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 .

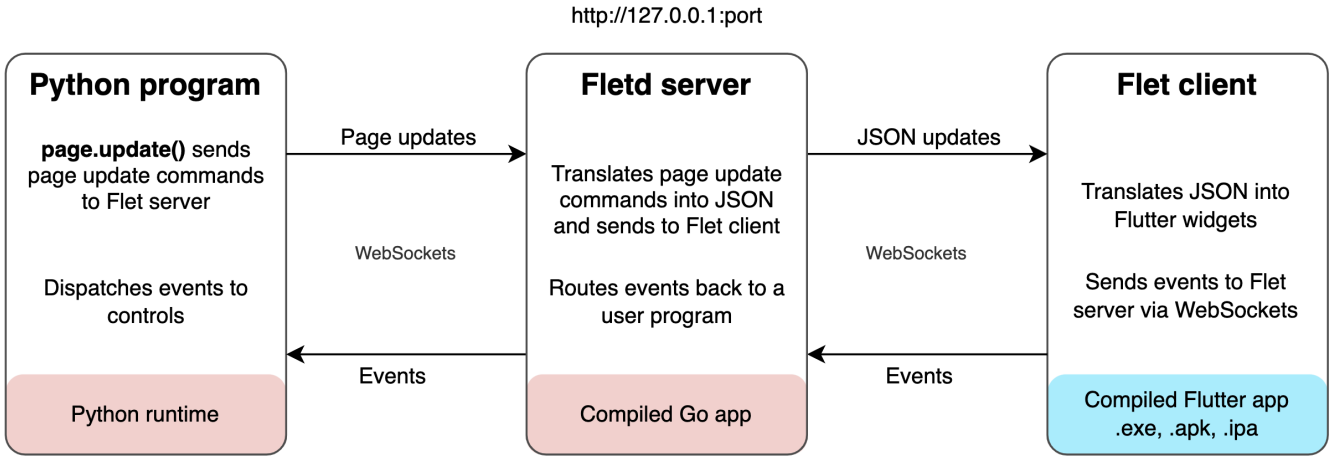
Το μοντέλο αυτό στηρίζεται στο πλαίσιο Flet, που επιτρέπει την ανάπτυξη στατικών σελίδων, δηλαδή σελίδων που δεν χρειάζονται φόρτωμα δεδομένων, όπως είδαμε στο παράδειγμα της εφαρμογής στις προηγούμενες ενότητες (που το περιεχόμενο της σελίδας καθοριζόταν σε πραγματικό χρόνο, με χρήση του template Jinja2).

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 πλαίσιου 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 ειναι μια εφαρμογή Flutter.

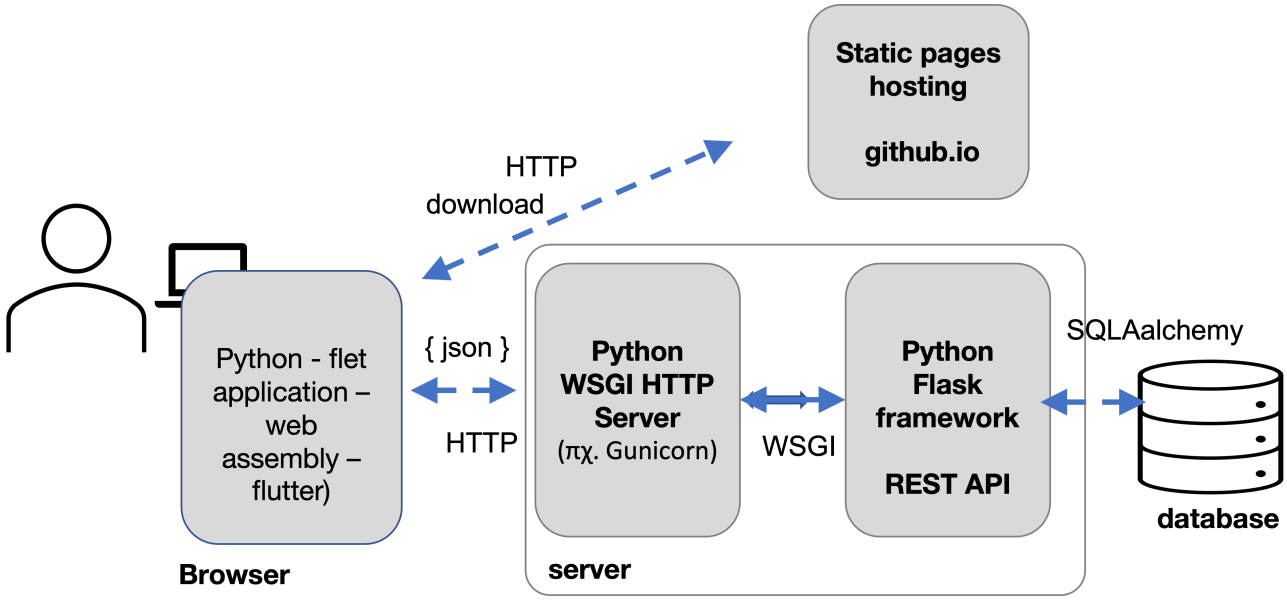
Από το περιβάλλον Flet μπορούμε να αναπτύξουμε εφαρμογές για το λειτουργικό σύστημα του υπολογιστή μας, αλλά και διαδικτυακές εφαρμογές ιστού.

