών κωδικών

17. Ο εξυπηρετητής της εφαρμογής Flet/Flask

18. Ανακεφαλαίωση του κύκλου ανάπτυξης της εφαρμογής Flet/Flask

19. Άλλες πηγές

---

# 1. Εισαγωγή

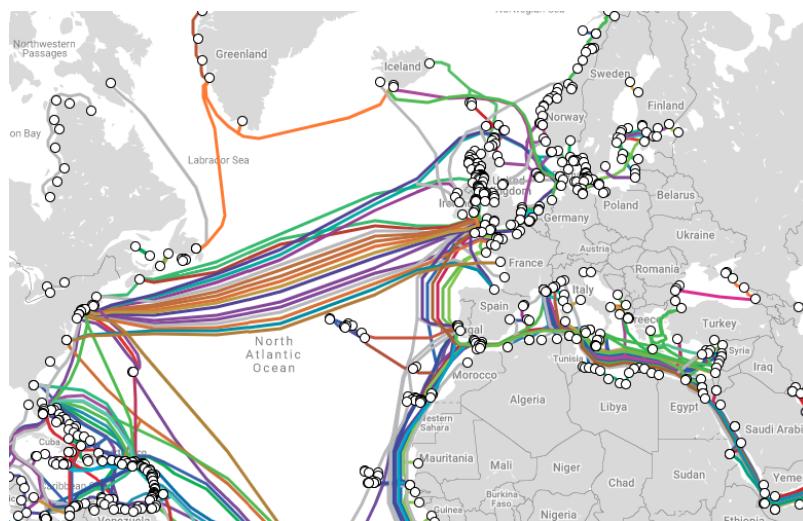
---

Το αντικείμενο αυτής της ενότητας είναι να εξετάσουμε τις βασικές τεχνολογίες που χρησιμοποιούνται στις εφαρμογές του διαδικτύου, και ιδιαίτερα του παγκόσμιου ιστού (web). Αυτό θα γίνει μέσα από ένα παράδειγμα μιας εφαρμογής. Θα γίνουν δύο διαφορετικές υλοποιήσεις της ίδιας εφαρμογής:

- (a) αρχικά με χρήση των κλασσικών τεχνολογιών HTML/CSS/JavaScript στον πελάτη και του framework Flask στον εξυπηρετητή, και στη συνέχεια
- (β) με χρήση του framework Flet/Python στον πελάτη, και ξανά του Flask στον εξυπηρετητή.

Θα παρατηρήσουμε τις διαφορές στη σχεδίαση στις δύο αυτές περιπτώσεις και θα συζητήσουμε τις διάφορες τεχνολογικές επιλογές που είναι διαθέσιμες για ανάπτυξη εφαρμογών παγκόσμιου ιστού, με ιδιαίτερη έμφαση στη γλώσσα προγραμματισμού Python.

Το **διαδίκτυο (internet)** είναι ένα σύνολο από πρωτόκολλα και υποδομές (δίκτυα) που επιτρέπουν τη διασύνδεση υπολογιστών και επικοινωνία μεταξύ εφαρμογών. Η πιο σημαντική εφαρμογή του διαδικτύου είναι ο **παγκόσμιος ιστός (web)**.



*χάρτης από το [submarinecablemap.com](http://submarinecablemap.com)*

Στις επόμενες ενότητες θα δούμε αρχικά το πρωτόκολλο **HTTP** που είναι ο τρόπος επικοινωνίας browser/server, μετά θα περάσουμε στο πρώτο παράδειγμα (a) που περιλαμβάνει υλοποίηση μιας εφαρμογής με χρήση της γλώσσας **HTML/CSS**, για διαμόρφωση της δομής της ιστοσελίδας, καθώς και τη βιβλιοθήκη Flask που μάς επιτρέπει να δρομολογούμε και να ανταποκρινόμαστε σε αιτήματα που έρχονται από τον φυλλομετρητή στον εξυπηρετητή για τη σελίδα αυτή. Επίσης σε αυτό το παράδειγμα, θα δούμε τη γλώσσα **Jinja2** που μάς επιτρέπει να σχεδιάζουμε δυναμικά ιστοσελίδες με παραμετρικό τρόπο, και τέλος θα δούμε πώς μπορούμε να κάνουμε διανομή της εφαρμογής μας σε μια πλατφόρμα φιλοξενίας της. Όλα αυτά θα γίνουν μέσα από ένα παράδειγμα ανάπτυξης μιας εφαρμογής που ελέγχει τις γνώσεις μας στην Python.

Στη συνέχεια μετά την ενότητα 8 θα προχωρήσουμε στο παράδειγμα (β), εκεί θα δούμε το framework **Flet** που είναι υλοποίηση σε Python της γραφικής γλώσσας συγγραφής εφαρμογών στο διαδίκτυο Flutter. Χρησιμοποιώντας την Flet θα δημιουργήσουμε μια εφαρμογή single page (που δεν ξαναφορτώνεται κάθε φορά που ζητάει δεδομένα από τον εξυπηρετητή) και θα σχεδιάσουμε τον αντίστοιχο εξυπηρετητή που ανταποκρίνεται στα αιτήματά της. Με την ευκαιρία αυτής της δεύτερης υλοποίησης της εφαρμογής μας, θα δούμε δύο νέα αλλά πολύ σημαντικά θέματα: την εισαγωγή στις βάσεις δεδομένων, με χρήση της βιβλιοθήκης SQLAlchemy καθώς και στην κρυπτογράφηση των μυστικών κωδικών των χρηστών με χρήση της βιβλιοθήκης werkzeug.security. Τέλος θα συγκρίνουμε τις δύο προσεγγίσεις. Να σημειωθεί ότι σε πρώτη ανάγνωση κάποιος μπορεί να παραλείψει τα κεφάλαια 3 μέχρι 7, και να προχωρήσει κατευθείαν στο 8, όμως θα πρέπει να αναφέρουμε ότι δεν νοείται εισαγωγικό μάθημα στις τεχνολογίες διαδικτύου, χωρίς τις τεχνολογίες HTML/CSS/Javascript.

---

## 2. Το πρωτόκολλο HTTP - Αρχιτεκτονική εφαρμογών

---

Οι εφαρμογές του παγκόσμιου ιστού χρησιμοποιούν το πρωτόκολλο **HTTP (hyper-text transfer protocol)**, για την ανταλλαγή μηνυμάτων μεταξύ ενός **φυλλομετρητή (browser)** και ενός **εξυπηρετητή (web server)** οι οποίοι είναι οι δύο βασικοί κόμβοι των εφαρμογών του παγκόσμιου ιστού.

Το πρωτόκολλο HTTP ορίστηκε πρώτη φορά το 1989 από τον Tim Berners-Lee. Σήμερα είμαστε στην έκδοση HTTP/2. Τα μηνύματα HTTP είναι δομημένα κείμενα, έχουμε δύο κατηγορίες μηνυμάτων:

- Μηνύματα **αιτήματα** (από τον φυλλομετρητή στον εξυπηρετητή): GET ή POST συνήθως
- Μηνύματα **αποκρίσεις** (από τον εξυπηρετητή στον φυλλομετρητή).

Ακολουθεί ένα πρόγραμμα Python, το οποίο χρησιμοποιεί τη βιβλιοθήκη **requests** και μάς επιτρέπει να εισαγάγουμε μια διεύθυνση URL και στη συνέχεια κάνει ένα αίτημα HTTP, λαμβάνει την απόκριση, και μάς επιστρέφει την κεφαλίδα της απόκρισης HTTP που λαμβάνουμε.

```
# πρόγραμμα που ζητάει ένα url και επιστρέφει την κεφαλίδα της
# απόκρισης HTTP
import requests

while True:
    url = input('url:')
    if url == '': break
    if not url.startswith('http'):
        url = 'http://'+ url
    try:
        with requests.get(url) as response:
            print("\nRESPONSE STATUS: ", response.status_code)
            print("RESPONSE HEADER")
            for key, value in response.headers.items():
                print("{:30s} {}".format(key, value))
    except:
        print('error opening', url)
```

(ο κώδικας)[example1-http.py] Ένα παράδειγμα χρήσης:

```
url:http://hci.ece.upatras.gr/
RESPONSE STATUS: 200
```

RESPONSE HEADER	
Date	Sat, 27 Mar 2021 12:30:29 GMT
Server	Apache/2.4.18 (Ubuntu)
Set-Cookie	language=en; expires=Sun, 27-Mar-2022 12:30:29 GMT; Max-Age=31536000; path=/
Link	; rel="https://api.w.org/"
Vary	Accept-Encoding
Content-Encoding	gzip
Content-Length	9848
Keep-Alive	timeout=5, max=100
Connection	Keep-Alive
Content-Type	text/html; charset=UTF-8

Πειραματιστείτε με το πρόγραμμα αυτό στέλνοντας μηνύματα σε διάφορες διευθύνσεις του διαδικτύου, δέστε προσεκτικά την κεφαλίδα της απόκρισης που πήρατε, μπορείτε να βγάλετε συμπεράσματα για το τι τύπου είναι ο εξυπηρετητής που σάς απάντησε (είναι η κεφαλίδα **Server**), τι τύπου είναι η απόκριση, (**Content-Type**), αν επισυνάπτονται αρχεία cookies και τι διάρκεια ζωής έχουν (**Set-Cookie**), κλπ. )

Επίσης σκεφτείτε με ποιον τρόπο θα μπορούσατε να χρονομετρήσετε το συνολικό χρόνο από την αποστολή του αιτήματος μέχρι την παραλαβή της απόκρισης (χρήση της **time.process\_time()**)

### Συνηθισμένοι κωδικοί σε μηνύματα απόκρισης HTTP:

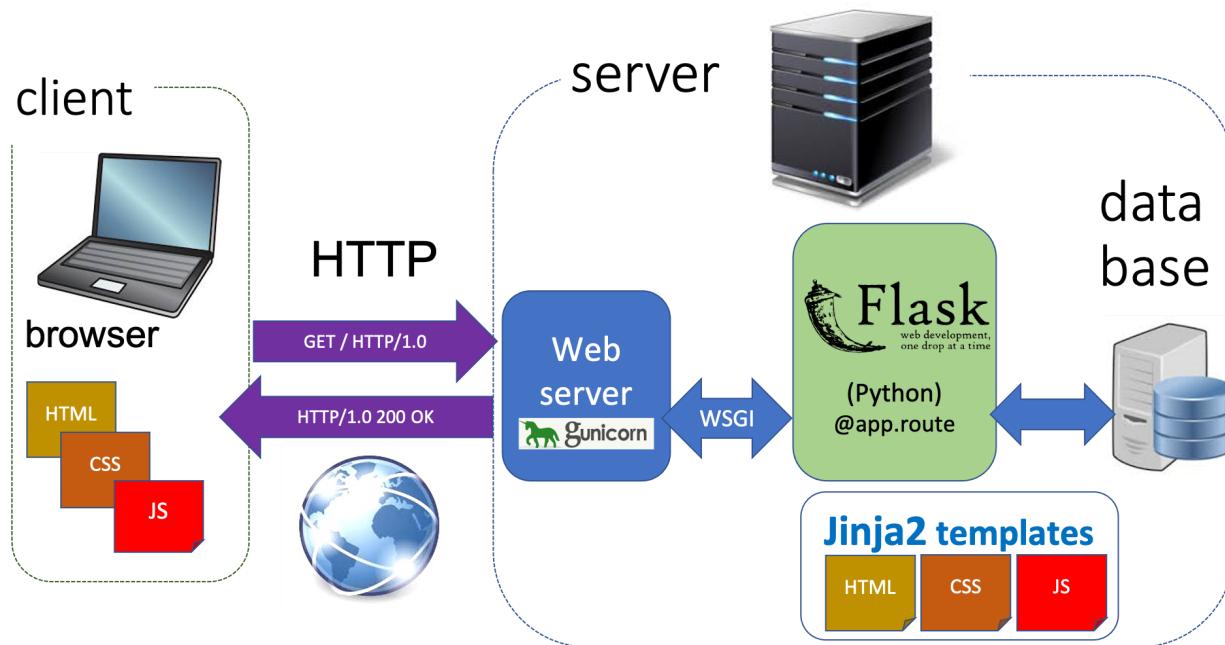
Στο προηγούμενο παράδειγμα παρατηρήσαμε ότι το Response status ήταν 200, αυτό σημαίνει ότι το μήνυμα μας το επεξεργάστηκε ο εξυπηρετητής με επιτυχία, ωστόσο έχει ενδιαφέρον να διερευνήσουμε τι άλλους κωδικούς κατάστασης της απόκρισης μπορεί να λάβουμε και ποια η σημασία των κωδικών αυτών.

Παρακάτω ακολουθεί ένας συνοπτικός πίνακας.

- 1xx: μήνυμα πληροφορίας
- 2xx: κατάσταση επιτυχούς επεξεργασίας
- 3xx: προώθηση του Client σε διαφορετικό URL
- 4xx: σφάλμα προερχόμενο από τον Client
- 5xx: σφάλμα προερχόμενο από τον Server

### Αρχιτεκτονική εφαρμογών διαδικτύου

Κλείνοντας την εισαγωγική ενότητα θα δώσουμε μια συνοπτική εικόνα της αρχιτεκτονικής μιας εφαρμογής του διαδικτύου με βάση και τις τεχνολογίες που θα χρησιμοποιήσουμε στο παράδειγμα (a) στις επόμενες ενότητες.



Αρχιτεκτονική παράδειγμα (a)

Από την εικόνα αυτή παρατηρούμε πόσο σύνθετη είναι η αρχιτεκτονική μιας εφαρμογής διαδικτύου.

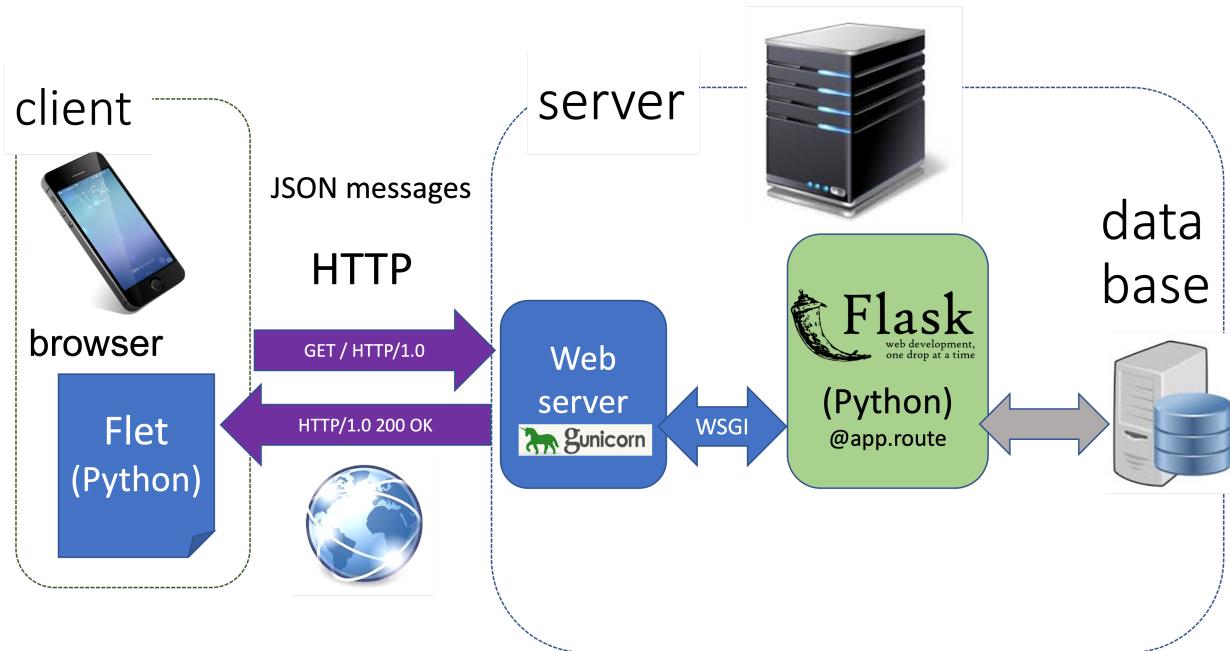
Τα κύρια στοιχεία είναι ένας εξυπηρετητής (web server) που στο παράδειγμά μας θα χρησιμοποιήσουμε ένα τμήμα της βιβλιοθήκης Flask, όταν όμως φιλοξενήσουμε την εφαρμογή μας σε δημόσιο χώρο φιλοξενίας (Heroku) θα χρησιμοποιήσουμε τη βιβλιοθήκη Gunicorn της Python. Ένα άλλο στοιχείο της αρχιτεκτονικής είναι το πρωτόκολλο επικοινωνίας μεταξύ του εξυπηρετητή και της εφαρμογής μας που είναι το WSGI. Η ίδια η εφαρμογή που θα αναπτύξουμε θα γίνει με χρήση της βιβλιοθήκης Flask, ενώ το περιεχόμενο των σελίδων που θα στείλουμε στον πελάτη θα είναι γραμμένο σε HTML/CSS/JavaScript.

Αυτό είναι το πιο κλασσικό παράδειγμα εφαρμογής διαδικτύου, μάλιστα θα μπορούσαμε να αντικαταστήσουμε το framework Flask/Python με άλλες τεχνολογίες. Μερικές δημοφιλείς αναφέρονται στη συνέχεια:

framework	Γλώσσα προγραμματισμού	logo
express	Javascript	
laravel	PHP	
spring	Java	

Στον πελάτη (φυλλομετρητή) οι τεχνολογίες είναι πιο σταθερές, αν και πρόσφατα προστέθηκαν νέες δυνατότητες, που με χρήση της τεχνολογίας web assembly μπορεί να τρέξει κώδικας διαφόρων γλωσσών προγραμματισμού και στον φυλλομετρητή. Μια τέτοια σχετικά πρόσφατη τεχνολογία παρέχει το framework **Flet** που επιτρέπει τη χρήση της Python στον browser.

Σε αυτή την περίπτωση η αρχιτεκτονική της εφαρμογής φαίνεται στο επόμενο σχήμα:



Αρχιτεκτονική παράδειγμα (β)

Αν συγκρίνουμε τα δύο σχήματα, παρατηρούμε ότι το (β) είναι πολύ πιο απλό, για έναν μάλιστα προγραμματιστή Python παρέχει το πλεονέκτημα χρήσης της γλώσσας και στις δύο πλευρές. Δεν χρειάζονται οι τεχνολογίες HTML, CSS, JavaScript, Jinja2 και όπως θα δούμε η εφαρμογή Flask είναι πολύ πιο απλή. Βεβαίως και εδώ υπάρχουν μειονεκτήματα, αφού η Flet είναι μια τεχνολογία με πολύ λιγότερες δυνατότητες από αυτές της HTML/CSS/JavaScript, λιγότερο γνωστή άρα υπάρχουν πολύ λιγότερες πρόσθετες βιβλιοθήκες, ενώ δεν υποστηρίζουν όλοι οι φυλλομετρητές αυτή τη νέα τεχνολογία.

Στην επόμενη ενότητα θα εστιάσουμε σε μια σύντομη παρουσίαση των τεχνολογιών συγγραφής και μορφοποίησης ιστοσελίδων του παραδείγματος (α). Αν κάποιος όμως θα προτιμούσε να χρησιμοποιήσει κατ-ευθείαν τη Flet μπορεί να παραλείψει την επόμενη ενότητα.

### 3. Η γλώσσα συγγραφής ιστοσελίδων HTML / γλώσσα μορφοποίησης CSS

Η HTML είναι γλώσσα επισημείωσης περιεχομένου (markup) για την οργάνωση μιας ιστοσελίδας, με τη γλώσσα αυτή ορίζουμε το περιεχόμενο και τη δομή μιας ιστοσελίδας.

