



Graphical Interfaces

Intro to Flet library, MVC Pattern

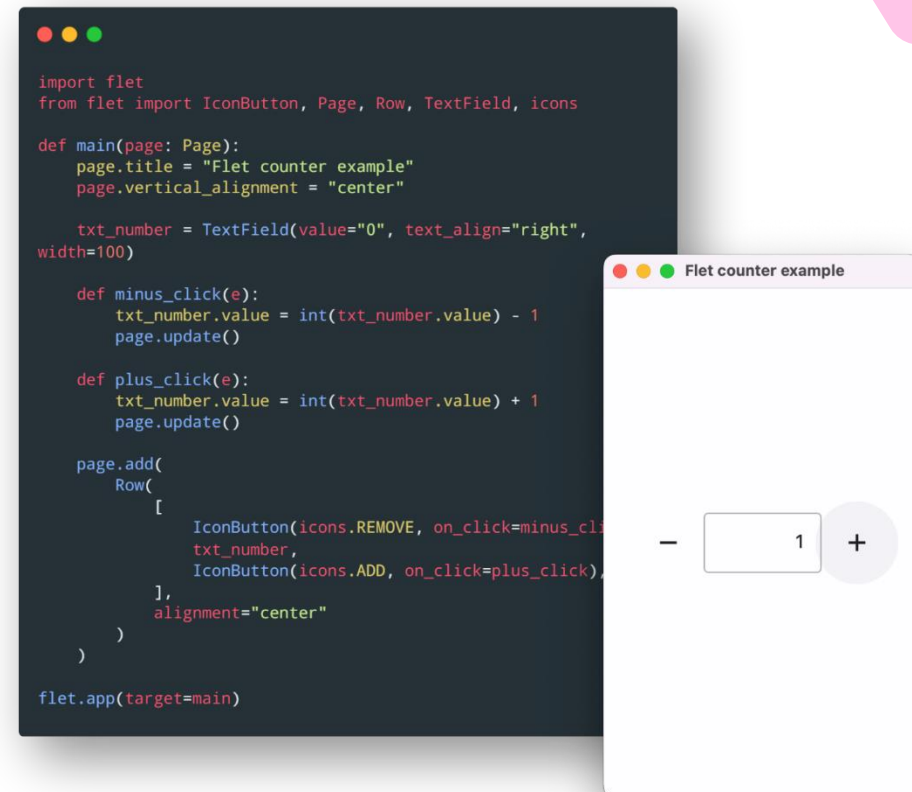
Giuseppe Averta

Carlo Masone

Francesca Pistilli

Davide Buoso

Gaetano Falco

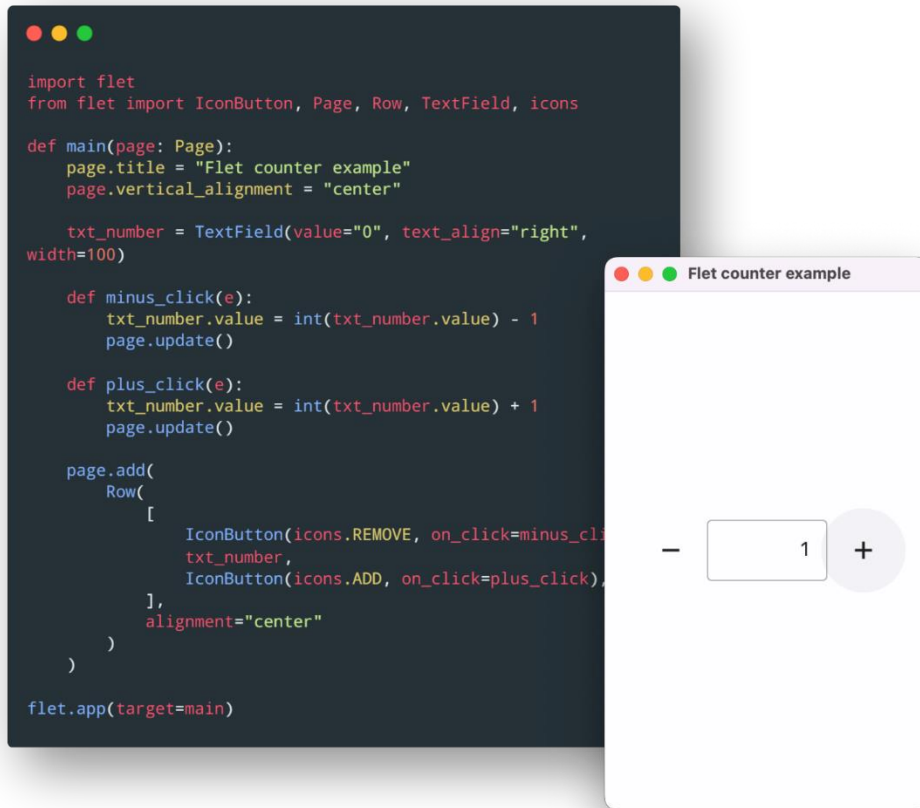




Graphical Interfaces

- Why GUIs
- Building GUIs with FLET
 - Intro to flet library
 - Controllers
- Pattern MVC
 - Motivation and logic of Model-view-controller
 - Code skeleton