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, οπως η tkinter, και άλλες.

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 αλληλεπιδρούν με τον χρήστη. Για το τμήμα αυτό θα κάνουμε χρήση του πλαισίου Flet. Οι

σελίδες αυτές ακολουθούν το μοντέλο εφαρμογής Signle 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 0.4, και η βιβλιοθήκη είναι σε συνεχή ανάπτυξη.

```
% pip3 install flet
Collecting flet
Downloading flet-0.4.2-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 βολικό 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 νέο 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 δημιουργήσετε 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 η εμφάνιση και η δομή της σελίδας αναμειγνύονται στον ίδιο κώδικα, και δεν ξεχωρίζουν σε κώδικα HTML και CSS, όπως γίνεται στο παραδοσιακό μοντέλο.

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.

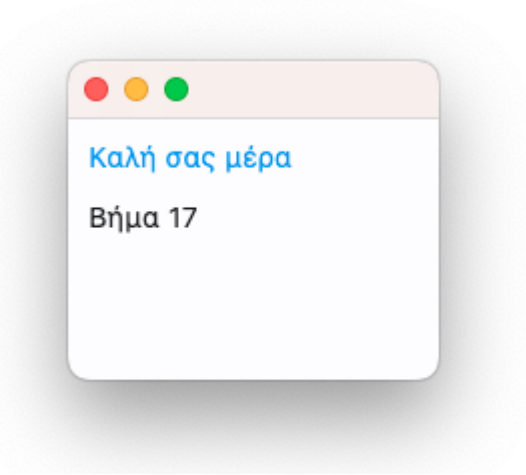
Για παράδειγμα αν επιθυμούμε να εντάξουμε το γραφικό αντικείμενο `ft.Text()` που εμφανίζει κείμενο, αντίστοιχα σε μια παράγραφο κειμένου `<p> ... </p>` της HTML ή σε ένα γραφικό αντικείμενο `tk.Label()` της tkinter, θα πρέπει να δώσουμε τις εξής εντολές.

```
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(1)