Ας δούμε στη συνέχεια τη δομή ενός τυπικού αρχείου HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ο τίτλος</title>
    <style>
      body {
        font-family: sans-serif;
        background-color: yellow;
      }
    </style>
  </head>
  <body>
    <h1>Κεφαλίδα 1</h1>
    <p> Πρώτη παράγραφος.</p>
  </body>
</html>
```

Το αρχείο αυτό παράγει το παρακάτω αποτέλεσμα:



Όπως παρατηρούμε το αρχείο μετά την αρχική δήλωση στην πρώτη γραμμή που λέει στον φυλλομετρητή ότι είναι έγγραφο HTML5, στη συνέχει είχουμε μια ιεραρχία από στοιχεία με ρίζα το στοιχείο `<html>` και παιδιά του τα στοιχεία `<head>` και `<body>`, που και αυτά με τη σειρά τους περιέχουν άλλα στοιχεία.

Ένα στοιχείο HTML έχει την εξής δομή:

```
<etiketa παράμετρος="τιμή"> περιεχόμενο </etiketa>
```

ένα παράδειγμα:

```
<p id="p1"> καλημέρα </p>
```

Στη συνέχεια περιγράφονται οι βασικές ετικέτες με τις οποίες μπορούμε να δημιουργήσουμε στοιχεία HTML.

## Τυπικές ετικέτες

- `<html>, <head>, <body>` ετικέτες δόμησης του εγγράφου
- `<meta>` μετα-δεδομένα στην κεφαλίδα του εγγράφου
- `<style>` εισαγωγή κανόνων μορφοποίησης (CSS), σε επόμενη ενότητα θα περιγράψουμε τη σύνταξή τους.
- `<script>` εισαγωγή κώδικα JavaScript
- `<h1>...</h1>, <h2>, ... <h6>` Επικεφαλίδες διαφόρων επιπέδων
- `<ul> ...</ul>` Μη διατεταγμένη λίστα
- `<ol> ... </ol>` Διατεταγμένη λίστα (1.,2.,3. ...)
- `<pre>` κείμενο που διατηρεί τη στοίχισή του (pre-formatted), χρήσιμο για κώδικα.
- `<p> ... </p>` Παράγραφος (block element)
- `<div> ... </div>` Ορισμός μπλοκ, που αρχίζει από νέα γραμμή και αλλαγή γραμμής στο τέλος
- `<span> ... </span>` ορισμός μικρής περιοχής κειμένου (inline element)
- `<form>` φόρμα εισαγωγής στοιχείων από τον χρήστη
- `<input>` στοιχείο φόρμας
- `<table>` πίνακας
- `<a>` (άγκυρα) υπερσύνδεσμος
- `<img>` εικόνα
- `<video>` βίντεο
- `<audio>` ήχος

Ένα στοιχείο HTML μπορεί να φέρει μοναδική **ταυτότητα** που ορίζεται με το γνώρισμα `id` ή να ανήκει σε μια **κλάση**, την οποία μοιράζονται διάφορα στοιχεία, η κλάση ορίζεται με το γνώρισμα `class`.

Ένα παράδειγμα ορισμού της ταυτότητας ενός στοιχείου `<div>`.

```
<div id="myid"> κείμενο </div>
```

Οι ταυτότητες `id` και οι κλάσεις `class` χρησιμοποιούνται για την συσχέτιση των στοιχείων με κανόνες εμφάνισης του στοιχείου CSS.

## Ένα παράδειγμα

Έστω ότι θέλουμε να ζητήσουμε από τον χρήστη να διαλέξει μια από τις πιθανές απαντήσεις στο παρακάτω ερώτημα:

Τι θα τυπώσει το παρακάτω πρόγραμμα;

```
def f():
    a = 7
    print(a, end = ' ')
a = 5
f()
print(a, end = ' ')
```

Απάντηση (επιλέξτε):

- 5 5
- 7 5
- 7 7
- NameError
- δεν γνωρίζω

Υποβολή

Ο κώδικας HTML που ακολουθεί, υλοποιεί την παραπάνω συμπεριφορά:

```
<form>
<div id="quiz">
    Τι θα τυπώσει το παρακάτω πρόγραμμα;
    <pre>
def f():
    a = 7
    print(a, end = ' ')
a = 5
f()
print(a, end = ' ')
    </pre>
    Απάντηση (επιλέξτε):
    </div>
    <ol>
        <li><input type="radio" name="answer" value="1"> 5 5</li>
        <li><input type="radio" name="answer" value="2"> 7 5</li>
        <li><input type="radio" name="answer" value="3"> 7 7</li>
        <li><input type="radio" name="answer" value="4">
            NameError</li>
        <li><input type="radio" name="answer" value="5"> δεν
```

```
γνωρίζω</li>
</ol>
<button>Υποβολή</button>
</form>
```

(Την πλήρη λύση θα βρείτε στο αρχείο [example2-question.html](#))

Να σημειωθεί ότι δημιουργήσαμε μια φόρμα `<form>` μέσα στην οποία τοποθετήσαμε κείμενο και κώδικα (στοιχείο `<pre>`), καθώς και μια σειρά από στοιχεία `<input type="radio">` με κοινό `name`, όταν ο χρήστης επιλέξει ένα από αυτά και υποβάλουμε την φόρμα, θα επιστρέψει την αντίστοιχη τιμή `value`.

Περισσότερες οδηγίες για τη σύνταξη της HTML μπορείτε να βρείτε στο [w3 HTML Tutorial](#)

**Άσκηση:** Πώς θα μπορούσατε να αλλάξετε τον κώδικα ώστε το πρόγραμμα να επιστρέφει αντί για `answer`, μια μεταβλητή `score`, να βαθμολογεί με 1 τη σωστή απάντηση, -1 τη λάθος και 0 την απάντηση "δεν γνωρίζω"

## Κανόνες CSS

Όπως είδαμε στον αρχικό μας παράδειγμα της HTML για να εμφανιστεί το κείμενο με γραμματοσειρά "sans-serif" (χωρίς πατούρες, έντονες άκρες) και με κίτρινο υπόβαθρο είχαμε ενσωματώσει στο `<head>`, ένα στοιχείο `<style>` που περιείχε κανόνες CSS. Ένας κανόνας CSS ορίζει την εμφάνιση των στοιχείων HTML της ιστοσελίδας.

Για παράδειγμα αν επιθυμούμε τα στοιχεία `<h1>` και `<p>` να περιέχουν κείμενο χρώματος `blue` με εσωτερικό περιθώριο `1rem` ο σχετικός κανόνας CSS θα έχει την εξής μορφή:

```
h1, p {
  color: blue;
  padding: 1rem;
}
```

Στο παράδειγμα αυτό οι `h1`, `p` λέγονται **επιλογείς**, οι `color`, `padding` είναι οι **ιδιότητες** και `blue`, `1rem` είναι αντίστοιχα οι τιμές των ιδιοτήτων αυτών. Να σημειωθεί εδώ ότι συνήθως `1rem=16px`.

Οι επιλογείς μπορεί να αφορούν κλάσεις (πχ `<ετικέτα class="classname">`) ή τις ιδιότητες στοιχείων πχ `<ετικέτα id="ταυτότητα">`

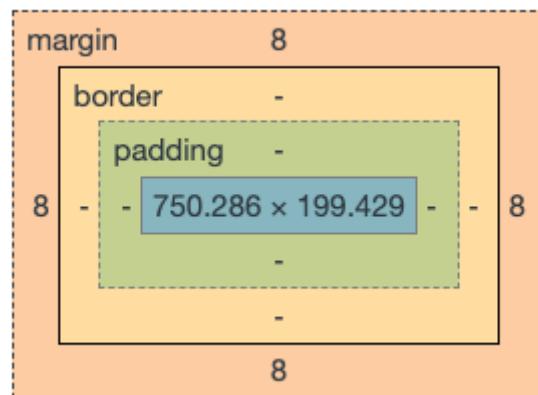
Για παράδειγμα ο κανόνας

```
#myid {font-size: 2rem;}
```

ορίζει ότι το μέγεθος χαρακτήρων του στοιχείου με ταυτότητα `id="myid"` είναι 2 rem (root em).

Οι κανονές CSS μπορεί να περιέχονται σε ένα εξωτερικό αρχείο .css ή μέσα στο ίδιο το αρχείο .html μέσα στην ετικέτα `<style>`.

Κάθε στοιχείο HTML εμφανίζεται ως ένα ορθογώνιο κουτί. Το κουτί αυτό αποτελείται από 4 περιοχές: περιεχόμενο, εσωτερικό περιθώριο (padding), περίγραμμα (border), εξωτερικό περιθώριο (margin). Στο παράδειγμα (από τα εργαλεία developer του chrome) ένα στοιχείο έχει διαστάσεις περιεχομένου 750x200px και εξωτερικό περιθώριο 8px.



Περισσότερες οδηγίες για τη σύνταξη κανόνων CSS μπορείτε να βρείτε στο [web3 CSS tutorial](#)

**Άσκηση:** Πώς θα μπορούσατε να αλλάξετε τον κώδικα της άσκησης ώστε το χρώμα υποβάθρου του κώδικα να γίνει #d5ecc2 και της σελίδας #eeebdd, καθώς και να υπάρχει περιθώριο ανάμεσα στην άκρη του παράθυρου και το περιεχόμενο.

---

## 4. Κατασκευάζουμε έναν εξυπηρετητή με Python/Flask

Στις προηγούμενες ενότητες είδαμε τις βασικές τεχνολογίες που χρησιμοποιούνται για να δημιουργήσουμε μια ιστοσελίδα HTML/CSS (Δεν αναφερθήκαμε στην τρίτη βασική τεχνολογία που είναι η JavaScript, περισσότερα για αυτή στο [web3 JavaScript tutorial](#)).

Ήρθε η ώρα να δημιουργήσουμε έναν εξυπηρετητή ο οποίος θα ανταποκρίνεται σε αιτήματα και θα αποστέλλει προς τον πελάτη ιστοσελίδες όπως αυτή με το quiz του παραδείγματος που είδαμε. Το πρόγραμμα που θα φτιάξουμε δεν θα μάς στέλνει μόνο ερωτήσεις, αλλά θα παίρνει και την απάντησή μας και θα υπολογίζει το σκορ μας.

Ένας εξυπηρετητής του παγκόσμιου ιστού μπορεί να γραφτεί σε διάφορες γλώσσες προγραμματισμού, εδώ θα δοκιμάσουμε τη γλώσσα Python και τη βιβλιοθήκη [Flask](#).

Πρέπει αρχικά να εγκαταστήσουμε την Flask και να δημιουργήσουμε ένα απλό πρόγραμμα με βάση και τις οδηγίες που θα βρούμε στο [Flask tutorial](#)

### Εγκατάσταση της Flask

Δημιουργούμε τον φάκελο (quiz-server) στον οποίο θα δημιουργήσουμε τον εξυπηρετητή μας [server.py](#)

Από το terminal του vs code, αρχικά δημιουργούμε ένα virtual environment με την εντολή (windows):

```
$ py -3 -m venv venv
```

(mac)

```
$ python3 -m venv venv
```

Στη συνέχεια ενεργοποιούμε αυτό το virtual environment (εικονικό περιβάλλον) που δημιουργήσαμε

(windows)

```
venv\Scripts\activate
```

**σημείωση:** σε εκδόσεις των windows πιθανόν στο terminal (powershell) να μην μάς επιτρέψει την ενεργοποίηση του εικονικού περιβάλλοντος με κάποιο μήνυμα του τύπου: [Activate.ps1 is not digitally signed. You cannot run this script.](#). Για να αντιμετωπίσουμε το πρόβλημα σύμφωνα με [υπόδειξη](#) πρέπει πρώτα να δώσετε την εντολή:

```
Set-ExecutionPolicy Unrestricted -Scope CurrentUser -Force
```

(mac)

```
$ . venv/bin/activate
```

Η απόκριση είναι:

```
(venv) nma@nmac quiz-server %
```

Αυτό είναι ένδειξη ότι έχει ενεργοποιηθεί το virtual environment στο οποίο μέσα θα εγκαταστήσουμε τη Flask.

Το virtual environment είναι ένας προστατευμένος χώρος της Python που δεν επηρεάζεται από άλλες βιβλιοθήκες που μπορεί να έχουμε εγκαταστήσει στον υπολογιστή μας, ή νεότερες εγκαταστάσεις της γλώσσας ή των εξαρτημάτων.

Είναι γενικά καλή πρακτική σύνθετες εφαρμογές όπως ο εξυπηρετητής Flask να εγκαθίστανται σε εικονικά περιβάλλοντα.

Η εγκατάσταση της Flask γίνεται με το εργαλείο `pip`

```
$ pip install Flask
```

Η εγκατάσταση της Flask, περιλαμβάνει μια σειρά εξαρτήματα, όπως την Werkzeug που υλοποιεί το πρωτόκολλο WSGI σύνδεσης ενός προγράμματος Python με ένα webserver, την Jinja2 που είναι μια βιβλιοθήκη template για εισαγωγή παραμετρικά περιεχομένου σε μια ιστοσελίδα, την click, που επιτρέπει την επικοινωνία με τη flask από γραμμή εντολών, την MarkupSafe για έλεγχο ασφάλειας στο περιεχόμενο που εισάγουμε στις ιστοσελίδες.

Στη συνέχεια θα δημιουργήσουμε την πιο απλή εφαρμογή εξυπηρετητή με τη Flask. Θα αντιγράψουμε το παράδειγμα από την [ιστοσελίδα της Flask](#) σε ένα αρχείο `server.py`:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return '<h1>καλή σας μέρα!</h1>'
```

Οι δύο πρώτες γραμμές δημιουργούν ένα στιγμιότυπο της εφαρμογής Flask. Η εφαρμογή Flask επικοινωνεί με τον εξυπηρετητή ιστού (web server) με το πρωτόκολλο WSGI, τον στάνταρ τρόπο που έχει ένα πρόγραμμα Python να επικοινωνεί με ένα εξυπηρετητή ιστού. Ο δημιουργός του στιγμιότυπου Flask() δέχεται ως παράμετρο το κυρίως πρόγραμμα της εφαρμογής που είναι το όνομα του ίδιου του αρχείου `__name__`.

Η κύρια λειτουργία της εφαρμογής είναι η δρομολόγηση αιτημάτων που φτάνουν στον εξυπηρετητή. Τα αιτήματα φτάνουν σε URL, σε διευθύνσεις του ιστού. Η εφαρμογή μας δημιουργεί ένα χάρτη από διευθύνσεις αιτημάτων και του αντίστοιχου κώδικα που θα ανταποκριθεί σε κάθε ένα αίτημα.

Η πρώτη μας εφαρμογή έχει μόνο ένα τέτοιο σημείο δρομολόγησης. Τα αιτήματα που φτάνουν στη σχετική διεύθυνση "/" τα χειρίζεται η συνάρτηση `hello_world()`, η οποία όταν κληθεί επιστρέφει μια συμβολοσειρά.

Αν ο εξυπηρετητής ήταν συνδεδεμένος για παράδειγμα στη διεύθυνση "www.server.com", κάθε φορά που έφτανε ένα αίτημα στη διεύθυνση αυτή θα το εξυπηρετούσε η αντίστοιχη συνάρτηση.

Αν είχαμε ένα ακόμη σημείο δρομολόγησης "/login", αυτό θα αφορούσε αιτήματα που φτάνουν στη διεύθυνση "www.server.com/login".

Στο τέλος του προγράμματος προσθέτουμε την εντολή που ενεργοποιεί την εφαρμογή Flask και μάλιστα σε debug κατάσταση:

```
if __name__ == "__main__":
    app.run(debug=True)
```

Τώρα είμαστε έτοιμοι να τρέξουμε την εφαρμογή μας και αυτό θα πρέπει να γίνει από τη γραμμή εντολών (terminal):

```
python server.py
```

Η εφαρμογή Flask μάς γνωστοποιεί ότι είναι ενεργή και έτοιμη πλέον να εξυπηρετήσει αιτήματα:

```
* Serving Flask app "server" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a
  production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 318-838-524
```

Να σημειωθεί ότι μαζί με την Flask έχουμε κατεβάσει ένα web server που τρέχει σε τοπικό επίπεδο.

Αν πάρουμε μια απάντηση όπως η παραπάνω, αυτό σημαίνει ότι ο εξυπηρετητής είναι διαθέσιμος και μπορούμε να συνδεθούμε σε αυτόν από τον φυλλομετρητή μας πληκτρολογώντας τη διεύθυνση `localhost:5000`

`localhost` είναι το συμβολικό όνομα για τον τοπικό υπολογιστή (127.0.0.1) ενώ η θύρα στην οποία θα πρέπει να συνδεθούμε από τον φυλλομετρητή ορίζεται η 5000.

**Σημείωση:** συνήθως η θύρα που εξυπηρετούνται εφαρμογές του παγκόσμιου ιστού είναι η θύρα 80, και η οποία παραλείπεται από τις διευθύνσεις σε συνήθη χρήση, αφού εννοείται, εδώ χρησιμοποιούμε μια άλλη θύρα για να μην διαταράξουμε τη συνήθη χρήση του φυλλομετρητή μας.

Αν όλα πάνε καλά όταν ανοίξουμε τον φυλλομετρητή στη διεύθυνση localhost:5000, θα δούμε το μήνυμα "καλή σας μέρα!" που μάς έστειλε ο εξυπηρετητής.

## Εξυπηρέτηση ιστοσελίδας

Αν θέλουμε να πάρουμε μια ιστοσελίδα, πχ το αρχείο `question.html` αντί για απλά το κείμενο της καλημέρας, όπως στο προηγούμενο παράδειγμα, αυτό θα πρέπει να γίνει αλλάζοντας το σημείο δρομολόγησης (route) ώστε να κληθεί η ιστοσελίδα με την μέθοδο `render_template()` της Flask, ως εξής:

```
import render_template from Flask
...
@app.route('/')
def quiz():
    return render_template('question.html')
```

Αυτό προϋποθέτει ότι έχουμε δημιουργήσει ένα φάκελο `templates` και εκεί έχουμε αποθηκεύσει το αρχείο `question.html`.

Όταν τώρα στον φυλλομετρητή δώσουμε τη διεύθυνση `localhost:5000` θα δούμε τη γνωστή φόρμα του τεστ, που είχαμε δει σε προηγούμενη ενότητα, όμως τώρα μάς την έστειλε η εφαρμογή Flask.

## Παραμετρικά σημεία δρομολόγησης

Επίσης είναι δυνατόν να έχουμε παραμετρικά σημεία δρομολόγησης, όπως στο παράδειγμα:

```
@app.route('/user/<name>')
def user(name):
    return "Καλή σου μέρα {}".format(name)
```

Στην περίπτωση αυτή ανάλογα με την τιμή του URL έχουμε διαφορετικό μήνυμα στον χρήστη.

Όταν φορτώσουμε τη σελίδα `localhost:500/user/Niko` θα πάρουμε στον φυλλομετρητή μας το μήνυμα

Καλή σου μέρα Niko!

Αυτή η δυνατότητα είναι ιδιαίτερα χρήσιμη για να μεταφέρουμε πληροφορία στον εξυπηρετητή από τον πελάτη, μέσω του URL.

---

## 5. Η μηχανή template Jinja2

Το αρχείο `question.html` που στείλαμε στο προηγούμενο παράδειγμα ήταν απλά ένα στατικό ήδη έτοιμο αρχείο, γραμμένο στη γλώσσα HTML/CSS.

Θα ήταν καλό όμως να μπορούσαμε να αλλάζαμε το περιεχόμενό του, ώστε με το ίδιο αρχείο να στέλνουμε διαφορετικές ερωτήσεις του τεστ.

Αυτό μπορεί να γίνει με χρήση της μηχανής template **Jinja2** που περιλαμβάνεται στο Flask.

Αν υποθέσουμε ότι επιθυμούμε να εισάγουμε το όνομα του χρήστη (μεταβλητή `name` στο αρχείο, αυτό γίνεται εισάγοντας τη σχετική μεταβλητή στο αρχείο `html` ως ακολούθως:

```
<h1>Python-Quiz – Χρήστης: {{user_name }}</h1>
```

και καλώντας τη μέθοδο σχεδίασης της σελίδας, ως εξής:

```
return render_template('question.html', user_name=name)
```

Δηλαδή έχουμε ορίσει μια μεταβλητή `Jinja2`, την `user_name`, στην οποία όταν καλούμε το `template` περνάμε μια τιμή, αυτή της μεταβλητής `name` της Python. Μέσα στο `template` εισάγουμε την μεταβλητή μέσα σε διπλά άγκιστρα:

```
{{ user_name }}
```

Η μηχανή `Jinja2` κατά την κλήση της `render_template()` μεταφέρει την τιμή στη θέση αυτή και αποστέλλει στον χρήστη τη σελίδα με το νέο περιεχόμενο.