Why GUIs



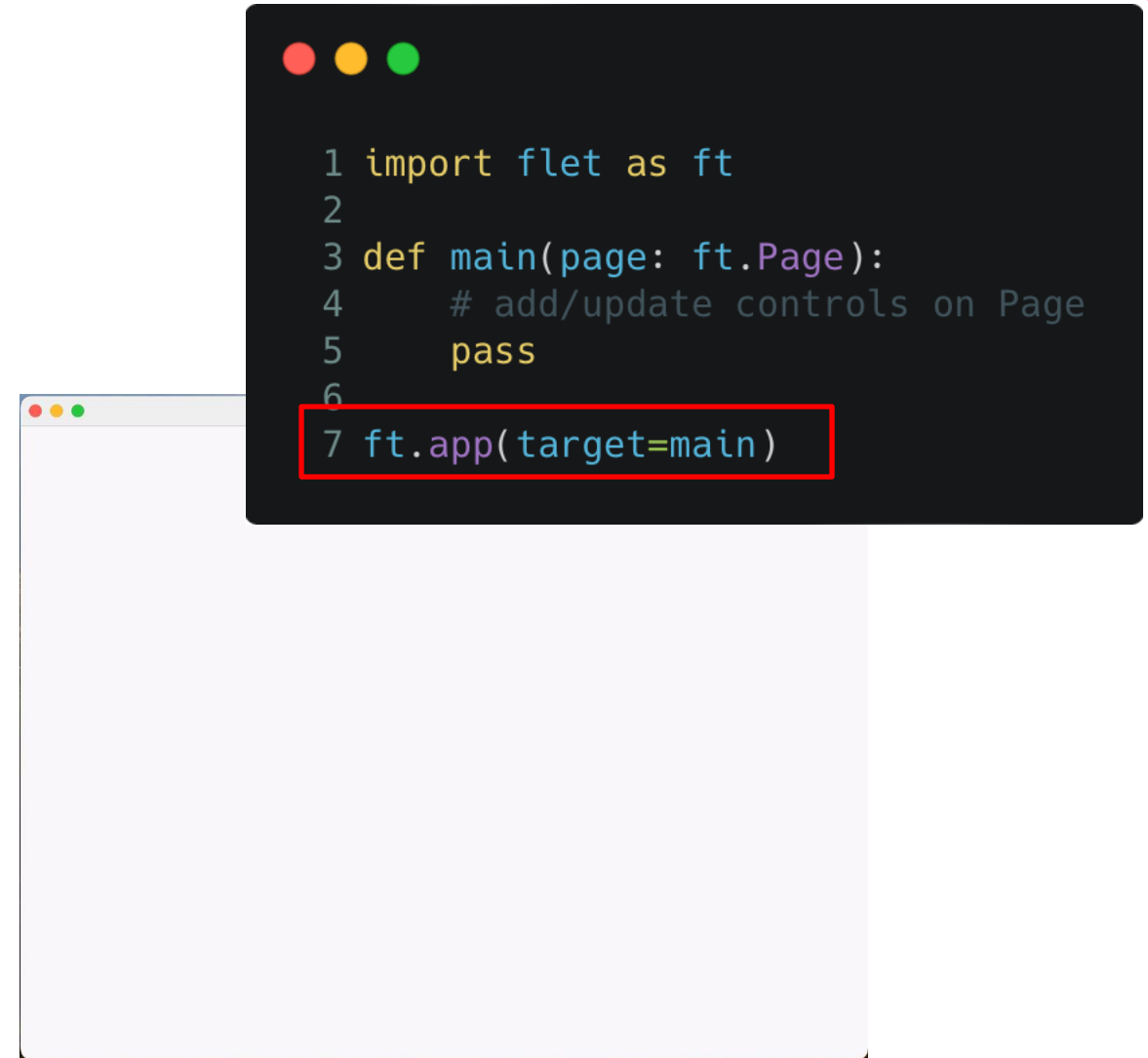
- Enhanced user-friendly interaction
- Productivity boost
- Accessibility and inclusivity in software
- Visual representation of data