ft.app(target=main)
```

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



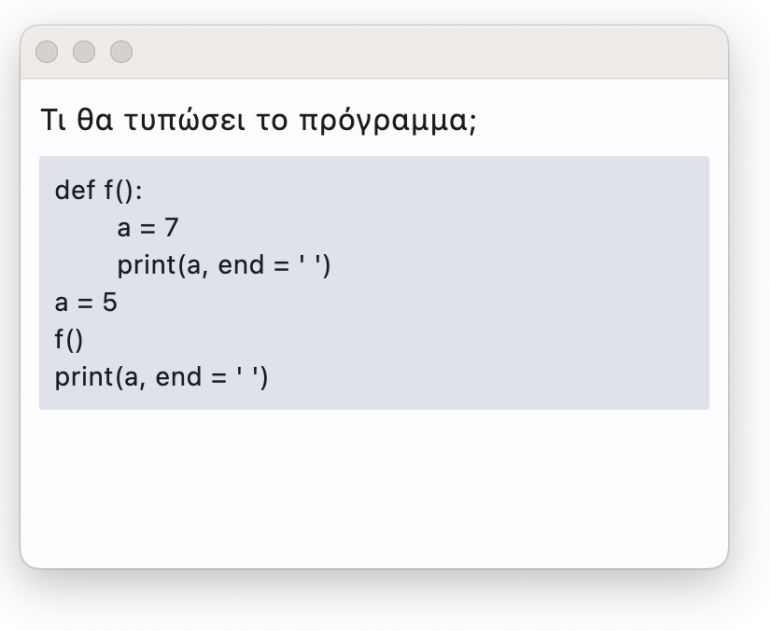
Άλλα γραφικά αντικείμενα που διατίθενται είναι τα Icon, Image, Markdown, ProgressBar, διάφορα είδη Buttons (π.χ. ElevatedButton, FilledButton, FloatingActionButton, IconButton, κλπ.), διάφοροι επιλογείς όπως Checkbox, Dropdown, Radio, Slider, Switch, το στοιχείο TextField για εισαγωγή κειμένου, κλπ. 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:



Υποδοχείς (containers)

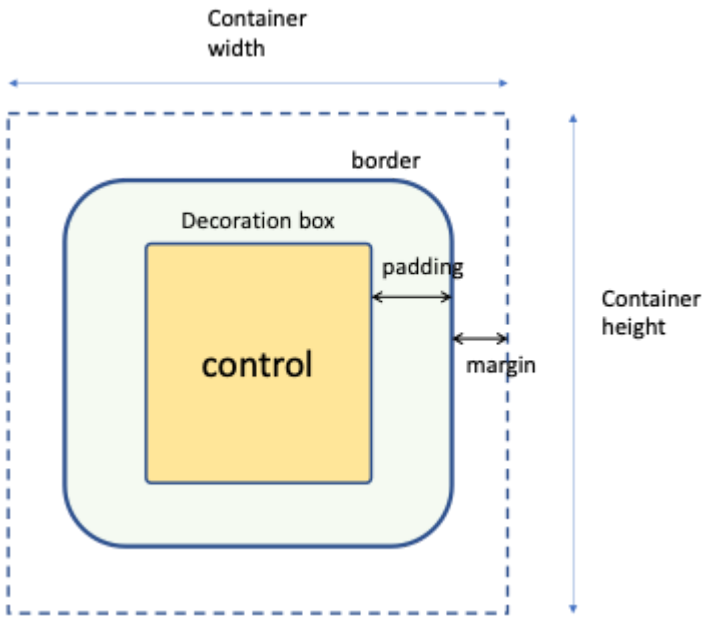
Η Flet διαδέτει γραφικά στοιχεία υποδοχείς που περιέχουν άλλα γραφικά στοιχεία.

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)
# εναλλακτικά:
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".

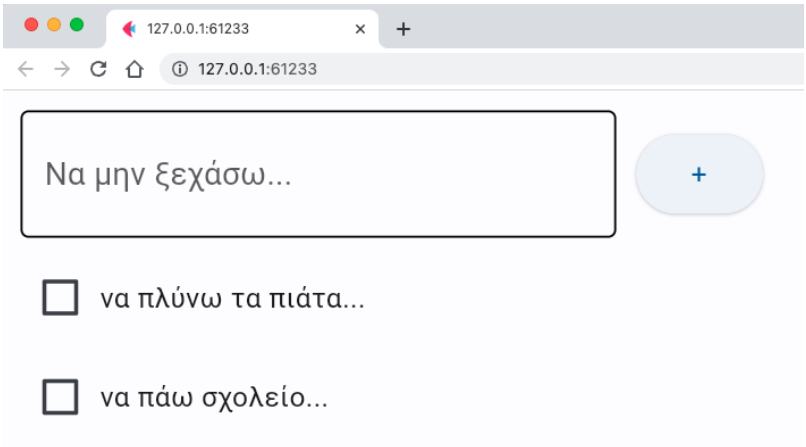
Ας δούμε τώρα ένα παράδειγμα

```
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 εισάγαγει 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 εισάγαγει 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.

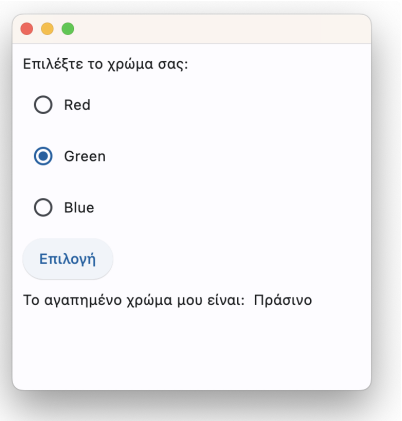
Μια φόρμα περιλαμβάνει συνήθως πεδία εισαγωγής κειμένου, πλήκτρα επιλογής μιας από αμοιβαία αποκλειόμενες επιλογές (radio buttons), στοιχεία checkbox που επιτρέπουν την επιλογή τους ή όχι, κλπ.