Σε μια μεταβλητή `Jinja2` μπορούμε να εφαρμόσουμε φίλτρα, ως εξής: `{} | filter`

Μερικά τέτοια φίλτρα είναι: `lower` (μικρά γράμματα), `upper` (κεφαλαία), `safe` (εκτέλεση του κώδικα HTML). Το τελευταίο φίλτρο είναι ιδιαίτερα χρήσιμο για την ενσωμάτωση κώδικα `html` ο οποίος θα θέλαμε να εκτελεστεί, ώστε για παράδειγμα να μπορούμε να ενσωματώσουμε τον κώδικα `python` σε ετικέτες `<pre>`, `</pre>` με συνέπεια ο κώδικας να εμφανιστεί με στοίχιση, ώστε να είναι κατανοητός.

Επίσης υπάρχουν άλλες δομές στην `Jinja2`, όπως η δομή επανάληψης `for` για επαναληπτική παρουσίαση δεδομένων ενός πίνακα. Αν έχουμε ένα πίνακα `replies`, μπορούμε να παρουσιάσουμε τα στοιχεία του ως λίστα ως εξής:

```
<ol>
{% for reply in replies %}
    <li>{{ reply }}</li>
{% endfor %}
</ol>
```

Με αυτόν τον τρόπο θα παρουσιάσουμε τις απαντήσεις σε ένα ερώτημα πολλαπλής επιλογής, όπως θα δούμε σε μια επόμενη ενότητα.

Μια άλλη χρήσιμη δομή είναι η δομή υπό-συνθήκη εκτέλεσης ενός κώδικα (συνθήκη if), ακολουθεί ένα παράδειγμα.

```
{% if user %}
    Γειά σου, {{ user }}!
{% else %}
    Καλημέρα σας!
{% endif %}
```

## Κληρονομικότητα Jinja2 templates

Η μηχανή template Jinja2 επιτρέπει την κληρονομικότητα μεταξύ templates. Αυτή η ιδιότητα είναι ιδιαίτερα χρήσιμη αν θέλουμε κάποια στοιχεία να τα επαναλάβουμε σε όλες τις ιστοσελίδες, χωρίς να αντιγράψουμε τον ίδιο κώδικα σε πολλά αρχεία, κάτι που μπορεί να είναι πηγή σφαλμάτων, αν για παράδειγμα θέλουμε να κάνουμε αλλαγές.

Ο μηχανισμός κληρονομικότητας εξηγείται στη συνέχεια με ένα παράδειγμα. Έστω ότι έχουμε ένα βασικό αρχείο, το οποίο ονομάζουμε **base.html** και στο οποίο ενσωματώνουμε τα κοινά στοιχεία όλων των ιστοσελίδων, πχ το τμήμα **<head>**, ενώ το τμήμα **<body>** αλλάζει σε διαφορετικές ιστοσελίδες.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title> {% block title %}{% endblock title %} </title>
    <style>
        ....
    </style>
</head>
<body>
```

```
{% block body %}  
  {% endblock body %}  
</body>  
</html>
```

Το αρχείο αυτό σε δύο σημεία έχει μπλοκ κώδικα που είναι παραμετρικός, στον τίτλο και στο σώμα του αρχείου html.

Όλα τα υπόλοιπα στοιχεία πρόκειται να κληρονομηθούν σε όλες τις ιστοσελίδες που θα περιλάβουν το **base.html**.

Να ένα παράδειγμα μιας σελίδας login που κληρονομεί το αρχείο **base.html**.

```
{% extends "base.html" %}  
{% block title %} Login {% endblock title %}  
{% block body %}  
  <h1> Καλωσήρθατε στο Python-Quiz</h1>  
  <form>  
    <input type="text" name="name" placeholder="το όνομά σας..."/>  
    <button>Ξεκινήστε</button>  
  </form>  
{% endblock body %}
```

## 6. Παράδειγμα: η εφαρμογή **pythonquiz**

Έχουμε δημιουργήσει μια εφαρμογή python για αυτο-αξιολόγηση γνώσεων μέσω ερωτήσεων πολλαπλής επιλογής. Η εφαρμογή αρχικά λειτουργεί από γραμμή εντολών (πχ από το IDLE).

### Η εφαρμογή python

Όταν ξεκινήσει η εφαρμογή ρωτάει τον χρήστη το όνομά του, τον αναζητάει στη βάση δεδομένων και αν τον βρει του λέει το σκορ που είχε πετύχει σε προηγούμενες προσπάθειες, και στη συνέχεια του κάνει 5 τυχαίες ερωτήσεις από μια βάση δεδομένων ερωτήσεων.

Ένα παράδειγμα χρήσης της εφαρμογής είναι:

```
Δώσε το όνομά σου:kostas
Παίκτης: kostas
Προηγούμενες προσπάθειες:
28-03-21 Sun 17:39:47    score: 100.0%

Ερώτηση 1 από 5
Ερώτηση [Q51]
Ποιο από τα παρακάτω στοιχεία δεν μπορεί να είναι κλειδί ενός λεξικού:
Απαντήσεις:
1  (1,2)
2  [1,2]
3  (2,)
4  2
5  2.0
6  '2'
7. Δεν γνωρίζω
Η απάντησή σας (x για έξοδο):3
λάθος απάντηση, η σωστή απάντηση είναι η 2
Το score είναι -0.2
```

Αν η απάντηση του χρήστη είναι σωστή, αυτός παίρνει 1 μονάδα, αν είναι εσφαλμένη βαθμολογείται με αρνητική βαθμολογία  $-1/(n-1)$ , όπου  $n$  το πλήθος των απαντήσεων, και βαθμολογείται με 0 αν επιλέξει την τελευταία απάντηση που είναι πάντα "Δεν γνωρίζω".

Ο σκελετός της αρχιτεκτονικής της εφαρμογής φαίνεται στη συνέχεια.

```
import re
import os
import random
```

```

import datetime

total_questions = 5
class Quiz:
    allQuiz = {}
    def __init__(self, id, question, replies, correct):
        ...
    def calculate_score(self, reply):
        ...
    def __str__(self):
        ...

class Player:
    players = {}
    @staticmethod
    def load_players():
        ...
    def __init__(self, name, update=True):
        ...
    def new_game(self, score, update=True):
        ...
    def update_players(self):
        ...
    def __str__(self):
        ...

def load_quiz():
    ...
    ## load players
    Player.load_players()

def play_quiz():
    ...
    if __name__ == "__main__":
        load_quiz()
        play_quiz()

```

Η εφαρμογή συνδέεται με μια βάση δεδομένων, που για απλότητα έχει υλοποιηθεί ως δύο αρχεία κειμένου τα οποία περιέχουν το ένα ([questions.txt](#)) τις ερωτήσεις του τεστ, και το άλλο ([players.txt](#)) τα ιστορικά στοιχεία προηγούμενων προσπαθειών των παικτών.

Ξεκινάει με φόρτωμα των ερωτήσεων του quiz και των παικτών (συνάρτηση [load\\_quiz\(\)](#)). Στη συνέχεια καλείται η συνάρτηση [play\\_game\(\)](#), η οποία είναι ο βασικός ελεγκτής της ροής του παιχνιδιού, κάνει επιλογή τυχαίων ερωτήσεων και εμφανίζει τις ερωτήσεις, δέχεται την απάντηση του χρήστη, και υπολογίζει το σκορ.

Μπορείτε να πειραματιστείτε με την εφαρμογή αυτή.

Το ζητούμενο είναι να μεταφέρουμε την εφαρμογή στο διαδίκτυο. Δηλαδή ο χρήστης να επικοινωνεί με την εφαρμογή μέσω του φυλλομετρητή.

## Δημιουργία διεπαφής με την εφαρμογή

Κατ' αρχάς δημιουργούμε στο αρχείο `pythonquiz.py` συναρτήσεις διεπαφής με τη βάση δεδομένων, αυτές θα μπορέσουμε να τις καλέσουμε αν εισάγουμε το αρχείο στην εφαρμογή Flask με την εντολή: `import pythonquiz as quiz`.

Οι χρήσιμες συναρτήσεις της διεπαφής είναι οι εξής:

```
quiz.load_quiz()
# φορτώνει τα δεδομένα των ερωτήσεων για να
# τα έχει διαθέσιμα στην εφαρμογή, καλείται
# στην αρχή.

quiz.draw_questions()
# επιστρέφει array με κωδικούς 5 τυχαίων ερωτήσεων

quiz.show_question(id)
# για συγκεκριμένο κωδικό ερώτησης id
# επιστρέφει dictionary με τα εξής στοιχεία:
# {id": "", "question": "", "replies":
# {1: "", 2:"", ...}, "correct": int }

quiz.question_score(id, reply):
# υπολογίζει το σκορ για συγκεκριμένη επιλογή του χρήστη

quiz.player_history(name):
# επιστρέφει το σκορ από προηγούμενες
# προσπάθειες του χρήστη `name`

quiz.save_game(name, score):
# αποθηκεύει το score και το datetime της προσπάθειας
```

Αυτή θα είναι η διεπαφή με την οποία θα επικοινωνεί η εφαρμογή Flask με τη βάση δεδομένων των ερωτήσεων και των παικτών.

Θα πρέπει βεβαίως να σημειώσουμε ότι η υλοποίηση αυτή έχει γίνει για λόγους απλότητας. Σε μια πιο συνηθισμένη εκδοχή, τα δεδομένα θα είναι αποθηκευμένα σε μια βάση δεδομένων, και οι παραπάνω συναρτήσεις θα έκαναν πράξεις στη βάση (πχ με εντολές SQL).

## Μετατροπή του template `question.html`

Στη συνέχεια θα πρέπει να φροντίσουμε να εισάγουμε παραμετρικά στο αρχείο `question.html` τα στοιχεία μιας τυχαίας ερώτησης, για τον σκοπό αυτό χρησιμοποιούμε τη μηχανή template `Jinja2`.

Μέσω των μεταβλητών, `user_name`, `id`, `question` και `replies`, περνάμε στο αρχείο παραμετρικά τα στοιχεία της ερώτησης, ως εξής:

```
<body>
    <h1>Python-Quiz – Χρήστης: {{ user_name }}</h1>
    <h2>Ερώτηση αριθμ. {{ id }} </h2>
    <form>
        <div id="quiz">
            {{ question | safe }}
        </div>
        Απάντηση (επιλέξτε):
        <ol>
            {% for no, reply in replies.items() %}
                <li><input type="radio" name="answer" value="{{ no }}> {{ reply }}</li>
            {% endfor %}
        </ol>
        <button>Υποβολή</button>
    </form>
</body>
```

Το αρχείο αυτό καλείται από την αντίστοιχη συνάρτηση δρομολόγησης ως εξής:

```
q = quiz.show_question(id)
return render_template('question.html', question = q["question"], \
    id = id, user_name=name, replies = q["replies"])
```

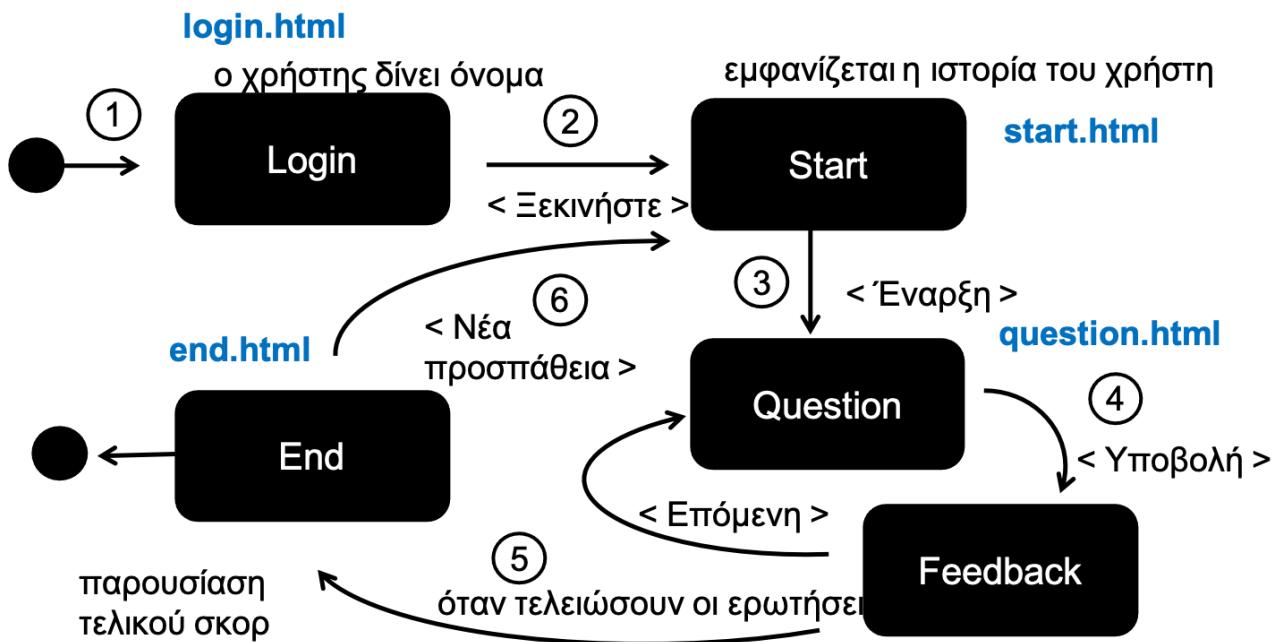
Αυτή είναι η πρώτη μας προσπάθεια παραμετρικής χρήσης του αρχείου `question.html` και η μετατροπή του σε template `Jinja2`.

Στη συνέχεια θα επεκτείνουμε τη σχεδίαση με δημιουργία νέων `templates` για το `login`, παρουσίαση ιστορικού, παρουσίαση τελικού σκορ καθώς επίσης και μια οθόνη παροχής ανάδρασης στον χρήστη.

### Σχεδίαση της εφαρμογής δρομολόγησης

Στη συνέχεια σχεδιάζουμε τον χάρτη της εφαρμογής μας.

Αυτός θα έχει τη μορφή που φαίνεται στο σχήμα:



Θα πρέπει να σχεδιάσουμε με βάση αυτή την προδιαγραφή 4 ιστοσελίδες, οι 3 είναι σχετικά απλές:

- **login.html** εισαγωγή ονόματος χρήστη
- **start.html** εμφάνιση ιστορικού χρήστη και εκκίνηση εφαρμογής
- **end.html** τελικό σκορ και χάρτης, πλήκτρο επανεκκίνησης

Η 4η ιστοσελίδα όμως, η **question.html** είναι αρκετά πιο σύνθετη, αφού χρησιμεύει για δύο λειτουργίες:

1. παρουσίαση της ερώτησης και επιλογή απάντησης από τον χρήστη (όπως είδαμε στην προηγούμενη ενότητα),
2. Παροχή ανάδρασης και συνέχεια στην επόμενη ερώτηση.

Για να μπορέσουμε να πετύχουμε και τις δύο λειτουργίες έχουμε εισάγει νέες μεταβλητές μέσω του template της Ninja2, οι οποίες ανάλογα με την τιμή τους ενεργοποιούνται και εμφανίζουν διαφορετικά στοιχεία. Για παράδειγμα η μεταβλητή **feedback** περιέχει το κείμενο ανάδρασης, και η μεταβλητή **next\_question** τον κωδικό της επόμενης ερώτησης, που θα κληθεί όταν ο χρήστης πάρει την ανάδραση και πατήσει το πλήκτρο "Επόμενη".

Μάλιστα στο αρχείο **question.html** έχει ορισθεί υπό συνθήκη ο προορισμός υποβολής της φόρμας ως εξής:

```

{%
    if disabled %
        <form action=/q/{{next_question}}>
    {% else %}
        <form action=/q/{{id}}>
    {% endif %}
}

```

Η δομή αυτή συνθήκης της Jinja2 χρησιμοποιείται εδώ για να κατευθύνει την υποβολή της φόρμας είτε στο URL που ορίζει η id, δηλαδή για υποβολή απάντησης, ή στο URL που ορίζει η next\_question, δηλαδή για αίτημα για επόμενη ερώτηση.

Με βάση τα παραπάνω καταλήγουμε στα εξής σημεία δρομολόγησης και τα αντίστοιχα URL, που φαίνονται στη συνέχεια:

```
@app.route("/login")
def login():

@app.route("/start")
def start():

@app.route("/end")
def end():

@app.route('/q/<id>')
def question(id):
```

Το πιο ενδιαφέρον από αυτά είναι το τελευταίο το οποίο είναι παραμετρικό URL, όπου είναι ο κωδικός της ερώτησης.

Για παράδειγμα η ερώτηση "Q45" εμφανίζεται στο URL `localhost:5000/Q45`, όπως φαίνεται στη συνέχεια:

The screenshot shows a web-based quiz application. The title is "Python-Quiz - Χρήστης: Nikos" and the section is "Ερώτηση αριθμ. 2 / Q45". The question asks: "Ποιο το αποτέλεσμα:" followed by a code snippet:

```
li = [1,2]
li.extend(2*[4])
print(li)
```

The answer section is titled "Απάντηση (επιλέξτε):" and lists four options:

- [1,2,4,4]
- [1,2,8]
- [1,2,[4,4]]
- Δεν γνωρίζω

A "Υποβολή" button is at the bottom right.

## Διαχείριση συνεδρίας

Ένα πρόβλημα που έχουμε στο διαδίκτυο, είναι η διατήρηση των δεδομένων της κατάστασης σε διαδοχικές κλήσεις σε ένα εξυπηρετητή από τον ίδιο χρήστη. Το πρωτόκολλο HTTP είναι stateless (δεν διατηρεί την κατάσταση).

Στο παράδειγμα αυτό η διατήρηση της κατάστασης γίνεται με χρήση του `session`, δηλαδή μικρού αρχείου κρυπτογραφημένων δεδομένων που στέλνονται στον φυλλομετρητή και επανέρχονται στον εξυπηρετητή, ώστε να ελεγχθεί ότι ο χρήστης είναι ο ίδιος στην επόμενη κλήση, και εν γένει να συντηρείται η *κατάσταση*, μεταξύ κλήσεων.

Αρχικά ορίζουμε ένα κλειδί κρυπτογράφησης:

```
app.config['SECRET_KEY'] = 'hard-to-guess-string'
```

Είναι φανερό ότι το κλειδί για να έχει αξία, θα πρέπει να φυλάσσεται σε ξεχωριστό αρχείο και να μην συνοδεύει τον κώδικα της εφαρμογής μας, η οποία μπορεί να είναι διαθέσιμη σε δημόσια αποθετήρια, όπως το GitHub.

Στη συνέχεια για να γράψουμε δεδομένα στο `session` το κάνουμε ως εξής:

```
session["user_name"] = name
```

Ενώ αν θέλουμε να ανακτήσουμε τα δεδομένα αυτό γίνεται με την εξής εντολή:

```
name = session.get("user_name", None)
```

Θα πρέπει επίσης να σημειώσουμε ότι τα δεδομένα `session` είναι διαθέσιμα και στα `template` `Jinja2`.

---

## 7. Φιλοξενία της εφαρμογής Flask

Φιλοξενία της εφαρμογής Flask σε δημόσια διεύθυνση μέσω της υπηρεσίας [Heroku](#)

Η Heroku είναι μια πλατφόρμα ως υπηρεσία (Platform as a Service, PaaS) που επιτρέπει στους προγραμματιστές να δημιουργούν, να εκτελούν και να χρησιμοποιούν εφαρμογές πλήρως στο cloud. Μια εφαρμογή Heroku εκτελείται σε ένα dyno (ένα Linux box). Η Heroku χρησιμοποιεί υπολογιστικές υπηρεσίες της AWS (Amazon Web Services), ενώ υποστηρίζει εφαρμογές γραμμάνες σε διάφορες γλώσσες προγραμματισμού, Node.js, Python, Java, PHP, Go κ.λπ.

Ο λογαριασμός Heroku για την δωρεάν έκδοση που θα χρησιμοποιήσουμε πέφτει σε αναστολή λειτουργίας μετά από 30 λεπτά αδράνειας, για αυτό η σύνδεση στην εφαρμογή έχει κάποια μικρή καθυστέρηση, μέχρι την επανεκκίνηση του dyno. Ένας επαληθευμένος λογαριασμός συνοδεύεται από 1000 dyno-hour (CPU) ανά μήνα.

Το Heroku μάς επιτρέπει να δημιουργήσουμε προσαρμοσμένες διευθύνσεις διαδικτύου, αν για παράδειγμα έχουμε μια εφαρμογή με όνομα [mypythonquiz](#), και την εγκαταστήσουμε στην πλατφόρμα αυτή, θα μπορούμε να την εκτελέσουμε από το URL  
<https://mypythonquiz.herokuapp.com>.

Βήματα εγκατάστασης της εφαρμογής

### Βήμα1: Εγκαθιστούμε τη βιβλιοθήκη gunicorn

```
pip install gunicorn
```

Η βιβλιοθήκη **Gunicorn** είναι ένας εξυπηρετητής HTTP γραμμένος σε Python για εφαρμογές WSGI. Το πλεονέκτημα του εξυπηρετητή αυτού είναι ότι επιτρέπει να εκτελούμε εφαρμογές Python ταυτόχρονα εκτελώντας πολλές διεργασίες Python σε ένα μόνο dynamo.

### Βήμα 2: Δημιουργία αρχείου requirements.txt

Στη συνέχεια δημιουργούμε ένα αρχείο **requirements.txt** το οποίο είναι χρήσιμο για το νέο περιβάλλον, αφού σε αυτό περιγράφονται όλες οι βιβλιοθήκες από τις οποίες εξαρτάται η εφαρμογή μας.