GUIs in Python

- Few alternatives
 - Tk
 - Default graphic library
 - basic set of widgets for building graphical user interfaces.
 - Flutter
 - Proposed by Google as a multi-platform UI software devel kit
 - Flet
 - Library to build flutter apps in python
 - Easily build realtime web, mobile and desktop apps
 - Often prettier, with minimal frontend knowledge 😊
 - <https://flet.dev>

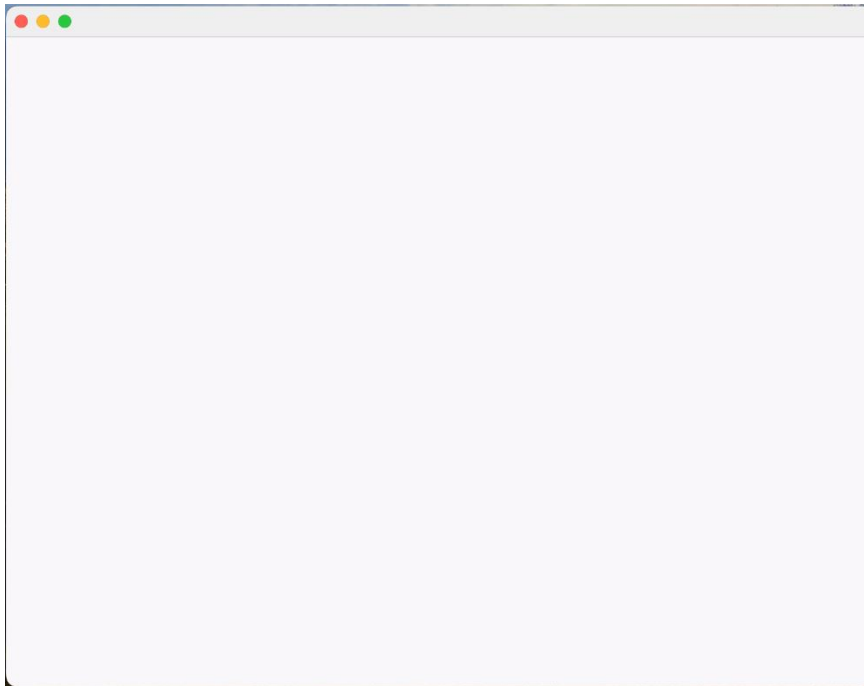
Let's start

- Our program will end with a call to `flet.app()`, where the app starts waiting for new user sessions
- Flet apps are organized in containers, where we put all the graphical elements



Let's start

- The top-most container is View (default option is FLET_APP)



```
1 import flet as ft
2
3 def main(page: ft.Page):
4     # add/update controls on Page
5     pass
6
7 ft.app(target=main, view=ft.AppView.FLET_APP)
```