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"` .

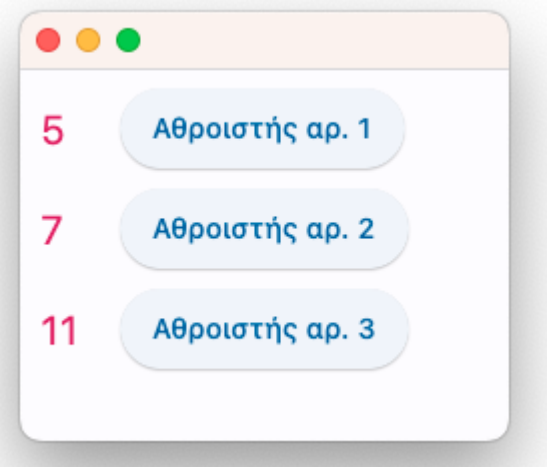
Θα πρέπει να σημειωθεί ότι η παλέτα έχει αποχρώσεις που παίρνουν τιμές από 100 μέχρι 900, με όσο μεγαλύτερη τιμή τόσο πιο σκούρα η χρωματική απόχρωση.

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'`

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, βοηθώντας 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"Αθροιστής αρ.
{self.id}", on_click=self.add_handler)])

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

ft.app(target=main)
```

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 retreiving 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
```

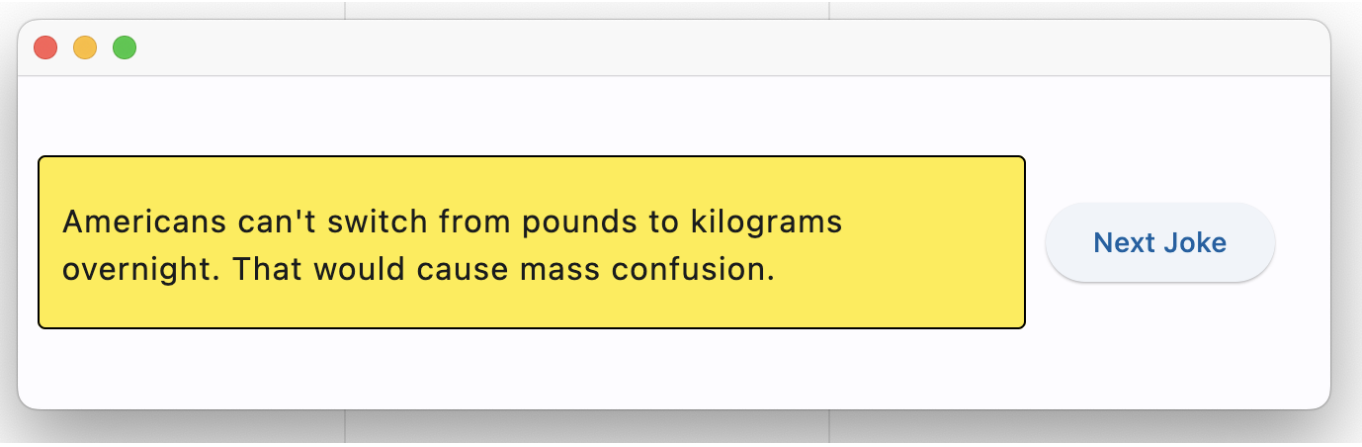
Στο παράδειγμα αυτό χρησιμοποιούμε τη μέθοδο `get` της requests για ανακτήσουμε πληροφορία από την ιστοσελίδα. Μάλιστα θέτουμε στην κεφαλίδα του αιτήματος την τιμή `'Accept': 'application/json` ώστε να μάς επιστρέψει τα δεδομένα σε μορφή json που είναι δομή παρόμοια με τα λεξικά της Python. Επίσης ορίζουμε 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`.