```
$ pip freeze > requirements.txt
```

Στο δικό μας παράδειγμα το αρχείο αυτό έχει το εξής περιεχόμενο:

```
click==7.1.2
Flask==1.1.2
gunicorn==20.1.0
itsdangerous==1.1.0
```

```
Jinja2==2.11.3  
MarkupSafe==1.1.1  
Werkzeug==1.0.1
```

### Βήμα 3: Δημιουργία αρχείου Procfile

Το αρχείο **Procfile** είναι ένα αρχείο από το οποίο το dyno θα αναζητήσει τις εντολές για να ξεκινήσει την εφαρμογή μας.

Στη δική μας περίπτωση που η εφαρμογή μας ονομάζεται mypythonquiz και το αρχείο του server είναι το αρχείο **server.py**, στο Procfile εισάγουμε την εξής εντολή:

```
web: gunicorn server:app
```

### Βήμα 4: Δημιουργία λογαριασμού Heroku

- Για την εγκατάσταση της εφαρμογής μας, αρχικά δημιουργούμε ένα λογαριασμό [Heroku](#).
- Στη συνέχεια πραγματοποιούμε λήψη και εγκατάσταση του Heroku CLI.
- Συνδεόμαστε στον λογαριασμό μας Heroku και ακολουθούμε τις οδηγίες για να δημιουργήσουμε ένα νέο δημόσιο κλειδί SSH και στη συνέχεια να ανεβάσουμε μέσω git την εφαρμογή.

```
$ heroku login $ heroku git:clone -a mypythonquiz $ git add . $ git commit -am "έκδοση 1.0" $ git push heroku master
```

Παρατηρούμε ότι εγκαθίστανται στο dyno η python και οι διάφορες εξαρτήσεις που έχουμε ορίσει στο αρχείο **requirements.txt**

Τελικά αν ολοκληρωθεί η εγκατάσταση θα πάρουμε ένα μήνυμα

```
remote: https://mypythonquiz.herokuapp.com/ deployed to Heroku
```

Είμαστε έτοιμοι να συνδεθούμε και να χρησιμοποιήσουμε την εφαρμογή μας από το διαδίκτυο.

Θα πρέπει να σημειωθεί ότι όλα τα παραπάνω είναι δυνατά μόνο αν έχουμε δωρεάν λογαριασμό στο Heroku. Αυτό επιτρέπεται για φοιτητές και εκπαιδευτικούς οργανισμούς μέσω του

## 8. Εισαγωγή στο full Python web stack

---

Μια σχετικά νέα πρόταση για ανάπτυξη διαδικτυακών εφαρμογών είναι αυτή που επιτρέπει να χρησιμοποιηθεί η γλώσσα προγραμματισμού Python και στο τμήμα πελάτη όπως και σε αυτό του εξυπηρετητή, αντί για το παραδοσιακό μοντέλο HTML/CSS/JS .

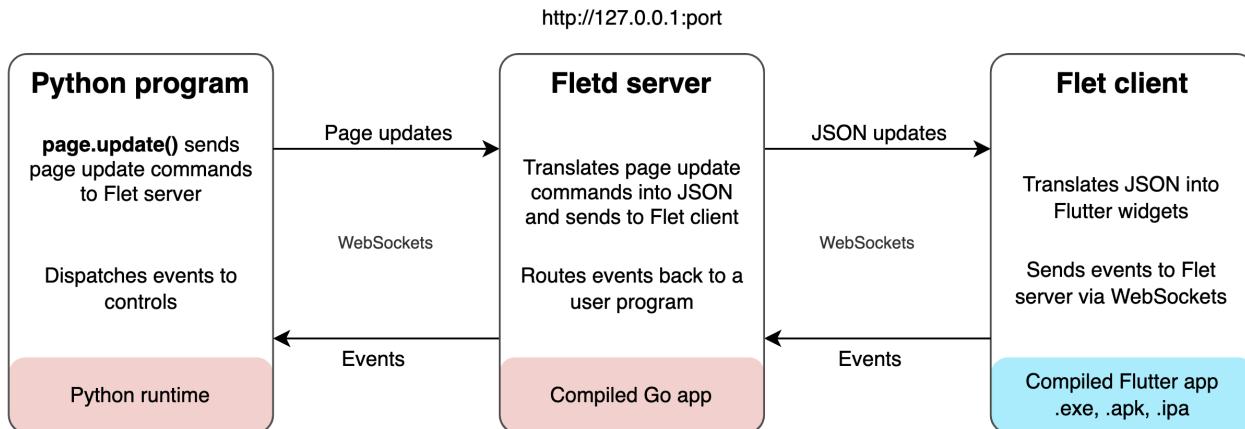
Το μοντέλο αυτό στηρίζεται στο πλαίσιο [Flet](#), που επιτρέπει την ανάπτυξη στατικών σελίδων, δηλαδή σελίδων που δεν χρειάζονται φόρτωμα δεδομένων, όπως είδαμε στο παράδειγμα της εφαρμογής στις προηγούμενες ενότητες (που το περιεχόμενο της σελίδας καθοριζόταν σε πραγματικό χρόνο, με χρήση του template [Jinja2](#)) από τον server.

Το Flet επιτρέπει, με χρήση της γλώσσας προγραμματισμού Python, να έχουμε πρόσβαση σε ένα υποσύνολο των γραφικών στοιχείων του [Flutter](#), ενός δημοφιλούς τα τελευταία χρόνια πλαισίου ανάπτυξης εφαρμογών για κινητά, το διαδίκτυο και τον υπολογιστή, που έχει δημιουργηθεί από την Google και παραδοσιακά συνδυάζεται με τη γλώσσα προγραμματισμού Dart. Το Flutter προσφέρει ένα σύνολο έτοιμων γραφικών στοιχείων (widgets) και εργαλείων, που απλοποιούν τη δημιουργία αισθητικά άρτιων και λειτουργικών διεπαφών χρήστη για διαφορετικές πλατφόρμες, μεταξύ των οποίων το περιβάλλον του φυλλομετρητή.

Η βιβλιοθήκη Flet χρησιμοποιεί την τεχνολογία WebAssembly/Emscripten, με χρήση της βιβλιοθήκης [Pyodide](#) που καθιστά δυνατή την εγκατάσταση και εκτέλεση κώδικα Python στον φυλλομετρητή. Η WebAssembly (συναντάται συχνά με τη συντομογραφία wasm) είναι μια δυαδική μορφή για εκτέλεση κώδικα στο διαδίκτυο. Πρόκειται για μια εικονική μηχανή χαμηλού επιπέδου που εκτελεί κώδικα ταχύτερα από τις παραδοσιακές μηχανές JavaScript και έχει σχεδιαστεί για να είναι φορητή σε διάφορες πλατφόρμες και συσκευές. Η WebAssembly δημιουργήθηκε ως κοινή προσπάθεια πολλών κατασκευαστών φυλλομετρητών, συμπεριλαμβανομένων των Mozilla, Google, Microsoft και Apple, για να αντιμετωπίσει ορισμένους από τους περιορισμούς απόδοσης της JavaScript και να επιτρέψει στους προγραμματιστές να γράφουν εφαρμογές ιστού υψηλής απόδοσης σε άλλες γλώσσες προγραμματισμού, όπως στην περίπτωσή μας στην Python.

Το Flet είναι σε φάση ανάπτυξης, για την ώρα (Απρίλιος 2023) η βιβλιοθήκη μπορεί να παράγει εφαρμογές desktop και web, ενώ είναι σε εξέλιξη η ανάπτυξη της τεχνολογίας για εφαρμογές κινητών Android και iOS.

Η αρχιτεκτονική του περιβάλλοντος εκτέλεσης μιας εφαρμογής desktop στον υπολογιστή φαίνεται στη συνέχεια, όπως περιγράφεται στην [τεκμηρίωση](#) της βιβλιοθήκης αυτής.



Σύμφωνα με το σχήμα, μια εφαρμογή Flet που τρέχει σε παράθυρο του λειτουργικού συστήματος περιλαμβάνει τρεις διεργασίες, το πρόγραμμα Python, τον εξυπηρετητή Fletd και τον πελάτη, που είναι μια εφαρμογή Flutter. Η επικοινωνία μεταξύ των επί μέρους διεργασιών γίνεται με websockets.

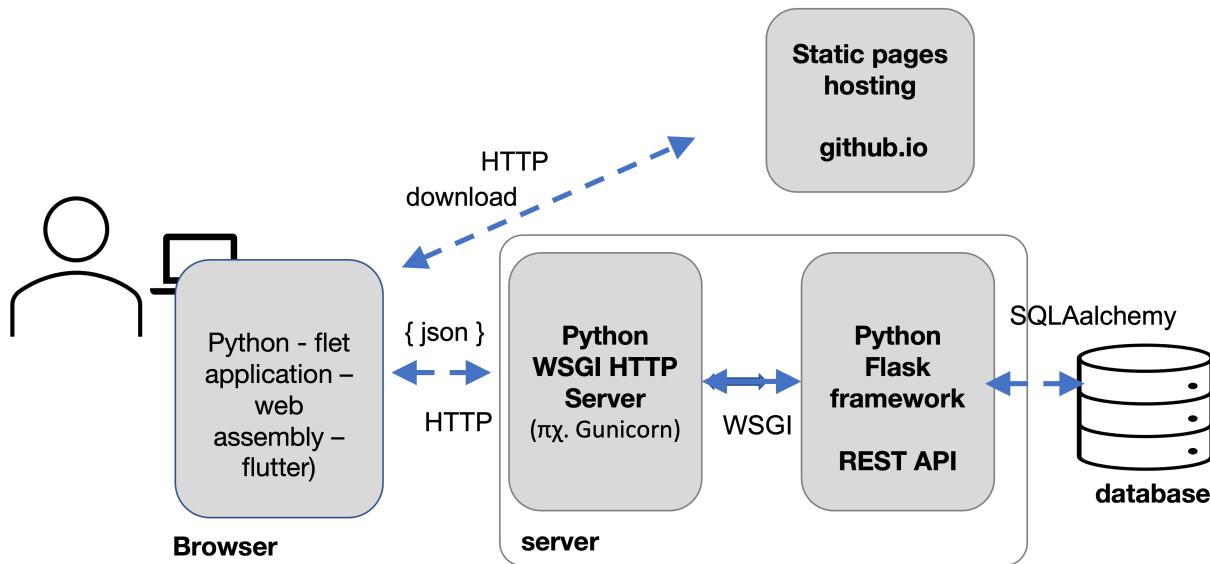
Από το περιβάλλον Flet μπορούμε να αναπτύξουμε εφαρμογές για το λειτουργικό σύστημα του υπολογιστή μας, αλλά και διαδικτυακές εφαρμογές ιστού.

Στο πλαίσιο του μαθήματος αυτού μας ενδιαφέρει το δεύτερο, αν και θα πρέπει να σημειωθεί ότι και η πρώτη λύση έχει ενδιαφέρον, γιατί η flet επιτρέπει την γρήγορη ανάπτυξη διαδραστικών εφαρμογών, πολύ πιο εύκολα από άλλες γραφικές βιβλιοθήκες, όπως η tkinter, και άλλες.

## 9. Η αρχιτεκτονική διαδικτυακής εφαρμογής με full Python stack.

Στην ενότητα αυτή θα περιγραφεί η αρχιτεκτονική μιας διαδικτυακής εφαρμογής που ακολουθεί το full-Python stack, δηλαδή που η ανάπτυξη και του server και του client έχει στηριχτεί σε βιβλιοθήκες της Python.

Η βασική ιδέα της αρχιτεκτονικής φαίνεται στην παρακάτω εικόνα.



Θα αναπτύξουμε το τμήμα πελάτη (front-end) της εφαρμογής μας που μπορεί να αφορά ένα σύνολο από ιστοσελίδες οι οποίες αλληλεπιδρούν με τον χρήστη. Για το τμήμα αυτό θα κάνουμε χρήση του πλαισίου Flet. Οι σελίδες αυτές ακολουθούν το μοντέλο εφαρμογής Single Page, αφού η μετάβαση από σελίδα σε σελίδα δεν προϋποθέτει κλήση του εξυπηρετητή.

Η εφαρμογή αυτή θα μπορεί να φιλοξενείται σε εξυπηρετητή στατικών σελίδων, όπως για παράδειγμα το GitHub, δηλαδή σελίδων git.io.

Ο χρήστης όταν κατεβάζει στο περιβάλλον του φυλλομετρητή του την εφαρμογή, αυτή θα εκτελείται ως κώδικας webassembly και θα κάνει χρήση των γραφικών στοιχείων της Flutter για να εμφανιστεί στο παράθυρο του φυλλομετρητή.

Αν η εφαρμογή έχει ανάγκη από δεδομένα που βρίσκονται σε εξυπηρετητή ή σε κάποια βάση δεδομένων, θα πρέπει να ζητήσει τα δεδομένα με κλήσεις HTTP.

Θα πρέπει να έχουμε φροντίσει να έχουμε εγκαταστήσει έναν εξυπηρετητή REST (Representational State Transfer), δηλαδή έναν εξυπηρετητή ιστού που με χρήση του πρωτοκόλλου HTTP θα ικανοποιεί αιτήματα παροχής και ανάκτησης δεδομένων από εξουσιοδοτημένους πελάτες, όπως στην

περίπτωσή μας το τμήμα πελάτη της εφαρμογής μας. Συνήθως τα δεδομένα έχουν μορφή σειριοποιημένων (Json) αντικειμένων, που αντιστοιχούν σε λεξικά της Python.

## Εγκαστάσταση βιβλιοθήκης flet

Για να αναπτύξουμε την εφαρμογή flet αρκεί να κατεβάσουμε τη βιβλιοθήκη μέσω pip. Η τρέχουσα έκδοση είναι η 0.6, και η βιβλιοθήκη είναι σε συνεχή ανάπτυξη.

```
% pip3 install flet
Collecting flet
  Downloading flet-0.6.2-py3-none-macosx_12_0_arm64.whl (32.8 MB)
    Installing collected packages: websockets, websocket-client, watchdog, repath, packaging, oauthlib, h11, anyio, httpcore, flet-core, httpx, flet ...
      
```

Η βιβλιοθήκη περιλαμβάνει ένα web server ο οποίος μπορεί να χρησιμοποιηθεί ως localhost για εφαρμογές σε παράθυρα του λειτουργικού ή στον φυλλομετρητή μας, κατά τη φάση ανάπτυξης. Ας δούμε στη συνέχεια τις λειτουργίες και τη δομή μιας εφαρμογής Flet.

## Η πρώτη εφαρμογή

```
import flet as ft

def main(page: ft.Page):
    # add/update controls on Page
    pass

ft.app(target=main)
```

Η εφαρμογή αυτή θα τρέξει σε ένα παράθυρο του λειτουργικού συστήματος. Από προεπιλογή, η εφαρμογή Flet ξεκινά σε ένα νέο παράθυρο του λειτουργικού συστήματος, το οποίο είναι πολύ βιολικό για την φάση της ανάπτυξης. Ωστόσο, μπορεί επίσης να ανοίξει σε ένα νέο παράθυρο του φυλλομετρητή, τροποποιώντας την κλήση στο flet.app ως εξής:

```
ft.app(target=main, view=ft.WEB_BROWSER)
```

Ένα τυπικό πρόγραμμα Flet τελειώνει με μια κλήση στην flet.app(), όπου η εφαρμογή αρχίζει να περιμένει για νέες συνεδρίες χρήστη. Η συνάρτηση main() είναι ένα σημείο εισόδου σε μια εφαρμογή Flet. Καλείται σε ένα νέα νήμα για κάθε συνεδρία χρήστη με ένα στιγμιότυπο της κλάσης **Page** που περνάει σε αυτήν.

Κατά την εκτέλεση της εφαρμογής Flet στον φυλλομετρητή ξεκινά μια νέα σύνοδος χρήστη για κάθε ανοιχτή καρτέλα ή σελίδα. Όταν εκτελείται ως εφαρμογή στο λειτουργικό μας σύστημα, δημιουργείται μόνο μία συνεδρία.

Η Σελίδα είναι σαν ένας "καμβάς" του χρήστη. Για να δημιουργήσουμε τη διεπαφή μιας εφαρμογής προσθέτουμε και αφαιρούμε γραφικά στοιχεία (στοιχεία ελέγχου, controls) σε μια σελίδα, και αλλάζουμε την εμφάνισή τους ενημερώνοντας τα γνωρίσματά τους. Εδώ θα πρέπει να παρατηρήσουμε ότι εντολές που αφορούν την εμφάνιση και αυτές που αφορούν τη δομή της σελίδας αναμειγνύονται στον ίδιο κώδικα, και δεν ξεχωρίζουν σε κώδικα HTML και CSS, όπως γίνεται στο παραδοσιακό μοντέλο.

---

# 10. Γραφικά στοιχεία (widgets)

---

Η διεπαφή χρήστη περιέχει γραφικά στοιχεία (widgets ή controls). Ένα γραφικό στοιχείο μπορεί να περιέχεται στη σελίδα ή σε ένα υποδοχέα (container).

Για να εμφανίσουμε ένα γραφικό στοιχείο σε μια σελίδα, το προσθέτουμε στη λίστα στοιχείων ελέγχου της σελίδας (`page.controls`) και αφού ολοκληρωθεί η προσθήκη των στοιχείων καλείται η μέθοδος `page.update()` για να εμφανίσουμε τις αλλαγές της σελίδας στο παράθυρο.

Για παράδειγμα αν επιθυμούμε να εντάξουμε το γραφικό αντικείμενο `ft.Text()` που εμφανίζει κείμενο, αντίστοιχα σε μια παράγραφο κειμένου `<p> ... </p>` της HTML ή σε ένα γραφικό αντικείμενο `tk.Label()` της tkinter, θα πρέπει να δώσουμε τις εξής εντολές.

```
import flet as ft

def main(page: ft.Page):
    t = ft.Text(value="Καλημέρα Flet!", color="blue")
    page.controls.append(t)
    page.update()

ft.app(target=main)
```

Αν στο πρόγραμμα αυτό επιθυμούμε να προσθέσουμε ένα ακόμη γραφικό αντικείμενο `ft.Text` του οποίου το περιεχόμενο να ανανεώνεται συνεχώς (πχ με μικρή καθυστέρηση ενός msec ώστε να μπορούμε να δούμε τις διαδοχικές τιμές), τότε θα πρέπει να αλλάξουμε την τιμή του γνωρίσματος `value` του στοιχείου και στη συνέχεια να ανανεώσουμε τη σελίδα. Την συμπεριφορά αυτή επιτυγχάνουμε με τον κατωτέρω κώδικα:

```
import flet as ft
import time

def main(page: ft.Page):
    # add/update controls on Page
    t = ft.Text(value="Καλήμέρα Flet!", color="blue")
    page.controls.append(t)
    t1 = ft.Text()
    page.controls.append(t1)
    i = 0
    while True:
        t1.value = f"Βήμα {i}"
        page.update()
        i += 1
        time.sleep(0.1)
```

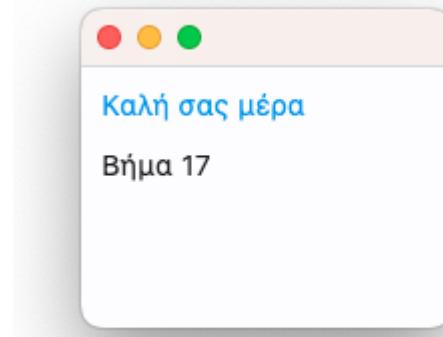
```

page.update()
i += 1
time.sleep(1)

ft.app(target=main)

```

Το αποτέλεσμα φαίνεται σε ένα παράθυρο του λειτουργικού ως εξής (με συνεχή ανανέωση του δεύτερου μηνύματος):



Άλλα γραφικά αντικείμενα που διατίθενται είναι τα Icon, Image, Markdown, ProgressBar, διάφορα είδη Buttons (π.χ. ElevatedButton, FilledButton, FloatingActionButton, IconButton, κλπ.), διάφοροι επιλογείς όπως Checkbox, Dropdown, Radio, Slider, Switch, το στοιχείο TextField για εισαγωγή κειμένου, κλπ. Δηλαδή περιλαμβάνονται τα περισσότερα γραφικά αντικείμενα που συναντάμε σε άλλες γραφικές βιβλιοθήκες όπως η tkinter και η HTML/CSS.

Ας δούμε ένα παράδειγμα ενός γραφικού στοιχείου που μας επιτρέπει να εισάγουμε μορφοποιημένο markdown κείμενο, χρήσιμο για παρουσίαση κώδικα.