Let's start

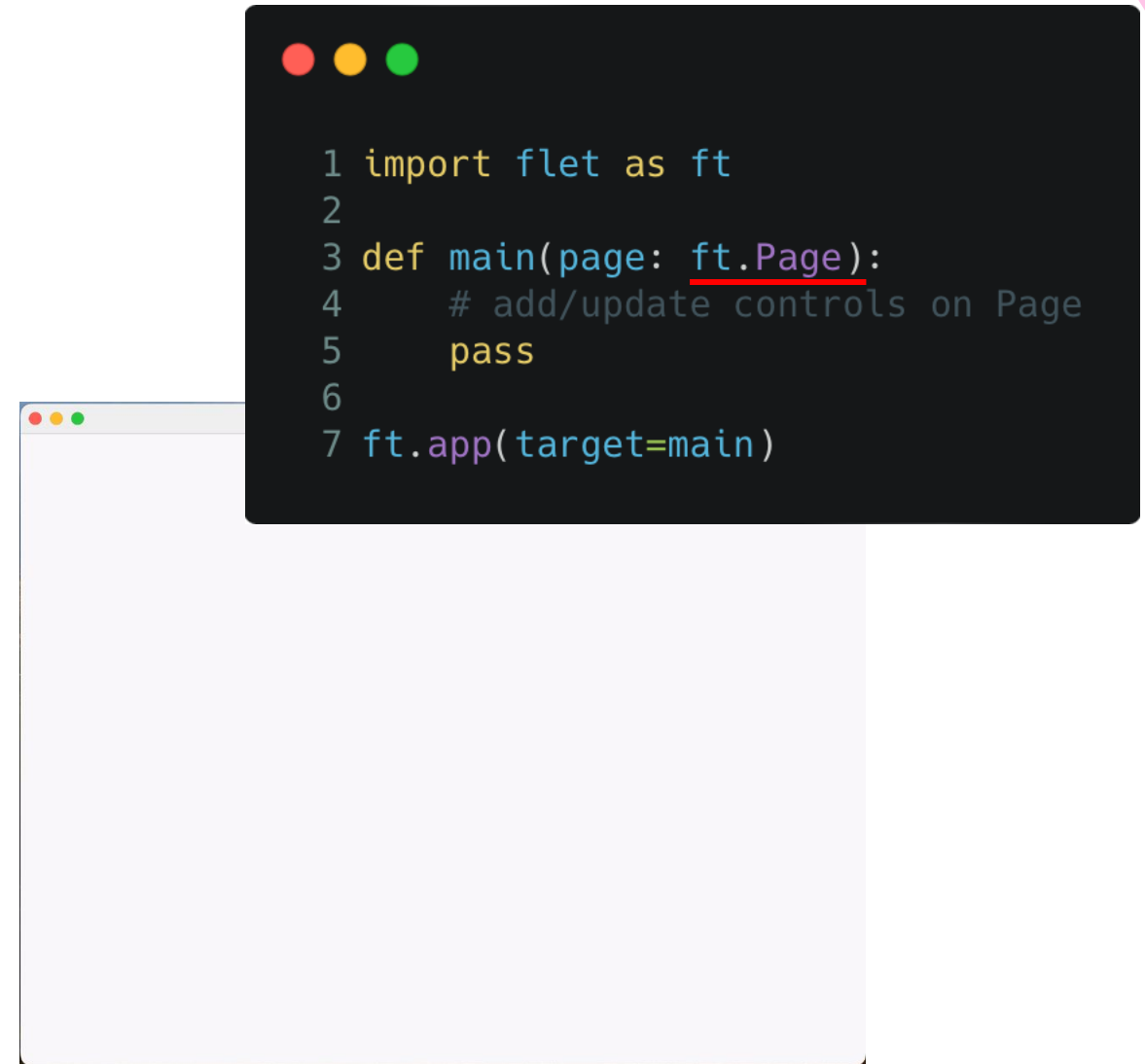
- But we can easily build a browser app

```
1 import flet as ft
2
3 def main(page: ft.Page):
4     # add/update controls on Page
5     pass
6
7 ft.app(target=main, view=ft.AppView.WEB_BROWSER)
```



Let's start

- Within View, we define a Page container, where we will place all the graphic elements.





GUI elements

- GUI elements are called Controls and are simply python classes which need to be instantiated in a Page (root Control)
- For example, to add some text we can use `ft.Text()`

GUI elements

- GUI elements are called Controls and are simply python classes which need to be instantiated in a Page (root Control)
- For example, to add some text we can use `ft.Text()`

```
1 import flet as ft
2
3 def main(page: ft.Page):
4     mytext = ft.Text(value="Ciao amici di TdP2024 :-)", color="Blue")
5
6
7 ft.app(target=main)
```

GUI elements

- GUI elements are called Controls and are simply python classes which need to be instantiated in a Page (root Control)
- For example, to add some text we can use `ft.Text()`

```
1 import flet as ft
2
3 def main(page: ft.Page):
4     mytext = ft.Text(value="Ciao amici di TdP2024 :-)", color="Blue")
5     page.controls.append(mytext)
6     page.update()
7
8
9 ft.app(target=main)
```

GUI elements

- GUI elements are called Controls and are simply python classes which need to be instantiated in a Page (root Control)
- Shortcut `page.add()` to directly add the control and update the page.

```
1 import flet as ft
2
3 def main(page: ft.Page):
4     mytext = ft.Text(value="Ciao amici di TdP2024 :-)", color="Blue")
5     page.controls.append(mytext)
6     page.update()
7
8     page.add(ft.Text(value="Quest'anno in Python"))
9
10
11 ft.app(target=main)
```