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://icanhazdadjoke.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 retreiving 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)
```

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 παοθήκευση 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

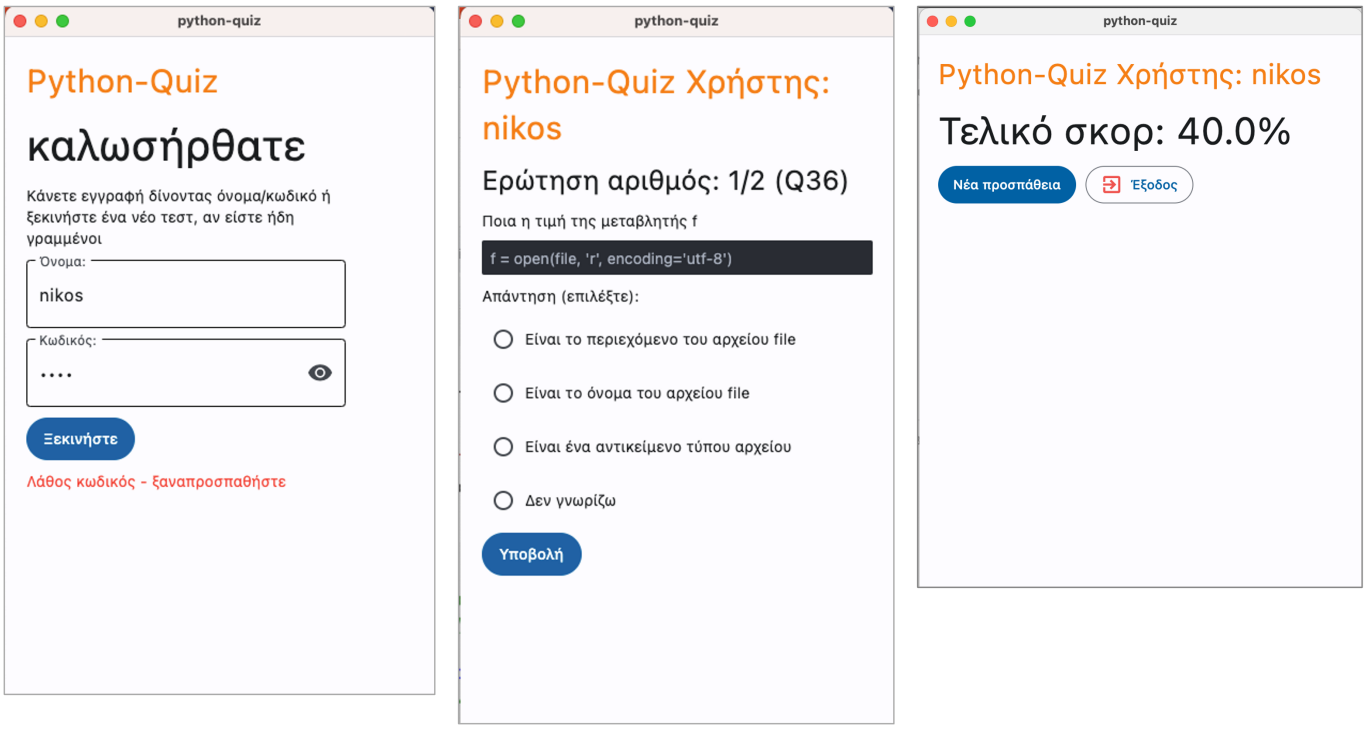
Μια καλή ιδέα σε μια εφαρμογή με αρκετό πλήθος σελίδων είναι να χρησιμοποιήσουμε για κάθε βασική οθόνη ένα γραφικό στοιχείο του χρήστη (userControl), το οποίο θα περιλαμβάνει όλα τα επί μέρους

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 έλεγχος 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 χρησιμοποείται η βιβλιοθήκη Flask.

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 ενσωνατώνει 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 διαφάλιση 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 επτύχει 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.

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(κριτίριο)`, 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 κάποις 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.

Κρυπτογράφηση μυστικών κωδικών

Το θέμα της ασφάλειας της πληροφορίας που διαχειριζόμαστε 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 υλοποιήσης του πρωτοκόλλου WSGI (Web Server Gateway Interface), του πρωτοκόλου 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:sha256:260000$8P8IcHXLsfUXgY3P$38d6abbd32863c80d92fd6db9647644742ea
eb3cf9a31d3efc096a07526dc785
```

- `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)
```

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 n ερωτήσεων
        the_quiz = [quiz.show_question(q) for q in questions]
        session["username"] = name
        print("session is...", session)
        return json.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 json.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. (α) 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.

Ανακεφαλαίωση: ο κύκλος ανάπτυξης μιας εφαρογής full-Python stack.

Ανακεφαλαιώνοντας, τα βήματα που ακολουθήσαμε για να αναπτύξουμε μια εφαρμογή με τεχνολογία full-Python stack ήταν τα εξής:

Βήμα 1. **Ανάπτυξη της εφαρμογής πελάτη** (font-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 απο 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 ελέγξουμε 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 εφαμρογής test-db.py και αφού εξασφλαλίσουμε 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.