```

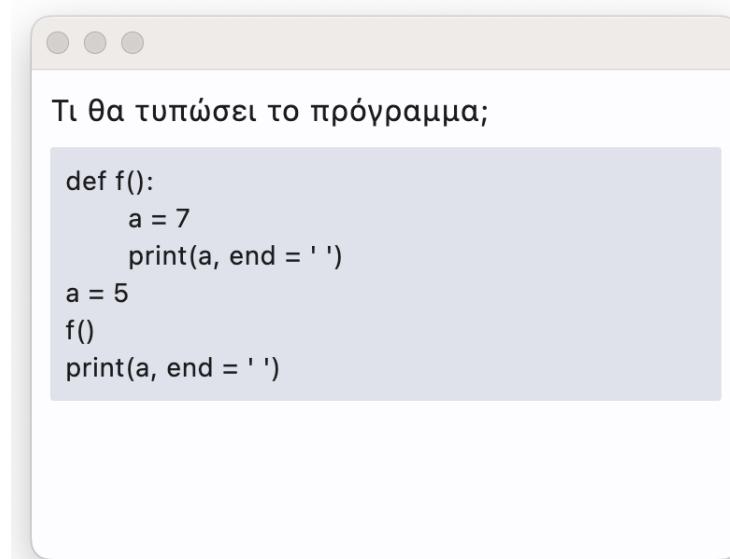
md_python = """
#### Τι θα τυπώσει το πρόγραμμα;
```
def f():
    a = 7
    print(a, end = ' ')
a = 5
f()
print(a, end = ' ')
```
"""

def main(page: ft.Page):
    page.scroll = "auto"
    page.add(

```

```
ft.Markdown(  
    md_python,  
    selectable=False,  
    extension_set=ft.MarkdownExtensionSet.GITHUB_WEB  
)  
  
ft.app(target=main)
```

Ο παραπάνω κώδικας θα παρουσιάσει το εξής:



# 11. Υποδοχείς (containers)

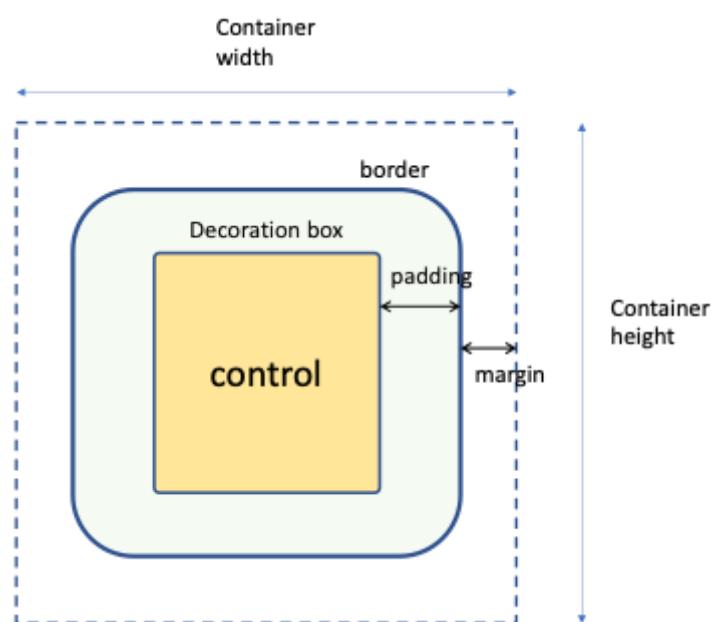
Η Flet διαθέτει γραφικά στοιχεία υποδοχείς που περιέχουν άλλα γραφικά στοιχεία.

Οι κύριοι υποδοχείς είναι το ίδιο το στοιχείο **Page** που όπως είδαμε περιέχει άλλα στοιχεία, το στοιχείο **Container** που περιέχει ένα άλλο γραφικό στοιχείο και ορίζει ένα πλαίσιο διακόσμησής του (decoration box), στο οποίο επιτρέπει να οριστούν διάφορες ιδιότητες του, όπως το πλάτος, ύψος, περιθώριο, αποστάσεις από τα όρια, χρώμα υποβάθρου ευθυγράμμιση του σε σχέση με τα άλλα στοιχεία, κλπ. Επίσης υπάρχουν τα στοιχεία διάταξης περιεχομένου, τα οποία περιέχουν άλλα στοιχεία, αυτά είναι οι υποδοχείς **Row** και **Column** που επιτρέπουν την διάταξη των στοιχείων που περιέχουν κατά την οριζόντια και την κατακόρυφη διάταξη αντίστοιχα.

Ας δημιουργήσουμε αρχικά ένα Υποδοχέα ενός στοιχείου **Text**

```
ft.Container(
    content=ft.Text("Container-1", color="WHITE"),
    margin=20, padding=20, alignment=ft.alignment.center,
    bgcolor=ft.colors.AMBER, width=150, height=150,
    border_radius=20,
),
```

Ο υποδοχέας αυτός περιέχει το στοιχείο και ορίζει τη στοίχισή του (alignment) τα περιθώρια μεταξύ των ορίων του container και το χρώμα του διακοσμητή του. Θα πρέπει να σημειωθεί ότι τα γνωρίσματα αυτά, ορίζονται όπως φαίνεται στο παρακάτω σχήμα:



Στη συνέχεια ας δούμε μια χρήση ένταξης του Container αυτού σε ένα στοιχείο Row το οποίο εντάσσεται σε μια σελίδα.

```
def main(page: ft.Page):
    page.title = "Containers"
    page.vertical_alignment = ft.MainAxisAlignment.CENTER
    page.horizontal_alignment = ft.CrossAxisAlignment.CENTER

    page.add(
        ft.Row(
            [
                ft.Container(
                    content=ft.Text("Container-1", color="WHITE"),
                    margin=20,
                    padding=20,
                    alignment=ft.alignment.center,
                    bgcolor=ft.colors.AMBER,
                    width=150,
                    height=150,
                    border_radius=20,
                ),
                ft.Container(
                    content=ft.Text("Container-2", color="WHITE"),
                    margin=20,
                    padding=20,
                    alignment=ft.alignment.center,
                    bgcolor=ft.colors.GREEN_200,
                    width=150,
                    height=150,
                    border_radius=20,
                ),
            ],
            alignment=ft.MainAxisAlignment.SPACE_EVENLY
        )
    )

ft.app(target=main)
```

Από το παράδειγμα φαίνεται ότι το στοιχείο `ft.Row()`, δέχεται ως όρισμα μια λίστα από στοιχεία, εδώ δύο `Containers` που με τη σειρά τους περιέχουν στοιχεία `Text`, επίσης δέχεται ως όρισμα το `γνώρισμα` που ορίζει τον τρόπο οριζόντιας στοίχισης (κατά μήκος του κυρίως άξονα), η τιμή που έχει το γνώρισμα `ft.MainAxisAlignment.SPACE_EVENLY` ορίζει ότι τα στοιχεία του Row, θα διαμοιράζουν ομοιόμορφα τον κενό χώρο που διατίθεται μεταξύ τους και με τα όρια της σελίδας.

Επίσης στο παράδειγμα αυτό παρατηρούμε δύο ακόμη σημαντικά γνωρίσματα της σελίδας, που είναι τα `page.vertical_alignment` και `page.horizontal_alignment`. Και τα στοιχεία αυτά έχουν πάρει τιμές που ορίζουν ότι το στοιχείο της σελίδας στοιχίζεται στο κέντρο και ως προς τον οριζόντιο και ως προς τον κατακόρυφο άξονα.

## Χειριστές συμβάντων

Όπως σε κάθε διαδραστικό σύστημα, βασικός μηχανισμός απόκρισης σε ενέργειες του χρήστη είναι ο μηχανισμός χειρισμού συμβάντων.

Ας δούμε ένα παράδειγμα ορισμού χειριστή συμβάντος (event handler) που καλείται όταν ο χρήστης πατήσει ένα πλήκτρο (button). Η σύνδεση της συνάρτησης χειριστή handler(event) γίνεται είτε ως γνώρισμα του γραφικού αντικειμένου button είτε με ξεχωριστή εντολή απόδοσης τιμής στην ιδιότητα `on_click`.

```
b = ft.ElevatedButton(text='Υποβολή', on_click=submit_handler)
# έναλλακτικά:
b = ft.ElevatedButton(text='Υποβολή')
b.on_click = submit_handler
```

Σε κάθε περίπτωση θα πρέπει να έχουμε ορίσει μια συνάρτηση `submit_handler(event)` η οποία θα καλείται όταν προκύψει το συμβάν "πάτημα πλήκτρου".

Ας δούμε τώρα ένα παράδειγμα

```
import flet as ft

def main(page: ft.Page):
    def add_handler(e):
        if not new_task.value: return
        page.add(ft.Checkbox(label=new_task.value))
        new_task.value = ""
        new_task.focus()
        new_task.update()

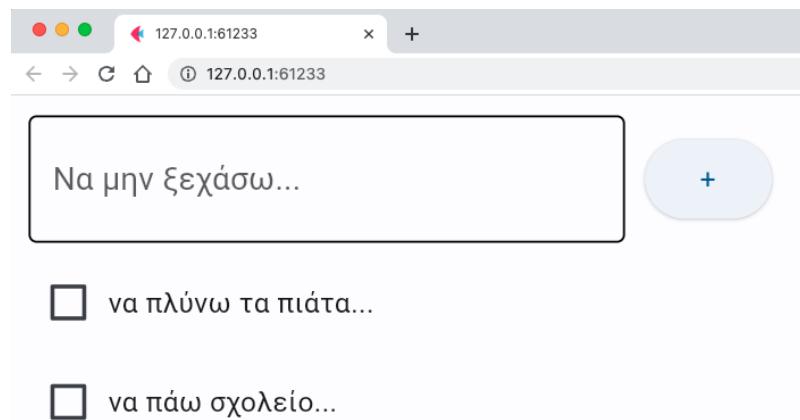
        new_task = ft.TextField(hint_text="Να μην ξεχάσω...", color="GRAY", width=250)
        page.add(ft.Row([new_task, ft.ElevatedButton(" + ", on_click=add_handler)]))

    ft.app(target=main)
```

Στο παράδειγμα αυτό παρατηρούμε ότι έχουμε μια σελίδα στην οποία αρχικά προστίθενται δύο γραφικά στοιχεία, το new\_text που είναι ένα TextField (πεδίο εισαγωγής κειμένου), και ένα ElevatedButton που είναι ένα πλήκτρο. Στο πλήκτρο έχει προστεθεί μια συνάρτηση χειρισμού του συμβάντος 'click', η add\_handler. Η συνάρτηση αυτή έχει συσχετιστεί με το συμβάν 'click', μέσω της ιδιότητας on\_click του πλήκτρου.

Παρατηρούμε ότι η συνάρτηση αυτή (η οποία όπως όλες οι συναρτήσεις χειριστές παίρνει ως όρισμα το αντικείμενο event), εκτελεί τις εξής λειτουργίες: (α) αν δεν υπάρχει κείμενο στο πεδίο εισαγωγής κειμένου δεν κάνει καμιά ενέργεια (β) αντίθετα, προσθέτει ένα στοιχείο Checkbox στη σελίδα, δίπλα από το οποίο εισάγει το κείμενο που ο χρήστης έχει εισαγάγει στο πεδίο εισαγωγής κειμένου, (γ) στη συνέχεια ανανεώνει το στοιχείο new\_task και επιστρέφει εκεί την εστίαση του χρήστη

Το αποτέλεσμα φαίνεται στην παρακάτω εικόνα:



Μια επέκταση του παραδείγματος αυτού είναι η εξής: Όπως φαίνεται η χρήση της σελίδας αυτής είναι ο χρήστης να εισαγάγει ένα κείμενο στο πεδίο κειμένου και να πατάει το πλήκτρο "+" ώστε να ενεργοποιήσει τη συνάρτηση χειριστή συμβάντων add\_handler. Αυτό όμως δεν είναι βολικό για τον χρήστη που αναγκάζεται να αφήσει το πληκτρολόγιο και να μεταφέρει το χέρι του στο ποντίκι, αν είναι σε περιβάλλον προσωπικού υπολογιστή. Για να αντιμετωπίσουμε το πρόβλημα αυτό θα πρέπει να μεριμνήσουμε εναλλακτικά του πατήματος του πλήκτρου, η πληκτρολόγηση του <Enter> να καλεί επίσης τον χειριστή συμβάντων. Αυτό μπορεί να γίνει με δημιουργία ενός νέου χειριστή όπως φαίνεται στη συνέχεια.

```
def keyboard_handler(event):
    if event.key == "Enter":
        add_handler(event)

page.on_keyboard_event = keyboard_handler
```

Ο νέος χειριστής συνδέεται με το συμβάν "kyboard\_event", όταν κληθεί, ελέγχει αν το συμβάν αφορά την πληκτρολόγηση του "Enter", αν ναι, τότε καλείται ο χειριστής add\_handler. Παρατηρούμε ότι το συμβάν που προέρχεται από πληκτρολόγιο έχει την ιδιότητα key που λαμβάνει τιμή του

αντίστοιχου πλήκτρου του πληκτρολογίου, "A", "B", κλπ, ενώ έχει ακόμη τις ιδιότητες event.shift για το πλήκτρο SHIFT, και αντίστοιχα για άλλα πλήκτρα ελέγχου.

## Δημιουργία φόρμας, στοιχεία επιλογής

Μια συνηθισμένη απαίτηση γραφικών διεπαφών είναι η δημιουργία φόρμας εισαγωγής και υποβολής δεδομένων από τον χρήστη.

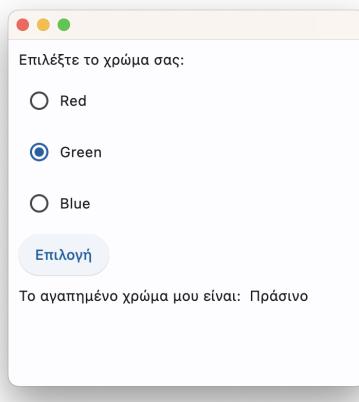
Μια φόρμα περιλαμβάνει συνήθως πεδία εισαγωγής κειμένου, πλήκτρα επιλογής μιας από αμοιβαία αποκλειόμενες επιλογές (radio buttons), στοιχεία checkbox που επιτρέπουν την επιλογή τους ή όχι, κλπ.

Μέχρι τώρα έχουμε δει γραφικά στοιχεία χρήσιμα για φόρμες, αυτά είναι το **TextField** (πεδίο εισαγωγής κειμένου), το **Checkbox**, στοιχείο που ο χρήστης μπορεί να επιλέξει ή όχι, το **ElevatedButton** που είναι ένα πλήκτρο που πατάει ο χρήστης για υποβολή της φόρμας.

Στο παράδειγμα που θα αναπτυχθεί στη συνέχεια και αφορά ένα quiz γνώσεων με επιλογή μιας σωστής απάντησης, θα χρησιμοποιήσουμε, ένα νέο γραφικό στοιχείο, το radio button, για την περίπτωση επιλογής μιας σωστής απάντησης. Αν οι σωστές απαντήσεις ήταν περισσότερες από μια, τότε θα αρκούσε δίπλα σε κάθε απάντηση να εμφανίσουμε ένα checkbox.

Το παράδειγμα χρήσης του radio button φαίνεται στη συνέχεια.

Έστω ότι επιθυμούμε να ζητήσουμε από τον χρήστη μια από τις παρακάτω 3 επιλογές:



Όταν κάνει μια επιλογή και πατήσει το πλήκτρο, τότε εμφανίζεται το σχετικό μήνυμα. Επίσης ας υποθέσουμε ότι είναι αναγκαστική η επιλογή ενός χρώματος.

Θα χρησιμοποιήσουμε το γραφικό στοιχείο RadioGroup που είναι ένας υποδοχέας στοιχείων Radio, όπως φαίνεται στον παρακάτω κώδικα:

```
import flet as ft

def main(page):
    def button_clicked(e):
```

```

if not calor_group.value: text.value = f"Παρακαλώ επιλέξτε χρώμα"
else: text.value = f"Το αγαπημένο χρώμα μου είναι: {cg.value}"
page.update()

text = ft.Text()
button = ft.ElevatedButton(text='Επιλογή', on_click=button_clicked)
calor_group = ft.RadioGroup(content=ft.Column([
    ft.Radio(value="Κόκκινο", label="Red"),
    ft.Radio(value="Πράσινο", label="Green"),
    ft.Radio(value="Μπλε", label="Blue")]))
page.add(ft.Text("Επιλέξτε το χρώμα σας:"), cakir_group, button,
text)

ft.app(target=main)

```

Θα μπορούσαμε να τροποποιήσουμε τον κώδικα, ώστε το μήνυμα να παράγεται κάθε φορά που ο χρήστης επιλέγει ένα χρώμα, χωρίς απαραίτητα να πατήσει το πλήκτρο "Υποβολή".

Στην περίπτωση αυτή θα πρέπει να προστεθεί ένας ακόμη χειριστής στο στοιχείο `cg`

```
on_change=radiogroup_changed
```

και η σχετική συνάρτηση-χειριστής θα μπορούσε να είναι:

```

def radiogroup_changed(e):
    t.value = f"Το αγαπημένο χρώμα μου είναι: {e.control.value}"
    page.update()

```

## Χρωματική παλέτα

Η επιλογή χρωμάτων θα πρέπει να γίνεται με προσοχή, αφού η Flet έχει υλοποιήσει την παλέτα του γραφικού συστήματος Material UI. Αν θέλουμε να δώσουμε τα δικά μας χρώματα θα πρέπει να συμβουλευτούμε την [παλέτα χρωμάτων](#) του Flutter.

Επειδή όμως η κλάση `Colors` δεν έχει υλοποιηθεί στην Flet, η επιλογή χρώματος θα πρέπει να γίνει με χρήση της συμβολοσειράς που αντιστοιχεί στη χρωματική απόχρωση. Για παράδειγμα για να επιλέξουμε το χρώμα `Colors.lightGreen[700]` θα πρέπει στην παράμετρο κάποιου στοιχείου να δώσουμε την τιμή `color = "lightGreen700"`.

Θα πρέπει να σημειωθεί ότι η παλέτα έχει αποχρώσεις που παίρνουν τιμές από 100 μέχρι 900, με όσο μεγαλύτερη τιμή τόσο πιο σκούρα η χρωματική απόχρωση.

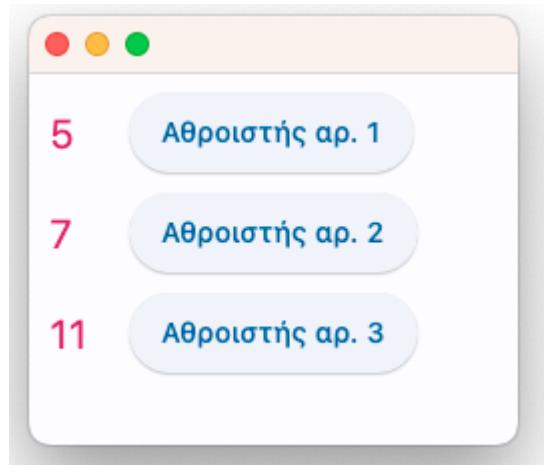
Τέλος να σημειωθεί ότι ένας περιορισμένος αριθμός από βασικά χρώματα μπορούν να περιγραφούν λεκτικά, "green", "blue", "pink" κλπ. Επίσης επιτρέπεται η εισαγωγή χρώματος με την δεκαεξαδική του αναπαράσταση **color = "#0B2447"**

---

## 12. Δημιουργία στοιχείων από τον χρήστη

Μια χρήσιμη δυνατότητα που μάς παρέχει το πλαίσιο Flet είναι η δημιουργία στοιχείων που μπορούμε να επαναχρησιμοποιήσουμε, ακολουθώντας το αντικειμενοστρεφές μοντέλο προγραμματισμού. Τα νέα αυτά στοιχεία ορίζονται ως κλάσεις που κληρονομούν το στοιχείο `userControl` της Flet και περιέχουν μια μέθοδο `build()` η οποία επιστρέφει ένα γραφικό στοιχείο (συνήθως ένα στοιχείο υποδοχέας Row, Column κλπ που περιέχει άλλα στοιχεία). Αντικείμενα της κλάσης αυτής μπορούμε στη συνέχεια να επαναχρησιμοποιήσουμε όπως και τα υπόλοιπα γραφικά στοιχεία της Flet στο κυρίως μας πρόγραμμα.

Ας δούμε ένα παράδειγμα. Έστω ότι δημιουργούμε ένα στοιχείο Αθροιστής (Counter) που αποτελείται από ένα στοιχείο `Text` που αρχικά έχει την τιμή 0 και ένα πλήκτρο `ElevatedButton` που όταν πατηθεί αυξάνει την τιμή κατά ένα. Έστω ότι επιθυμούμε κάθε αθροιστής να έχει μια ταυτότητα και θέλουμε να εισάγουμε πολλούς αθροιστές στη σελίδα μας, όπως στην εικόνα:



Δημιουργούμε το στοιχείο χρήστη `Counter`, και στη συνέχεια το εισάγουμε στο κυρίως πρόγραμμα με την εντολή `page.add(Counter(1), Counter(2), Counter(3))`. Όλη η συμπεριφορά του στοιχείου ορίζεται μέσα στην κλάση `Counter`, και έτσι ο κώδικας είναι καλύτερα δομημένος. Μάλιστα κάθε στοιχείο χρήστη μπορεί να υπάρχει σε ξεχωριστό αρχείο, βιοηθώντας την καλύτερη δόμηση και συντηρησιμότητα της εφαρμογής.