GUI elements

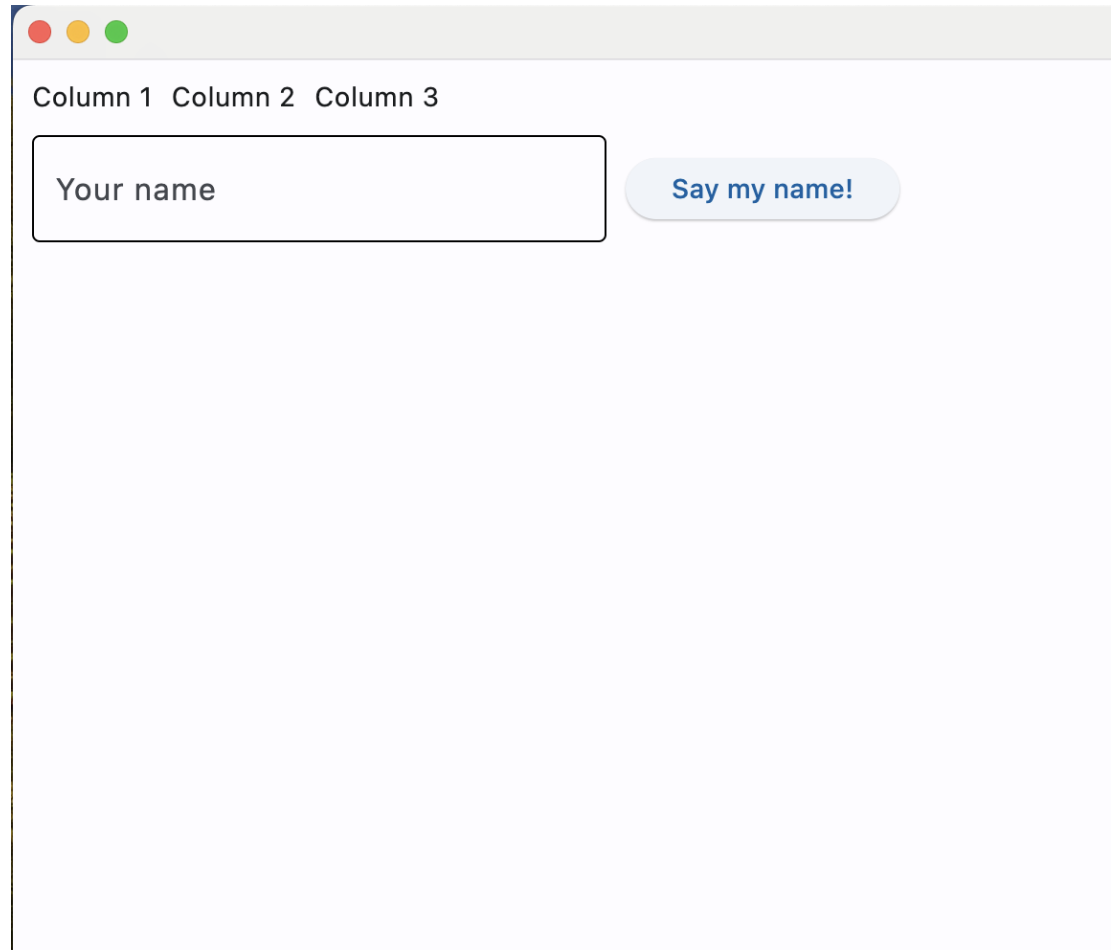
- Elements in Page are updated only when `page.update()` is called
- Note that the content of Controls can be updated anytime, and not only when added to the page, by simply changing the value field

```
1 import flet as ft
2 import time
3
4 def main(page: ft.Page):
5     t = ft.Text()
6     page.add(t) # it's a shortcut for page.controls.append(t) and then page.update()
7
8     for i in range(10):
9         t.value = f"Step {i}"
10        page.update()
11        time.sleep(1)
12
13 ft.app(target=main)
```

Step 8

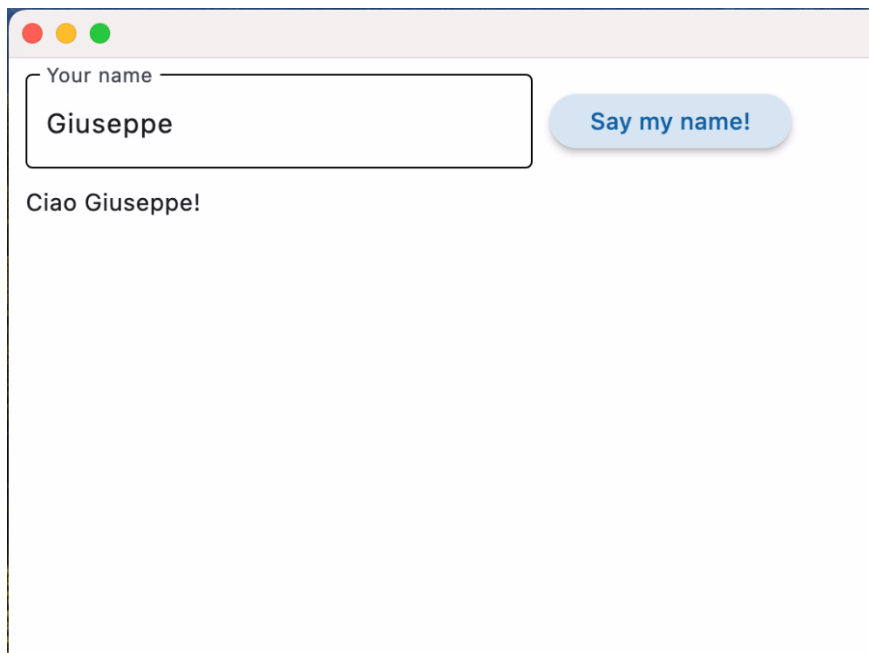
Giving some structure

```
1 import flet as ft
2
3 def main(page: ft.Page):
4     page.add(
5         ft.Row(controls=[
6             ft.Text("Column 1"),
7             ft.Text("Column 2"),
8             ft.Text("Column 3")
9         ])
10    )
11
12    page.add(
13        ft.Row(controls=[
14            ft.TextField(label="Your name"),
15            ft.ElevatedButton(text="Say my name!")
16        ])
17    )
18
19 ft.app(target=main)
```



Event handlers

- Some control items can trigger events:
 - e.g. buttons when clicked



```
1 import flet as ft
2
3 def main(page: ft.Page):
4
5     def handle_button(e):
6         page.add(ft.Text("Ciao " + name.value + "!"))
7
8     name = ft.TextField(label="Your name")
9     page.add(
10         ft.Row(controls=[
11             name,
12             ft.ElevatedButton(text="Say my name!", on_click=handle_button)
13         ])
14     )
15
16 ft.app(target=main)
```

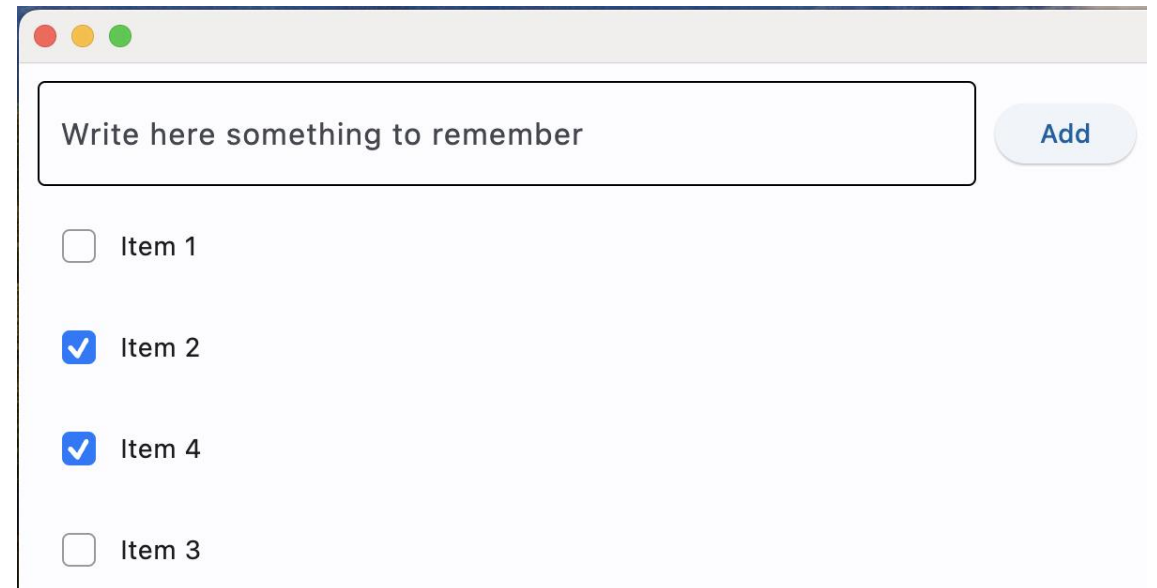


Many options

- <https://flet.dev/docs/controls>

Example: ToDo list

- Let's try to get something like this

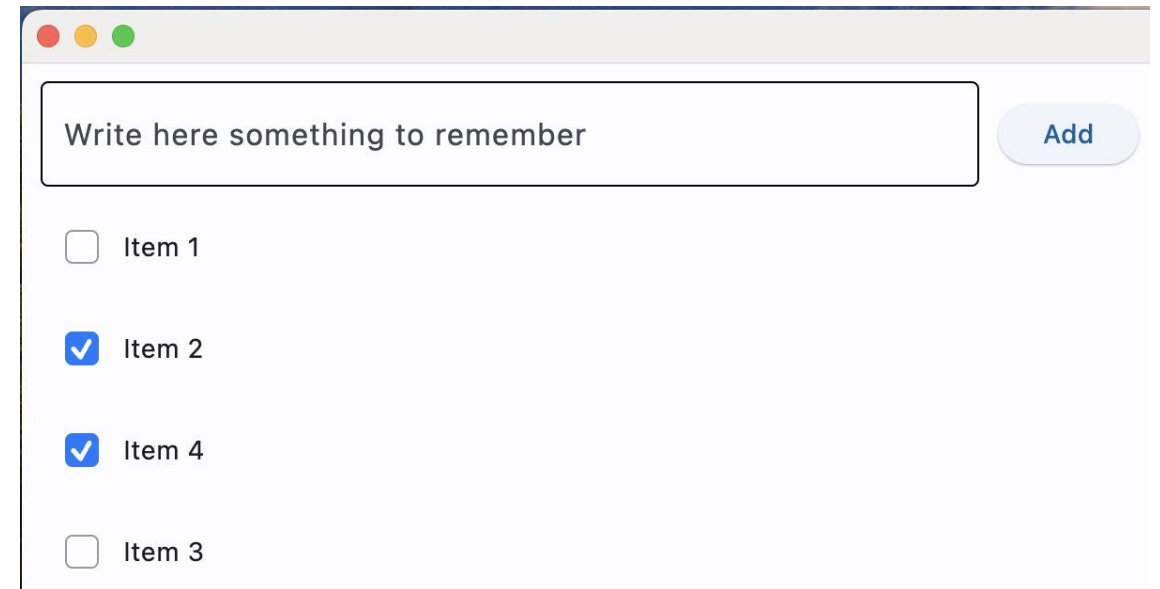


Write here something to remember Add

- ☐ Item 1
- ☒ Item 2
- ☒ Item 4
- ☐ Item 3

Example: ToDo list