```
import flet as ft
class Counter(ft.UserControl):
    def __init__(self, id=''):
        super().__init__()
        self.id = str(id)

    def add_handler(self, e):
        self.counter += 1
        self.text.value = str(self.counter)
```

```
self.update()

def build(self):
    self.counter = 0
    self.text = ft.Text(str(self.counter), width=30, size=20,
color="pink600")
        return ft.Row([self.text, ft.ElevatedButton(f"Aθροιστής αρ.
{self.id}", on_click=self.add_handler)])

def main(page):
    page.add(Counter(1), Counter(2), Counter(3))

ft.app(target=main)
```

## 13. Ανάκτηση και αποστολή δεδομένων στον εξυπηρετητή

Μέχρι τώρα έχουμε δει πώς να αναπτύσσουμε μια εφαρμογή με χρήστη του πλαισίου Flet είτε στον προσωπικό υπολογιστή είτε σε ένα φυλλομετρητή. Στη συνέχεια θα δούμε πως μια εφαρμογή Flet μπορεί να επικοινωνήσει με μια εξωτερική πηγή δεδομένων είτε για να ανακτήσει δεδομένα, είτε για να στείλει δεδομένα (πχ. το αποτέλεσμα του τεστ). Η λύση είναι να επικοινωνήσει ο φυλλομετρητής με τον εξυπηρετητή μέσω του πρωτοκόλλου HTTP.

Μια βιβλιοθήκη που χρησιμοποιείται ευρύτατα για επικοινωνία με ένα εξυπηρετητή μέσω HTTP είναι η βιβλιοθήκη `requests`. Ας δούμε ένα πρώτο παράδειγμα ανάκτησης δεδομένων από ένα πόρο του διαδικτύου και στη συνέχεια θα επεκτείνουμε τη λύση με χρήση της Flet.

Θα χρησιμοποιήσουμε τη σελίδα "<https://icanhazdadjoke.com>" που όταν συνδεθούμε μαζί της επιστρέφει ανέκδοτα, τα περισσότερα επειδή στηρίζονται σε παιχνίδι με τις λέξεις στην Αγγλική γλώσσα, βοηθάει καλή γνώση της Αγγλικής για να τα καταλάβουμε, είναι συνήθως αυτά που λέμε "κρύα" ανέκδοτα.

Το πρώτο παράδειγμα κώδικα είναι το εξής:

```
import requests as req
jokes_url = "https://icanhazdadjoke.com"

def process_response(response, **kws):
    response = response.json()
    print(response['joke'])

def exception_handler(request, exception):
    print('Error in retrieving joke', request.url if request else "", exception)

while True:
    try:
        response = req.get(jokes_url, headers={'Accept': 'application/json'}, timeout=0.5)
        process_response(response)
    except Exception as error:
        exception_handler(None, error)
    more = input("more dad's jokes?")
    if more.lower() == "n": break
```

Στο παράδειγμα αυτό χρησιμοποιούμε τη μέθοδο `get` της `requests` για ανακτήσουμε πληροφορία από την ιστοσελίδα. Μάλιστα θέτουμε στην κεφαλίδα του αιτήματος την τιμή '`Accept`':

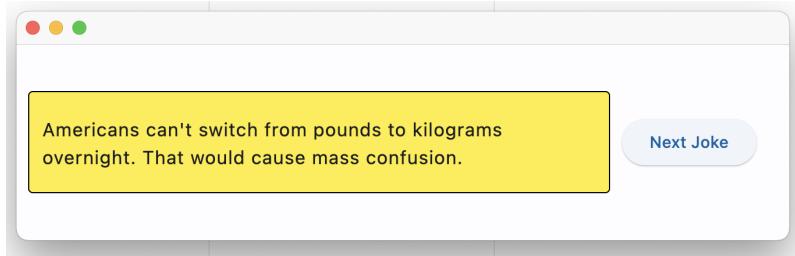
`'application/json'` ώστε να μάς επιστρέψει τα δεδομένα σε μορφή json που είναι δομή παρόμοια με τα λεξικά της Python. Επίσης ορίζουμε ένα `timeout` 0.5 sec. Να σημειωθεί ότι η κλήση στον πόρο του διαδικτύου είναι καλό να γίνεται μέσα σε μια δομή `try/except` ώστε σε περίπτωση αποτυχίας της σύνδεσης (πχ διακοπή του δικτύου), να μην προκαλείται σφάλμα στο πρόγραμμά μας.

Αν το αίτημα επιστρέψει σωστά τα δεδομένα, καλούμε τη συνάρτηση `process_response()`, η οποία μετατρέπει το μήνυμα σε δομή json με κλήση της μεθόδου `json()` και στη συνέχεια τυπώνουμε την τιμή του κλειδιού '`joke`' που είναι το ανέκδοτο. Αυτό το επαναλαμβάνει μέχρι ο χρήστης να τερματίσει το πρόγραμμα.

Όταν τρέξει το πρόγραμμα, παίρνουμε μια σειρά από ανέκδοτα κάθε φορά που πατάμε "Enter".

```
What did the digital clock say to the grandfather clock? Look, no
hands!
> more dad's jokes?
The invention of the wheel was what got things rolling
> more dad's jokes?
```

Έχει ενδιαφέρον στη συνέχεια να ενσωματώσουμε τη λειτουργία αυτή σε κώδικα flet, όπου ο χρήστης όταν πατάει ένα πλήκτρο, θα βλέπει ένα νέο ανέκδοτο, όπως στην παρακάτω εικόνα.



Προσπαθήστε μόνοι σας να δώσετε τη λύση, συνδυάζοντας όλα όσα έχουμε δει ως τώρα για τη flet, καθώς και τον κώδικα που είδαμε με χρήση της `requests` για σύνδεση http με τον πόρο [icanhazdadjoke.com](http://icanhazdadjoke.com).

Λύση του προβλήματος

Στη συνέχεια δίνεται μια λύση του προβλήματος. Δημιουργούμε μια συνάρτηση χειριστή του συμβάντος 'click', δηλαδή όταν ο χρήστης πατήσει το πλήκτρο "Νέο ανέκδοτο". Η συνάρτηση αυτή `next_joke_handler` εκτελεί το αίτημα, και όταν λάβει την απάντηση επιτυχώς, στέλνει την απάντηση στη συνάρτηση `process_response()`, η οποία εισάγει το νέο ανέκδοτο στο πεδίο κειμένου `joke_text`

```
import requests as req
import flet as ft

jokes_url = "https://icanhazad joke.com"

def main(page: ft.Page):
    page.vertical_alignment = ft.MainAxisAlignment.CENTER
    page.horizontal_alignment = ft.CrossAxisAlignment.STRETCH

    def next_joke_handler(e):
        def process_response(response, **kws):
            response = response.json()
            joke_text.value = response['joke']
            joke_text.update()
        def exception_handler(exception):
            print('Error in retrieving joke', exception)

        try:
            response = req.get(jokes_url, headers={'Accept': 'application/json'}, timeout=0.5)
            process_response(response)
        except Exception as error:
            exception_handler(error)

    joke_text = ft.TextField(bgcolor="YELLOW", width=500, max_lines=5)
    page.add(ft.Row([joke_text, ft.ElevatedButton(" Νέο ανέκδοτο ", on_click = next_joke_handler)], alignment=ft.CrossAxisAlignment.CENTER))

ft.app(target=main)
```

## 14. Επανασχεδίαση της εφαρμογής pythonquiz

---

Στην ενότητα αυτή θα χρησιμοποιήσουμε όλα όσα είδαμε στις προηγούμενες ενότητες για επανασχεδίαση της εφαρμογής pythonquiz χρησιμοποιώντας την τεχνολογία Flet/flutter αντί για HTML/CSS. Θα είναι μια ευκαιρία να συγκρίνουμε δύο αρκετά διαφορετικές προσεγγίσεις στην ανάπτυξη διαδικτυακών εφαρμογών. Επίσης με την ευκαιρία αυτή θα κάνουμε κάποιες επεκτάσεις στην προηγούμενη λύση, εισάγοντας το πεδίο κωδικός χρήστη, και συνεπώς το μηχανισμό κρυπτογράφησης και αποθήκευσης κωδικών χρήστη, ενώ θα χρησιμοποιήσουμε μια βάση δεδομένων sqlite για αποθήκευση των στοιχείων των χρηστών της εφαρμογής και του ιστορικού των τεστ που έχουν ολοκληρώσει, αυτό θα γίνει με χρήση της βιβλιοθήκης flask-sqlalchemy που μάς διευκολύνει στη σύνδεση με τους πίνακες της βάσης δεδομένων.

Αρχικά θα σχεδιάσουμε τις σελίδες της εφαρμογής χρησιμοποιώντας όλα όσα είδαμε ως τώρα.

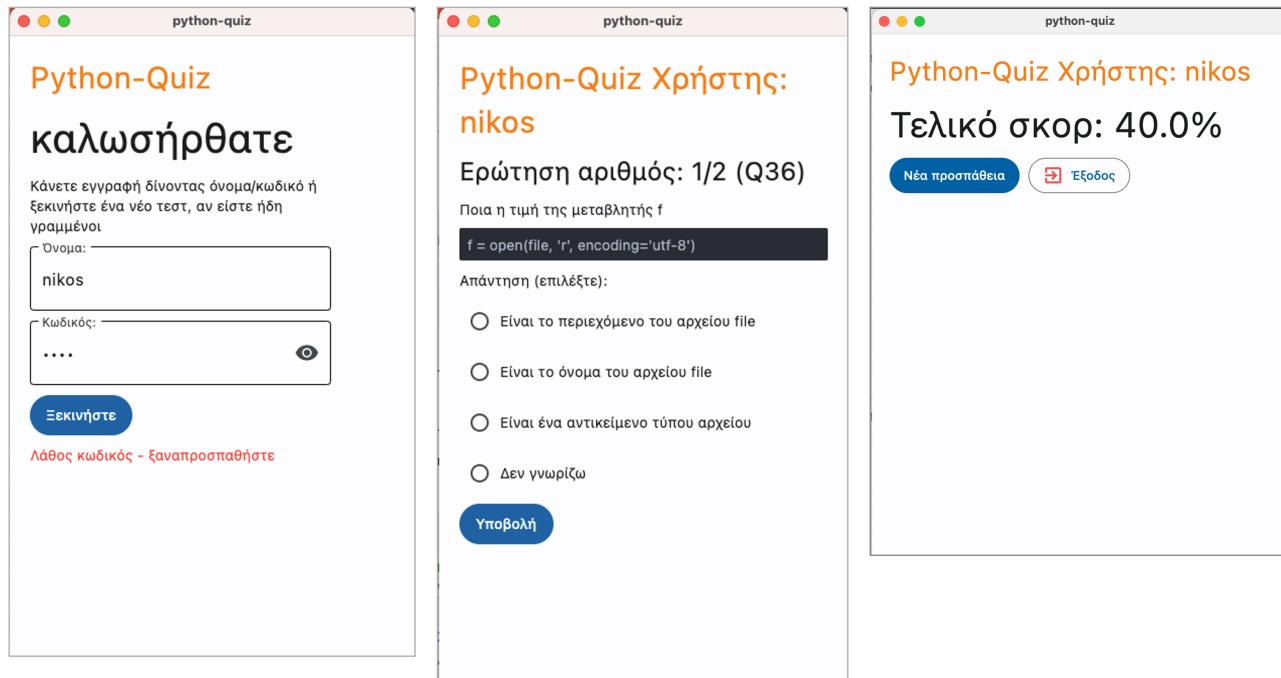
### Το πρόγραμμα πελάτης quiz-client

Μια καλή ιδέα σε μια εφαρμογή με αρκετό πλήθος σελίδων είναι να χρησιμοποιήσουμε για κάθε βασική οθόνη ένα γραφικό στοιχείο του χρήστη (userControl), το οποίο θα περιλαμβάνει όλα τα επί μέρους στοιχεία της οθόνης καθώς και τους χειριστές συμβάντων.

Στην εφαρμογή αυτή θα έχουμε τρεις βασικές οθόνες:

- Την αρχική οθόνη εισόδου/εγγραφής
- την οθόνη μιας τυπικής ερώτησης του ερωτηματολογίου που θα επαναλαμβάνεται για όσες ερωτήσεις περιέχει το τεστ
- την τελική οθόνη που δίνει την τελική επίδοση.

Μια τυπική εμφάνιση των οθονών αυτών φαίνεται στη συνέχεια:



Συνεπώς η εφαρμογή μας θα οργανώνεται σε 3 κύριες κλάσεις, οι οποίες αντιστοιχούν στις κύριες οθόνες. ([Login](#), [Quiz](#) και [End](#)).

```
class Login(ft.UserControl)

class Quiz(ft.UserControl)

class End(ft.UserControl)

class Controller()

def main(page: ft.Page):
    page.title = "python-quiz"
    Controller(page)

ft.app(target=main)
```

Επίσης δημιουργούμε μια κλάση [Controller](#) που θα έχει τον έλεγχο της εφαρμογής και θα κρατάει το σκορ της άσκησης, καθώς ο χρήστης απαντάει στα ερωτήματα.

Τέλος θα υπάρχει, όπως σε όλες τις εφαρμογές Flet, η συνάρτηση [main](#).

Ας δούμε πώς θα δημιουργήσουμε μια από τις κλάσεις αυτές, έστω την κλάση [Login](#). Όπως βλέπουμε στο σχήμα η οθόνη αυτή περιέχει κάποια αρχικά κείμενα και στη συνέχεια δύο πεδία στα οποία ο χρήστης καλείται να συμπληρώσει το όνομά-χρήστη του και τον μυστικό κωδικό του. Τέλος υπάρχει ένα πλήκτρο έναρξης του τεστ.

```
class Login(ft.UserControl):
    def __init__(self, controller):
        super().__init__(self)
        self.controller = controller

    def login_handler(self, event):
        print(self.name.value, self.passw.value)
        if not self.name.value or not self.passw.value:
            self.message.value = "Πρέπει να δώσετε το όνομα και τον
κωδικό σας"
            self.message.color = "red"
            self.update()
            return
        else:
            data = {"name": self.name.value, "passw":
self.passw.value}
        try:
            ## σύνδεση στον server και κλήση της οθόνης Quiz με το
τεστ
        except:
            self.show_message(None, msg="Αποτυχία σύνδεσης στο
python-quiz")
            return

    def show_message(self, event, msg=''):
        self.message.value = msg
        self.message.color = "red"
        self.update()
        return

    def build(self):
        self.controls = []
        self.controls.append(ft.Text(f"Python-Quiz",
color="yellow900", size=30))
        self.controls.append(ft.Text(f'καλωσήρθατε', size=40))
        self.controls.append(ft.Text(f"Κάνετε εγγραφή δίνοντας όνομα/
κωδικό ή ξεκινήστε ένα νέο τεστ, αν είστε ήδη γραμμένοι",
width='300'))
        self.name = ft.TextField(label='Όνομα:', hint_text="το όνομά
σας ...", width='300', on_focus=self.show_message)
        self.controls.append(self.name)
        self.passw = ft.TextField(label="Κωδικός:", password=True,
width='300', can_reveal_password=True)
        self.controls.append(self.passw)
        self.controls.append(ft.FilledButton("Ξεκινήστε", on_click=
self.login_handler))
        self.message = ft.Text("")
```

```
self.controls.append(self.message)
return ft.Column(self.controls)
```

Η κλάση αυτή που κληρονομεί από την κλάση `UserControl` έχει τη μέθοδο `build`, η οποία ομαδοποιεί τα στοιχεία της οθόνης σε ένα υποδοχέα (εδώ ένα στοιχείο `Column`) και επιστρέφει αυτό το αντικείμενο.

Επίσης η κλάση περιέχει τους χειριστές συμβάντων που μπορεί να κληθούν, όπως ο χειριστής του πατήματος του πλήκτρου, η μέθοδος `login_handler()`. Η μέθοδος αυτή θα ελέγξει πρώτα αν ο χρήστης έχει συμπληρώσει τα πεδία `name` και `passw`, αν όχι θα στείλει μήνυμα στην οθόνη μέσω του στοιχείου `self.message` (θα κληθεί η μέθοδος `show_message()`). Αν τα πεδία είναι συμπληρωμένα θα καλέσει τον εξυπηρετητή στέλνοντας του τα στοιχεία του χρήστη (με χρήστη της βιβλιοθήκης `requests`), ώστε να γίνει άλεγχος αν ο χρήστης υπάρχει και ο κωδικός είναι σωστός, αν όχι να δημιουργήσει νέο χρήστη. Να σημειωθεί ότι κάθε χρήστης θα πρέπει να έχει διαφορετικό όνομα-χρήστη.

Όταν αρχίζουμε να αναπτύσσουμε τις σελίδες δεν συμπληρώνουμε την κλήση στον εξυπηρετητή, γιατί το πρόγραμμα-πελάτης αναπτύσσεται ανεξάρτητα.

Στόχος είναι η κλάση `Controller` που δημιουργεί το στοιχείο αυτό, να το εμφανίσει στην οθόνη με την εντολή `self.page.add(Login(self))`, στη συνέχεια όταν πρόκειται να αντικατασταθεί στο παράθυρο του φυλλομετρητή η οθόνη αυτή από μια άλλη οθόνη, (πχ την πρώτη ερώτηση του τεστ), τότε ο `Controller` θα διώξει από την οθόνη τα στοιχεία που περιέχει και θα φορτώσει το νέο, με τις εντολές:

```
if self.page.controls: self.page.controls.pop()
self.page.add(Quiz(self))
```

Να σημειωθεί ότι ο `Controller`, στέλνει ως όρισμα των στιγμιότυπων των κλάσεων, αναφορά στον εαυτό του (`self`), ώστε αυτοί να έχουν πρόσβαση στις μεθόδους του.

Αν θέλουμε να συγκρίνουμε την κατασκευή ιστοσελίδων με χρήση της Flet σε σχέση με τον παραδοσιακό τρόπο HTML/CSS, εδώ όλες οι σελίδες είναι κλάσεις στο ίδιο πρόγραμμα και η παρουσίασή τους επίσης γίνεται μέσω των τιμών των γνωρισμάτων των γραφικών στοιχείων. Οι εντολές διάταξης του περιεχομένου της σελίδας, όπως οι `ft.Column`, `ft.Row` είναι πολύ πιο εύκολες στη χρήση από τις αντίστοιχες εντολές flexbox της CSS. Ο κώδικας είναι πολύ πιο συνοπτικός, και η μετάβαση από σελίδα σε σελίδα δεν εμπλέκει τον εξυπηρετητή, λογική Single Page Application (SPA).

Όπως θα δούμε στη συνέχεια και ο εξυπηρετητής θα γίνει πολύ πιο απλός, αφού τώρα θα εξυπηρετήσει πολύ λιγότερα σημεία δρομολόγησης (routes), για αποστολή δεδομένων: το σημείο σύνδεσης του χρήστη που επιστρέφει ένα νέο τεστ, το σημείο αποστολής της επίδοσης του χρήστη

σε ένα τεστ, καθώς και το σημείο αιτήματος ενός νέου τεστ. Τα δεδομένα μάλιστα που ανταλλάσσουν ο εξυπηρετητής με τον φυλλομετρητή είναι επίσης λιγότερα, αφού δεν επιστρέφει ιστοσελίδες, αλλά μόνο δεδομένα json.

Τέλος όπως παρατηρήσαμε, η εμφάνιση της σελίδας, χωρίς ιδιαίτερη μάλιστα προσπάθεια, είναι αισθητικά άρτια, αφού χρησιμοποιούνται τα γραφικά στοιχεία της βιβλιοθήκης flutter.

## Σύνδεση του πελάτη με τον εξυπηρετητή

Για τη σύνδεση του πελάτη με τον εξυπηρετητή χρησιμοποιείται η βιβλιοθήκη `requests` στον πελάτη, όπως συζητήσαμε σε προηγούμενη ενότητα. Αυτή είναι βιβλιοθήκη για επικοινωνία μέσω του πρωτοκόλλου HTTP. Από την πλευρά του εξυπηρετητή χρησιμοποιείται η βιβλιοθήκη Flask.

**Σημείωση:** Η λύση που προτείνεται εδώ είναι λειτουργική για την περίπτωση που η εφαρμογή Flet εκτελείται τοπικά είτε σε παράθυρο του λειτουργικού συστήματος είτε στον φυλλομετρητή, μέσω τοπικού εξυπηρετητή, όμως η λύση αυτή δεν μπορεί να λειτουργήσει, όταν ο πελάτης εκτελείται σε περιβάλλον Pyodide/ webassembly. Ο λόγος, όπως εξηγείται στη συνέχεια, σχετίζεται με περιορισμούς που υπάρχουν ακόμη στην εκτέλεση κάποιων βιβλιοθηκών της Python στο περιβάλλον Pyodide.

Καταγράφουμε αρχικά τα σημεία στα οποία η εφαρμογή μας χρειάζεται να επικοινωνήσει με τον εξυπηρετητή. Αυτά είναι τα εξής τρία:

(α) Το πρώτο σημείο αφορά στην αποστολή των στοιχείων του χρήστη μετά τη συμπλήρωση της οθόνης Login. Το αποτέλεσμα του αιτήματος αυτού είναι είτε ένα μήνυμα λάθους, στην περίπτωση που ο κωδικός χρήστη δεν αντιστοιχεί στο όνομα, είτε το τεστ που αποτελείται από μια λίστα με ερωτήσεις (σώμα ερώτησης, απαντήσεις, ένδειξη για το ποια ερώτηση είναι σωστή). Το τεστ αποστέλλεται αν τα στοιχεία αντιστοιχούν σε χρήστη που υπάρχει ήδη και το ζεύγος όνομα/κωδικός είναι σωστά ή αν πρόκειται για νέο χρήστη τα στοιχεία του οποίου αποθηκεύονται στη βάση δεδομένων.

(β) Το δεύτερο σημείο αφορά στην αποστολή προς τον εξυπηρετητή του αποτελέσματος του τεστ μετά την ολοκλήρωσή του. Το αποτέλεσμα θα πρέπει να συνοδεύεται από τα στοιχεία του συγκεκριμένου χρήστη, αφού το πρωτόκολλο HTTP δεν συντηρεί την κατάσταση της επικοινωνίας, άρα δεν "θυμάται" με ποιον χρήστη μίλησε πριν.

(γ) Το τρίτο σημείο αφορά το αίτημα για ένα ακόμη τεστ, από χρήστη που έχει ήδη δώσει τα διαπιστευτήριά του. Και σε αυτή την περίπτωση ο εξυπηρετητής επιστέφει ένα νέο τεστ, όμως το αίτημα διαφέρει από το (α) αφού δεν χρειάζεται να ταυτοποιηθεί ο χρήστης, ο οποίος είναι ήδη ταυτοποιημένος.

Στα παραπάνω σημεία θα καλέσουμε - με χρήστη της μεθόδου POST του πρωτοκόλλου HTTP - αντίστοιχα σημεία δρομολόγησης (routes) του εξυπηρετητή τα οποία θα πρέπει να αναμένουν σχετικά αιτήματα, όπως θα δούμε σε επόμενη ενότητα.

Πρέπει να προσέξουμε ιδιαίτερα το θέμα της διατήρησης της ταυτότητας του χρήστη, ώστε ο εξυπηρετητής όταν λάβει το αποτέλεσμα του τεστ να ξέρει σε ποιον χρήστη αντιστοιχεί. Αυτό γίνεται με χρήση του μηχανισμού διαχείρισης συνεδρίας (session), δηλαδή κρυπτογραφημένων δεδομένων που στέλνονται στον πελάτη, τα οποία επανα-στέλνονται πίσω σε κάθε νέο αίτημα του πελάτη, ταυτοποιώντας τον αποστολέα.

Ο μηχανισμός αυτός ενεργοποιείται όταν ο εξυπηρετητής επιβεβαιώσει την ταυτότητα του χρήστη κατά τη σύνδεση του, οπότε στέλνει προς τον πελάτη ένα ζεύγος μεταβλητής/τιμής μαζί με την απόκρισή του. Για παράδειγμα, στον εξυπηρετητή η σχετική εντολή μπορεί να είναι :

```
session["username"] = "Kostas"
```

Με την εντολή αυτή, ο εξυπηρετητής ενσωματώνει στην απόκρισή του τα δεδομένα "username" = "Kostas" (η τιμή κρυπτογραφημένη). Σημειώνεται σχετικά ότι θα πρέπει να έχει οριστεί το κλειδί κρυπτογράφησης στην εφαρμογή app του Flask, με μια εντολή όπως η παρακάτω:

```
app.config['SECRET_KEY'] = "a-very-secret-key"
```

Το αντικείμενο που στέλνει ο εξυπηρετητής είναι ένα SecureCookieSession. Όταν ο εξυπηρετητής επικοινωνεί με τον browser, η επιστροφή των δεδομένων του session από τον φυλλομετρητή γίνεται αυτόματα, αρκεί να είναι ενεργή η επιλογή αποθήκευσης cookies. Στο παράδειγμά μας, σε συνθήκες που η εφαρμογή Flet τρέχει ως τοπική εφαρμογή σε παράθυρο του λειτουργικού συστήματος, θα πρέπει να μεριμνήσουμε μόνοι μας για την διασφάλιση της συνεδρίας. Αυτό γίνεται μέσω του αντικειμένου Session της βιβλιοθήκης requests.

Δημιουργούμε αρχικά ένα στιγμιότυπο του Session:

```
session = requests.Session()
```

Και στέλνουμε τα αιτήματα στον εξυπηρετητή με χρήση της μεθόδου post() του session, όπως στο παράδειγμα:

```
result = session.post("http://127.0.0.1:5000/login", json = {"name": "nikos", "passw": "1234"}, timeout = 5)
```

Η διαχείριση του αποτελέσματος `result`, δηλαδή της απόκρισης που θα λάβει ο πελάτης στο αίτημά του, είναι αντικείμενο που θα συζητήσουμε στη συνέχεια.

Η πλήρης έκδοση της συνάρτησης `login_handler()` που ενεργοποιείται όταν ο χρήστης επιλέξει αποστολή των στοιχείων του, είναι η εξής:

```
def login_handler(self, event):
    ## εδώ στέλνουμε τα στοιχεία του χρήστη στον server και παίρνουμε το session['user'] και το ιστορικό του
    if not self.name.value or not self.passw.value: # δεν έχει συμπληρωθεί ένα από τα πεδία
        self.show_message(None, msg="Πρέπει να δώσετε το όνομα και τον κωδικό σας")
        return
    else:
        data = {"name": self.name.value, "passw": self.passw.value}
        try:
            result = session.post(server + "login", json = data,
timeout = 5)
            if result.status_code == 200 and not 'error' in result.json():
                self.controller.user = self.name.value
                self.controller.start_questions(result.json()) ## άρχισε το τεστ
            elif 'error' in result.json().keys():
                self.show_message(None, msg="Λάθος κωδικός – ξαναπροσπαθήστε")
                return
            else:
                self.show_message(None, msg="Αποτυχία σύνδεσης στο python-quiz")
                return
        except:
            self.show_message(None, msg="Αποτυχία σύνδεσης στο python-quiz")
        return
```

Με παρόμοιο τρόπο με το στοιχείο Login, αναπτύσσονται και τα υπόλοιπα στοιχεία της εφαρμογής, το Quiz και το End.

Τέλος θα πρέπει να γίνει ιδιαίτερη αναφορά στην κλάση Controller που είναι ο βασικός μηχανισμός ελέγχου της ροής της εφαρμογής.

Όταν δημιουργείται το στιγμιότυπο της κλάσης, καλείται η μέθοδος `start_login()`, η οποία φορτώνει το στοιχείο Login στη σελίδα, αφού καθαρίσει τη σελίδα από προηγούμενα στοιχεία:

```
if self.page.controls: self.page.controls.pop()
self.page.add(Login(self))
```

Το στοιχείο **Login**, αν ο χρήστης επιτύχει σύνδεση, καλεί τη μέθοδο **start\_questions()** του controller, όπως είδαμε ήδη στην παρουσίαση του Login. Η μέθοδος αυτή αναλαμβάνει να δημιουργήσει στιγμιότυπα της κλάσης **Quiz**, ένα για κάθε ερώτηση του τεστ. Αυτά φορτώνονται στη λίστα **questions\_user\_controls**. Στη συνέχεια καλείται για πρώτη φορά η μέθοδος **update\_question()**, η οποία φορτώνει στη σελίδα την πρώτη ερώτηση, που το παρουσιάζει το πρώτο στιγμιότυπο της κλάσης **Quiz**. Στη συνέχεια κάθε φορά που ο χρήστης απαντάει μια ερώτηση καλείται ξανά η μέθοδος αυτή, που όταν φτάσει στο τέλος των ερωτήσεων ενημερώνει τον εξυπηρετητή για το τελικό σκορ του χρήστη, και τότε φορτώνει το στοιχείο **End**.

Τέλος η κλάση **End** παρουσιάζει το τελικό σκορ και ρωτάει τον χρήστη αν θέλει να ξαναπροσπαθήσει ή να βγει. στην πρώτη περίπτωση καλείται η μέθοδος **new\_game()** του controller, ενώ στη δεύτερη η **restart()**.

Σημειώνεται ότι κατά την ανάπτυξη της εφαρμογής δεν είναι σκόπιμο να γίνονται κλήσεις στον εξυπηρετητή, γιατί αυτό θα αύξανε την πολυπλοκότητα και θα ήταν δύσκολη η εκσφαλμάτωση. Όποτε στα σημεία διεπαφής με τον εξυπηρετητή απλά τοποθετούνται δεδομένα, που υποκαθιστούν την απάντησή του.

## Ο εξυπηρετητής (server)

Όπως σε όλες τις διαδικτυακές εφαρμογές θα πρέπει να δημιουργήσουμε το δεύτερο σκέλος της εφαρμογής που είναι ο εξυπηρετητής των αιτημάτων του πελάτη της εφαρμογής μας.

Στην περίπτωση μας, και σε αντίθεση από μια πιο παραδοσιακή εφαρμογή με χρήστη των τεχνολογιών HTML/CSS/JS τα σημεία δρομολόγησης, όπως ήδη αναφέρθηκε, είναι λιγότερα και οι ενέργειες του εξυπηρετητή είναι πιο απλές, αφού θα πρέπει να ελέγξει τα δεδομένα και να απαντήσει με ανάλογα δεδομένα Json.

Στο προηγούμενο παράδειγμα είχαμε ως βάση δεδομένων απλά αρχεία κειμένου, τα **players.txt** (για αποθήκευση των παικτών και των τεστ που έχουν ήδη κάνει), και **questions.txt** (για αποθήκευση των ερωτήσεων). Επίσης στο προηγούμενο παράδειγμα δεν είχαμε υλοποιήσει αποθήκευση και έλεγχο κωδικού χρήστη.

Στο παράδειγμα αυτό συνεπώς θα δούμε τις εξής επεκτάσεις της αρχικής λύσης:

- (α) Θα συνδεθούμε με μια βάση δεδομένων sqlite3 για αποθήκευση των τεστ και των χρηστών (όνομα και κωδικός) με χρήση της βιβλιοθήκης **flask\_sqlalchemy**
- (β) Θα διαχειριστούμε τους μυστικούς κωδικούς των χρηστών με χρήση της βιβλιοθήκης **werkzeug.security** που επιτρέπει την κρυπτογράφησή τους ώστε να αποθηκευτούν με κρυπτογράφηση στη βάση δεδομένων.

Θα δούμε κάθε μια από αυτές τις περιπτώσεις ξεχωριστά στη συνέχεια.

---

## 15. SQLAlchemy - σύνδεση στη βάση δεδομένων

Η βιβλιοθήκη `flask_sqlalchemy` είναι μια επέκταση του Flask χρήσιμη για διασύνδεση μέσω της SQLAlchemy με μια οποιαδήποτε σχεσιακή βάση δεδομένων SQL. Η `SQLAlchemy` είναι μια βιβλιοθήκη της Python για διασύνδεση με σχεσιακές βάσεις δεδομένων. Πληροφορίες για τη βιβλιοθήκη αυτή θα βρείτε στις [σελίδες τεκμηρίωσης της Flask-SQLAlchemy](#).

Η εγκατάστασή της απαιτείται: `pip install Flask-SQLAlchemy`.

Στην περίπτωση μας θα χρησιμοποιήσουμε τη βάση δεδομένων SQLite3 η οποία είναι ίσως η πιο απλή περίπτωση σχεσιακής βάσης δεδομένων, αφού αποθηκεύεται σε ένα απλό αρχείο στο δίσκο μας και δεν απαιτεί εξυπηρετητή, ενώ δεν διαθέτει σύνθετους μηχανισμούς ελέγχου πρόσβασης του χρήστη.

Η βασική ιδέα της SQLAlchemy είναι η δημιουργία μοντέλων που αντιπροσωπεύουν τις οντότητες της βάσης δεδομένων μας (αυτή είναι η ιδέα των Object Relational Mappers, ORM). Για το παράδειγμα μας τα μοντέλα θα αφορούν τις οντότητες User και Game.

Τα δύο αυτά μοντέλα ορίζονται ως εξής:

```
from werkzeug.security import generate_password_hash,
check_password_hash
import datetime

app = Flask(__name__)

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///quiz.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = True

db = SQLAlchemy(app)
class User(db.Model):
    username = db.Column(db.String(80), primary_key=True)
    password = db.Column(db.String(250), nullable=False)

    def __repr__(self):
        return f'<User {self.username}>'

class Game(db.Model):
    name = db.Column(db.String(80), db.ForeignKey('user.username'),
primary_key=True)
    when = db.Column(db.String(80), primary_key=True)
    score = db.Column(db.Float(), nullable=False)
```

```
def __repr__(self):
    return f'<Game {self.name}-{self.when}>'
```

Παρατηρούμε από το παράδειγμα ότι ορίζουμε τα γνωρίσματα κάθε οντότητας, καθώς και αν αυτά είναι υποχρεωτικά (`nullable=False`), αν ειναι το πρωτεύον κλειδί που λαμβάνει μοναδικές τιμές για κάθε εγγραφή (`primary_key=True`), ενώ ορίζουμε τη διασύνδεση των μοντέλων μέσω του μηχανισμού του ξένου κλειδιού (`db.ForeignKey('user.username')`). Επίσης για κάθε γνώρισμα, ορίζουμε τους τύπους δεδομένων του, που αντιστοιχούν στους τύπους δεδομένων που συαντάμε σε βάσεις δεδομένων SQL, (`db.String(80)` συμβολοσειρά 80 χαρακτήρων), άλλοι τύποι δεδομένων είναι `BigInteger`, `Boolean`, `Date`, `DateTime`, `Enum`, `Double`, `Float`, `Integer`.

Στη συνέχεια εισάγουμε δεδομένα στη βάση δεδομένων με χρήση του μοντέλου ως εξής:

```
name = 'zeon'
password = '5678'
user = User.query.filter_by(username=name).first()
if user:
    print(f'user with name {name} already exists')
else:
    new_user = User(username=name, password=password)
    try:
        db.session.add(new_user)
        db.session.commit()
    except Exception as error:
        print('Error in creating user {name}', error)
```

Στο παράδειγμα αυτό έστω ότι επιθυμούμε να εισάγουμε τον χρήστη με όνομα `zenon` και κωδικό `'5678'` με χρήση του μοντέλου `User`. Θα κάνουμε πρώτα έλεγχο αν υπάρχει χρήστης με αυτό το όνομα, με χρήση της μεθόδου `query.filter_by(κριτήριο)`, αν δεν υπάρχει τότε εισάγουμε τον χρήστη αφού δημιουργήσουμε ένα στιγμιότυπο του μοντέλου `new_user`.

Αντίστοιχα αν θέλουμε να εισάγουμε ένα παιχνίδι του χρήστη `zenon` στο οποίο πέτυχε σκορ 0.8, αυτό θα γίνει ως εξής:

```
import datetime

name = "zenon"
score = 0.8
when = datetime.datetime.now().strftime('%d-%m-%y %a %H:%M:%S')
new_game = Game(name=name, when=when, score=score)
try:
    db.session.add(new_game)
    db.session.commit()
```

```
print(f'το τεστ του {name} αποθηκεύτηκε')
except Exception as error:
    print(f'σφάλμα στην αποθήκευση του τεστ του {name}', error)
```

Παρατηρούμε και εδώ ότι με αντίστοιχο τρόπο δημιουργούμε ένα στιγμιότυπο του μοντέλου Game

Θα πρέπει να σημειωθεί ότι τα παραπάνω παραδείγματα θα πρέπει να ελεγχθούν μέσω ενός test\_db.py αρχείου που υλοποιεί όλες τις διασυνδέσεις με τη βάση δεδομένων, όπως εισαγωγή νέου χρήστη, έλεγχος κωδικού, εισαγωγή νέου τεστ, κλπ., χωρίς εμπλοκή του κώδικα πελάτη.

Τέλος να σημειωθεί ότι η βάση δεδομένων που δημιουργείται μπορεί να ελεγχθεί είτε από τη γραμμή εντολών είτε με χρήση γραφικής διεπαφής όπως η εφαρμογή **Db Browser for SQLite** που μπορείτε να κατεβάσετε στον υπολογιστή σας.

Στην ενότητα αυτή παρατηρήσαμε ότι εισαγάγαμε τα στοιχεία ενός χρήστη χωρίς κρυπτογράφηση του μυστικού κωδικού: `db.session.add({"username": "Zeon", "password": "5678"})`. Αυτή είναι πραγματικά κακή ιδέα και **πηγή κινδύνων**. Αν κάποιος αποκτήσει πρόσβαση στη βάση δεδομένων μας θα έχει πρόσβαση στους μυστικούς κωδικούς των χρηστών μας.

Στην επόμενη ενότητα θα προσπαθήσουμε να αντιμετωπίσουμε το πρόβλημα αυτό με κρυπτογράφηση των κωδικών.

---

## 16. Κρυπτογράφηση μυστικών κωδικών

Το θέμα της ασφάλειας της πληροφορίας που διαχειρίζόμαστε σε μια διαδικτυακή εφαρμογή είναι πολύ σοβαρό και βεβαίως δεν μπορεί να καλυφθεί στις σύντομες αυτές σημειώσεις.

Όμως περισσότερο για ευαισθητοποίηση στο θέμα αυτό, θα κάνουμε μια σύντομη αναφορά αντιμετώπισής του με χρήση της βιβλιοθήκης `werkzeug.security`. Η βιβλιοθήκη αυτή αποτελεί τμήμα της βιβλιοθήκης `Werkzeug` (εργαλείο στα Γερμανικά) που είναι μια πλήρης βιβλιοθήκη υλοποίησης του πρωτοκόλλου WSGI (Web Server Gateway Interface), του πρωτοκόλλου διασύνδεσης εφαρμογών Python με web servers. Θα πρέπει να σημειωθεί ότι το `Flask` χρησιμοποιεί τη βιβλιοθήκη `Werkzeug` για σύνδεση με τον εξυπηρετητή ιστού, ενώ υπάρχουν αρκετοί εξυπηρετητές ιστού που μπορούν να επικοινωνήσουν με μια εφαρμογή Flask (Apache HTTP Server με `mod_wsgi`, Nginx με `uWSGI`, Gunicorn, κλπ.).

Από τη βιβλιοθήκη `werkzeug.security` θα χρησιμοποιήσουμε τις εξής δύο μεθόδους:

- `generate_password_hash(password, method='pbkdf2:sha256', salt_length=8)`: Αυτή η μέθοδος επιστρέφει τον κρυπτογραφημένο κωδικό που προκύπτει από ασφαλή κατακερματισμό του κωδικού χρησιμοποιώντας μια καθορισμένη μέθοδο (η προεπιλογή είναι PBKDF2 με SHA-256) και το μήκος αλατιού, δλδ. πλήθος επαναλήψεων. Ο προκύπτων κατακερματισμός (κρυπτογράφηση κωδικού) μπορεί να αποθηκευτεί σε μια βάση δεδομένων για την επαλήθευση του κωδικού πρόσβασης αργότερα. Για παράδειγμα όταν καλέσουμε τη συνάρτηση αυτή για τον κωδικό που έδωσε ο χρήστης παράγει το εξής αποτέλεσμα που θα αποθηκευτεί στη βάση δεδομένων

```
hashed_password = generate_password_hash("5678")
print(hashed_password)
pbkdf2:sha256:26000$8P8IcHXLsfUXgY3P$38d6abbd32863c80d92fd6db96476447
42eaeb3cf9a31d3efc096a07526dc785
```

- `check_password_hash(κρυπτογραφημένος-κωδικός, κωδικός-που-έδωσε-ο-χρήστης)`: Αυτή η μέθοδος ελέγχει αν ένας κωδικός πρόσβασης που έδωσε ο χρήστης ταιριάζει με τον κατακερματισμό που παράγεται από την `generate_password_hash()` που αποθηκεύτηκε στη βάση δεδομένων. Επιστρέφει `True` αν ο κωδικός πρόσβασης ταιριάζει και `False` σε αντίθετη περίπτωση.

Με βάση τα παραπάνω η αποθήκευση των κωδικών που δίνουν οι χρήστες γίνεται ως εξής:

```
name = "zeon"
password = "1234"
password_hash = generate_password_hash(password)
```

```
new_user = User(username=name, password=password_hash)
try:
    db.session.add(new_user)
    db.session.commit()
except Exception as error:
    print('Error in creating user {name}', error)
```

Αντίστοιχα ο έλεγχος εγκυρότητας που δίνει ο χρήστης γίνεται ως εξής:

```
name = 'xenon'
password = '1234'
try:
    user = User.query.filter_by(username=name).first()
    print(user);
    if user and check_password_hash(user.password, password):
        print(f"Καλωσήλθες {name}!!!")
    else: print('Εσφαλμένο όνομα/κωδικός χρήστη,
ξαναπροσπαθήστε!')
except Exception as error:
    print('σφάλμα ανάγνωσης', error)
```

## 17. Ο εξυπηρετητής της εφαρμογής μας

Το τελευταίο βήμα πριν την ολοκλήρωση και έλεγχο καλής λειτουργίας είναι η δημιουργία της εφαρμογής Flask που θα εξυπηρετεί τα αιτήματα του πελάτη Flet.

Έχουμε ήδη αναφέρει ότι χρειάζεται να εξυπηρετήσουμε τρία σημεία δρομολόγησης αιτημάτων:

- το σημείο "/login"
- το σημείο "/end"
- το σημείο "/newgame"

Ο σκελετός της εφαρμογής του εξυπηρετητή είναι ο εξής:

```
from flask import Flask
from flask import redirect, url_for
from flask import request, session, json
from werkzeug.security import generate_password_hash,
check_password_hash
from flask_sqlalchemy import SQLAlchemy
import datetime
import pythonquiz as quiz
quiz.load_quiz() # φόρτωμα των ερωτημάτων του τεστ
app = Flask(__name__)
app.config['SECRET_KEY'] = "a-very-secret-key"

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///quiz.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = True

db = SQLAlchemy(app)

## ορισμός των μοντέλων του SQLAlchemy

## σημεία δρομολόγησης

@app.route("/login", methods=['POST'])
...

@app.route("/end", methods=["POST"])
...

@app.route("/newgame", methods=["POST"])
...

if __name__ == "__main__":
    app.run(debug=True)
```

Θα περιγράψουμε ένα από τα σημεία αυτό που αντιστοιχεί στο σημείο δρομολόγησης "/login".

```
@app.route("/login", methods=['POST'])
def login():
    def send_quiz():
        questions = quiz.draw_questions() # επιλογή η ερωτήσεων
        the_quiz = [quiz.show_question(q) for q in questions]
        session["username"] = name
        print("session is...", session)
        return jsonify(the_quiz)

    print("ROUTE /login")
    name = request.json["name"]
    password_hash = generate_password_hash(request.json["passw"])
    # πρώτα ελέγχουμε αν ο χρήστης υπάρχει με αυτό το όνομα
    user_existing = User.query.filter_by(username=name).first()
    if user_existing: # εφόσον υπάρχει ο χρήστης ... ελέγχουμε αν ο
        κωδικός είναι σωστός
        check_password = check_password_hash(user_existing.password,
request.json["passw"])
        if check_password:
            return send_quiz() # στέλνουμε το τεστ στον χρήστη
        else:
            return jsonify({"error": "Λάθος κωδικός"})
    else:
        password_hash = generate_password_hash(request.json["passw"])
        new_user = User(username=name, password=password_hash)
        db.session.add(new_user)
        db.session.commit()
        return send_quiz()
```

Αρχικά στις μεταβλητές name και password\_hash εκχωρούνται οι τιμές που έδωσε ο χρήστης (η δεύτερη μετά από εφαρμογή της συνάρτησης κατακερματισμού). Να σημειωθεί ότι εφαρμόζεται η μέθοδος json() για ανάκτηση των δεδομένων από το αντικείμενο request που περιέχει το αίτημα, ώστε να μετατραπούν από συμβολοσειρά σε δομή Json.

Στη συνέχεια ελέγχουμε αν υπάρχει ήδη χρήστης με το ίδιο όνομα στη βάση δεδομένων. (a) Αν υπάρχει, κάνουμε ένα ακόμη έλεγχο αν ο κώδικος που έδωσε αντιστοιχεί στον κωδικό που έχει αποθηκευτεί στη βάση δεδομένων (χρήση της μεθόδου check\_password\_hash()). Αν ναι, καλείται η συνάρτηση send\_quiz(), η οποία δημιουργεί το αντικείμενο the\_quiz που είναι μια λίστα με λεξικά, το καθένα με τα στοιχεία μιας ερώτησης του τεστ. Εκτελεί σειριοποίηση της δομής αυτής με την μέθοδο json.jsonify() και το αποτέλεσμα στέλνεται ως απόκριση στον χρήστη. Μια ακόμη βασική λειτουργία της συνάρτησης είναι να εισάγει τα δεδομένα του χρήστη στο αντικείμενο session.

(β) Σε περίπτωση που ο χρήστης υπάρχει αλλά ο κωδικός είναι εσφαλμένος επιστρέφει το αντικείμενο `json.jsonify({"error": "λάθος κωδικός"})`

(γ) Τέλος στην περίπτωση που ο χρήστης δεν υπάρχει, τότε δημιουργείται ένας χρήστης με τα στοιχεία που έδωσε ο χρήστης και στη συνέχεια καλείται η συνάρτηση `send_quiz()` όπως στην περίπτωση (α).

## Δημοσίευση της εφαρμογής σε εξυπηρετητή

Το τελευταίο στάδιο της ανάπτυξης της εφαρμογής, αφού έχει ολοκληρωθεί ο έλεγχος καλής λειτουργίας, είναι η ανάπτυξή της σε εξυπηρετητή που υποστηρίζει δημόσια πρόσβαση σε αυτήν. Υπάρχουν αρκετοί πάροχοι τέτοιων υπηρεσιών που χρησιμοποιούν υπηρεσίες νέφους για τον σκοπό αυτό. Για μικρές εφαρμογές σε πειραματικό στάδιο, όπως είναι φοιτητικά πρότζεκτ στο πλαίσιο εκπαίδευσης, με μικρές ανάγκες σε υπολογιστικούς πόρους, οι υπηρεσίες παρέχονται δωρεάν, ενώ για εμπορική χρήση ή για εφαρμογές με μεγάλη χρήση, δίνονται με κάποια χρέωση. Ενδεικτικά παραδείγματα είναι το [Heroku](#) (το οποίο όμως κατήργησε πρόσφατα την δωρεάν έκδοση, αν και μπορεί να την αιτηθεί κάποιος αν είναι σπουδαστής), η [fly.io](#) και η [railway.app](#). Επίσης υπάρχουν πάροχοι που υποστηρίζουν μόνο παροχή στατικών σελίδων, όπως το GitHub, που μπορεί κάποιος να συνδέσει με το αποθετήριο του κώδικά του.

Η δημοσίευση του εξυπηρετητή ακολουθεί τα βήματα που έχουν ήδη αναφερθεί στην ενότητα 7 για δημοσίευση στην πλατφόρμα *heroku*. Μια εναλλακτική επιλογή για φιλοξενία του εξυπηρετητή είναι η πλατφόρμα [pythonanywhere](#). Μάλιστα η διαδικασία είναι πολύ πιο απλή στην περίπτωση αυτή.

Όσον αφορά τη φιλοξενία του πελάτη, υπάρχουν αρκετές επιλογές για φιλοξενία στατικών σελίδων, μεταξύ των οποίων οι πιο γνωστές είναι οι netlify, replit, github pages, κλπ. Η replit είναι μάλιστα ιδιαίτερα απλή, αφού απαιτεί μόνο drop των σχετικών αρχείων στο φάκελο.

Η διαδικασία της δημοσίευσης της σελίδας Flet περιλαμβάνει αρχικά ένα βήμα μετασχηματισμού του κώδικα python στα απαραίτητα αρχεία html, javascript και css που περιέχουν τον κώδικα python μέσω της `pyodide` που είναι μια διανομή της Python που επιτρέπει να εκτελείται ως webassembly κώδικας Python στον browser.

```
% flet publish main.py
```

Από την εντολή αυτή δημιουργείται ο φάκελος `dist` ο οποίος περιέχει όλα τα απαραίτητα αρχεία :

- app.tar.gz
- main.py
- requirements.txt
- index.html
- manifest.json
- version.json

- python.js
- python-worker.js
- main.dart.js
- flutter.js
- flutter\_service\_worker.js
- favicon.png και οι φάκελοι assets, icons, κλπ.

Μπορούμε να ελέγξουμε το αποτέλεσμα με την παρακάτω εντολή:

```
% python3 -m http.server --directory dist
```

Στη συνέχεια στη διεύθυνση **localhost:8000** μπορούμε να τρέξουμε την εφαρμογή μας.

### Σημείωση:

Υπάρχουν όμως, όπως ήδη αναφέρθηκε, κάποιοι σοβαροί περιορισμοί που σχετίζονται με το περιβάλλον **Pyodide**. Ένας περιορισμός αφορά τις βιβλιοθήκες Python που μπορούν να εκτελεστούν στο περιβάλλον αυτό. Για παράδειδμα επειδή το περιβάλλον του φυλλομετρητή δεν επιτρέπει για λόγους ασφαλειας να κληθεί η διεπαφή sockets, η βιβλιοθήκη **requests** δεν μπορεί να συνδεθεί με εξωτερικούς πόρους, και συνεπώς ο πελάτης Flet δεν μπορεί να χρησιμοποιήσει τη βιβλιοθήκη αυτή για σύνδεση με τον εξυπηρετητή σε αυτή την περίπτωση. Έχουν προταθεί εναλλακτικές λύσεις, όπως η χρήση της διεπαφής fetch της javascript, όμως το θέμα αυτό δεν έχει ακόμη πλήρως αντιμετωπιστεί ώστε να είναι δυνατή η διασύνδεση του πελάτη Flet με διεπαφή REST με ένα εξυπηρετητή Flask με σχετικά απλό τρόπο.

---

# 18. Ανακεφαλαίωση: ο κύκλος ανάπτυξης μιας εφαρμογής full-Python stack.

---

Ανακεφαλαιώνοντας, τα βήματα που ακολουθήσαμε για να αναπτύξουμε μια εφαρμογή με τεχνολογία full-Python stack (Flet/Flask) (Ενότητες 8-17) ήταν τα εξής:

**Βήμα 1. Ανάπτυξη της εφαρμογής πελάτη (front-end),** έστω quiz-client.py, με χρήση της βιβλιοθήκης flet. Στα σημεία που η εφαρμογή απαιτεί πρόσβαση σε βάση δεδομένων είτε για ανάκτηση, είτε για αποθήκευση δεδομένων (σημεία σύνδεσης με τον εξυπηρετητή) αντικαθιστούμε τη σύνδεση του εξυπηρετητή (ο οποίος δεν έχει αναπτυχθεί ακόμη) με ενδεικτικά δεδομένα που θα μπορούσε να στείλει ο εξυπηρετητής στο σημείο εκείνο.

**Βήμα 2. Κατασκευάζουμε τη βάση δεδομένων** που θα εξυπηρετήσει την εφαρμογή μας. Ορίζουμε τις βασικές οντότητες, και τα γνωρίσματά τους, και δημιουργούμε τα μοντέλα του συστήματος ORM που θα αποτελέσουν διεπαφή με τα δεδομένα. Στη συνέχεια δημιουργούμε μια **δοκιμαστική εφαρμογή ελέγχου της πρόσβασης στη βάση δεδομένων** (πχ. εφαρμογή test-db.py) που αλληλεπιδρά με τη βάση δεδομένων εκτελώντας ενδεικτικές πράξεις που περιμένουμε από την εφαρμογή μας. Ελέγχουμε τα αποτελέσματα.

**Βήμα 3. Κατασκευάζουμε την εφαρμογή του εξυπηρετητή Flask** (server.py), χρησιμοποιώντας κώδικα από την test-db.py. Αναπτύσσουμε ένα-προς-ένα τα σημεία δρομολόγησης. Ελέγχουμε την λειτουργία του εξυπηρετητή με μια εφαρμογή πελάτη (πχ. test-client.py) που απλώς γεννάει τυπικά ερωτήματα και τα στέλνει μέσω της βιβλιοθήκης requests και τυπώνει την απάντηση που λαβαίνει, ώστε να ελέγχουμε τα σημεία δρομολόγησης του εξυπηρετητή μας.

**Βήμα 4. Ολοκλήρωση της εφαρμογής server.py με με τον πελάτη flet που αναπτύξαμε στο βήμα 1.** Για να γίνει αυτό συμπληρώνουμε την εφαρμογή client.py στα σημεία σύνδεσης με τη βάση χρησιμοποιώντας κώδικα από τη εφαρμογή πελάτη test-client.py, που χρησιμοποιήσαμε στο βήμα 43.

**Βήμα 5. Αναπτύσσουμε και τον πελάτη και τον εξυπηρετητή μας σε ένα πάροχο** (στον ίδιο ή διαφορετικούς). Πρώτα δημοσιεύουμε τον εξυπηρετητή, ελέγχουμε την καλή του λειτουργία με χρήση της δοκιμαστικής εφαρμογής test-db.py και αφού εξασφαλίσουμε ότι όλα τα σημεία δρομολόγησης ανταποκρίνονται ικανοποιητικά, τροποποιούμε τον πελάτη ώστε αυτός να συνδέεται στη δημόσια διεύθυνση του εξυπηρετητή μας. Τέλος ανεβάζουμε τον πελάτη σε πάροχο υπηρεσιών φιλοξένειας και χρησιμοποιούμε το URL που μας επιστρέφει ως τη διεύθυνση της εφαρμογής μας. Επαναλαμβάνουμε όλα τα σενάρια ελέγχου στη δημοσιευμένη εφαρμογή για να εξετάσουμε αν λειτουργεί ικανοποιητικά.

---

## 19. Άλλες πηγές

---

Οι τεχνολογίες που συζητήθηκαν στις παραπάνω ενότητες δεν είναι δυνατόν να αναπτυχθούν και να γίνουν κατανοητές σε βάθος στον περιορισμένο χρόνο μιας διάλεξης, είναι αναπόφευκτο ότι κάνουμε μια σύντομη εποπτική παρουσίαση ενός όμως πολύ σημαντικό πεδίου της τεχνολογίας.

Ο στόχος της συγκεκριμένης διάλεξης είναι να μάς δείξει μέσα από παραδείγματα υλοποίησης μιας σχετικά απλής εφαρμογής τις δυνατότητες που έχουμε να χρησιμοποιήσουμε το διαδίκτυο για να διαθέσουμε την εφαρμογή μας σε ένα ευρύτερο κοινό, και να σχεδιάσουμε πιο εύχρηστη αλληλεπίδραση του χρήστη με αυτή.

Αν κάποιοι έχουν ιδιαίτερο ενδιαφέρον για την περιοχή αυτή, μπορείτε να καταφύγετε σε τρια διαδικτυακά μαθήματα που έχουμε αναπτύξει στην πλατφόρμα <http://mathesis.cup.gr>

- Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTTP/CSS/JavaScript
- Προχωρημένα θέματα ανάπτυξης ιστοσελίδων
- Ανάπτυξη διαδικτυακών εφαρμογών με node.js/express.js

Αν κάποιοι θα ήθελαν να προχωρήσουν παραπέρα, θα μπορούσαν να παράγουν βελτιωμένες εκδόσεις της εφαρμογής `pythonquiz` που αναπτύξαμε στο παράδειγμα, μια πιθανή κατεύθυνση είναι η **αυθεντικοποίηση του χρήστη**, υπάρχουν σχετικές βιβλιοθήκες, όπως η `Flask-login`, που μάς βοηθούν σε αυτή την κατεύθυνση. Στο παράδειγμα (β) είδαμε μια τέτοια υλοποίηση.

Μια άλλη κατεύθυνση είναι να εμβαθύνουμε στη **βάση δεδομένων**. Εδώ μπορείτε να χρησιμοποιήσετε το Βιβλίο [Python Εισαγωγή στους Υπολογιστές](#), στο κεφάλαιο 15 "Μόνιμη αποθήκευση δεδομένων", όπου υπάρχει μια εισαγωγή στην SQLite3 καθώς και στην MongoDB, και πώς μπορείτε να συνδέσετε μια εφαρμογή Python με αυτές.

Επίσης στο Τμήμα ΗΜΤΥ, στο 5ο έτος, διδάσκεται ένα μάθημα με αντικείμενο ακριβώς αυτές τις τεχνολογίες "Προγραμματισμός Διαδικτύου", μπορείτε να κατεβάσετε το σχετικό υλικό από το eclass.

Τέλος όσοι ενδιαφέρονται μπορούν να αναλάβουν εργασίες στο θέμα αυτό, στο πλαίσιο του μαθήματος *Εισαγωγή στην Επιστήμη του Ηλεκτρολόγου Μηχανικού*, είναι ευκαιρία να βάλουν σε πράξη αφενός τις γνώσεις Python που αποκτήσατε στο προηγούμενο εξάμηνο, αφετέρου τα νέα στοιχεία που έχουν εισαχθεί σε αυτή τη διάλεξη. Στη φετεινή χρονιά δίνεται μάλιστα έμφαση στο παράδειγμα (β) Flet/flutter.