```
1 import flet as ft
2 def main(page):
3
4     def handleAdd(e):
5         page.add(ft.CupertinoCheckbox
6                 (label=txtBox.value, value=False))
7         txtBox.value = ""
8         txtBox.update()
9
10    txtBox = ft.TextField(label="Write here something to remember", width=500)
11    button = ft.ElevatedButton(text="Add", on_click=handleAdd)
12    page.add(ft.Row([txtBox, button]))
13
14 ft.app(target=main)
```

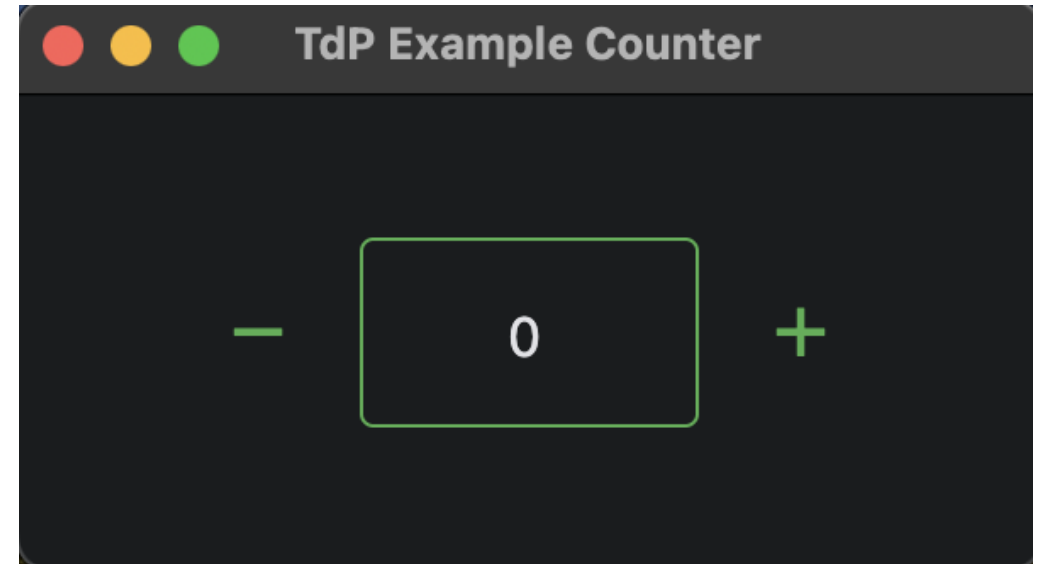


The screenshot shows a web application interface for a todo list. At the top, there is a text input field with the placeholder text "Write here something to remember" and an "Add" button to its right. Below the input field, there is a list of four items, each with a checkbox and a label:

- ☐ Item 1
- ☒ Item 2
- ☒ Item 4
- ☐ Item 3

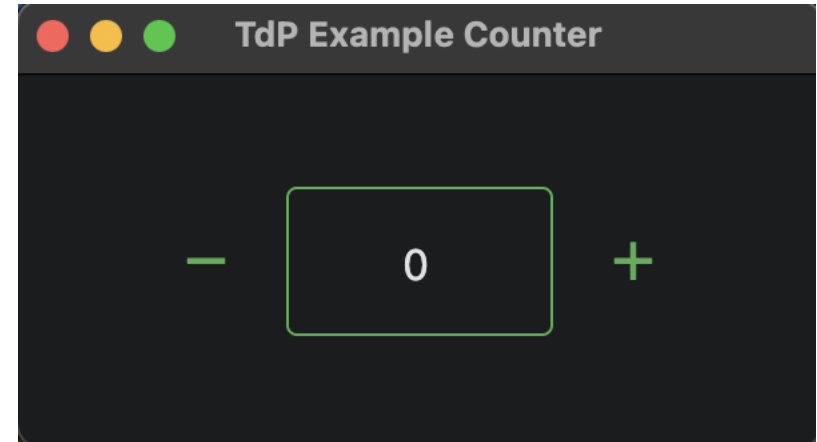
Example: Counter

- Let's build a counter



Example: Counter

```
1 import flet as ft
2
3 def main(page: ft.Page):
4     page.title = "TdP Example Counter"
5     page.vertical_alignment=ft.MainAxisAlignment.CENTER
6
7     def handleRemove(e):
8         val.value = val.value-1
9         val.update()
10    def handleAdd(e):
11        val.value = val.value+1
12        val.update()
13
14    val = ft.TextField(width=100,border_color="green",value=0,text_align="center")
15    page.add(ft.Row([
16        ft.IconButton(icon=ft.icons.REMOVE,icon_color="green",on_click=handleRemove),
17        val,
18        ft.IconButton(icon=ft.icons.ADD,icon_color="green",on_click=handleAdd)
19    ],
20    alignment=ft.MainAxisAlignment.CENTER
21    ))
22    page.update()
23
24 ft.app(target=main)
```



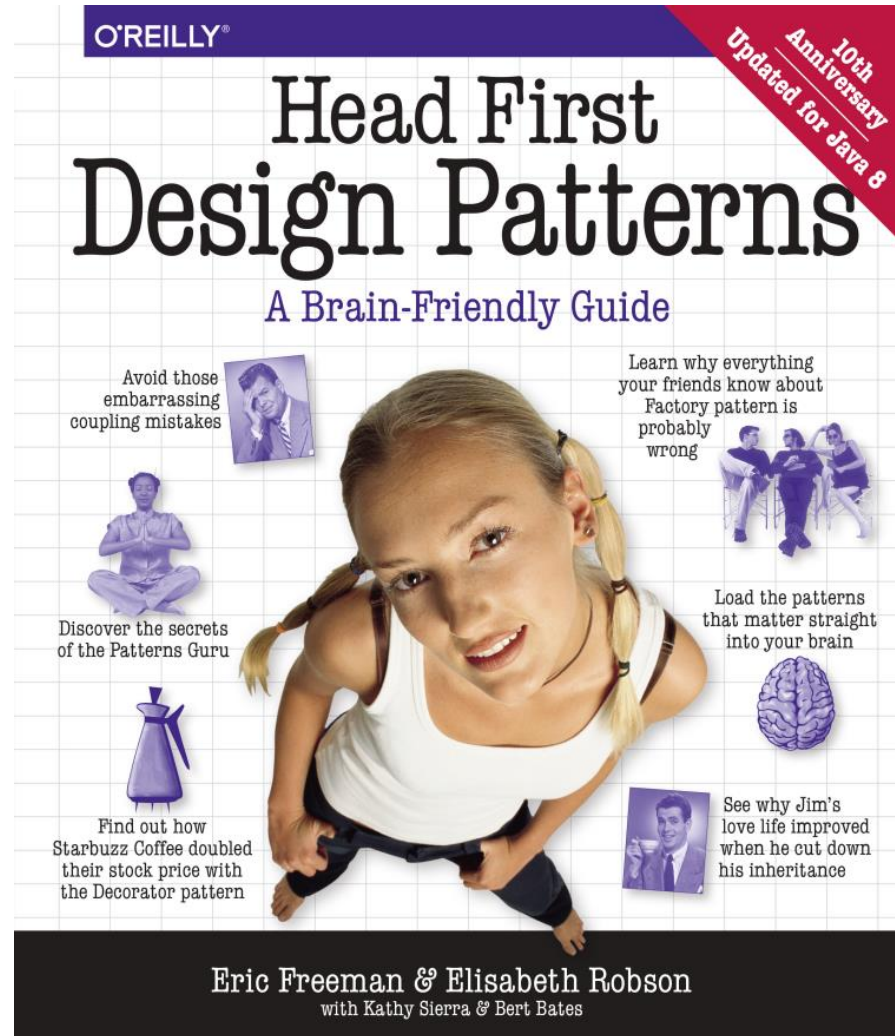


PATTERN MVC

Application complexity and MVC

- Interactive, graphical applications exhibit complex interaction patterns
- Flow of control is in the hand of the user
- Actions are mainly asynchronous
- How to organize the program?
- Where to store data?
- How to decouple application logic from interface details?
- How to keep in sync the inner data with the visible interface?

Design Patterns



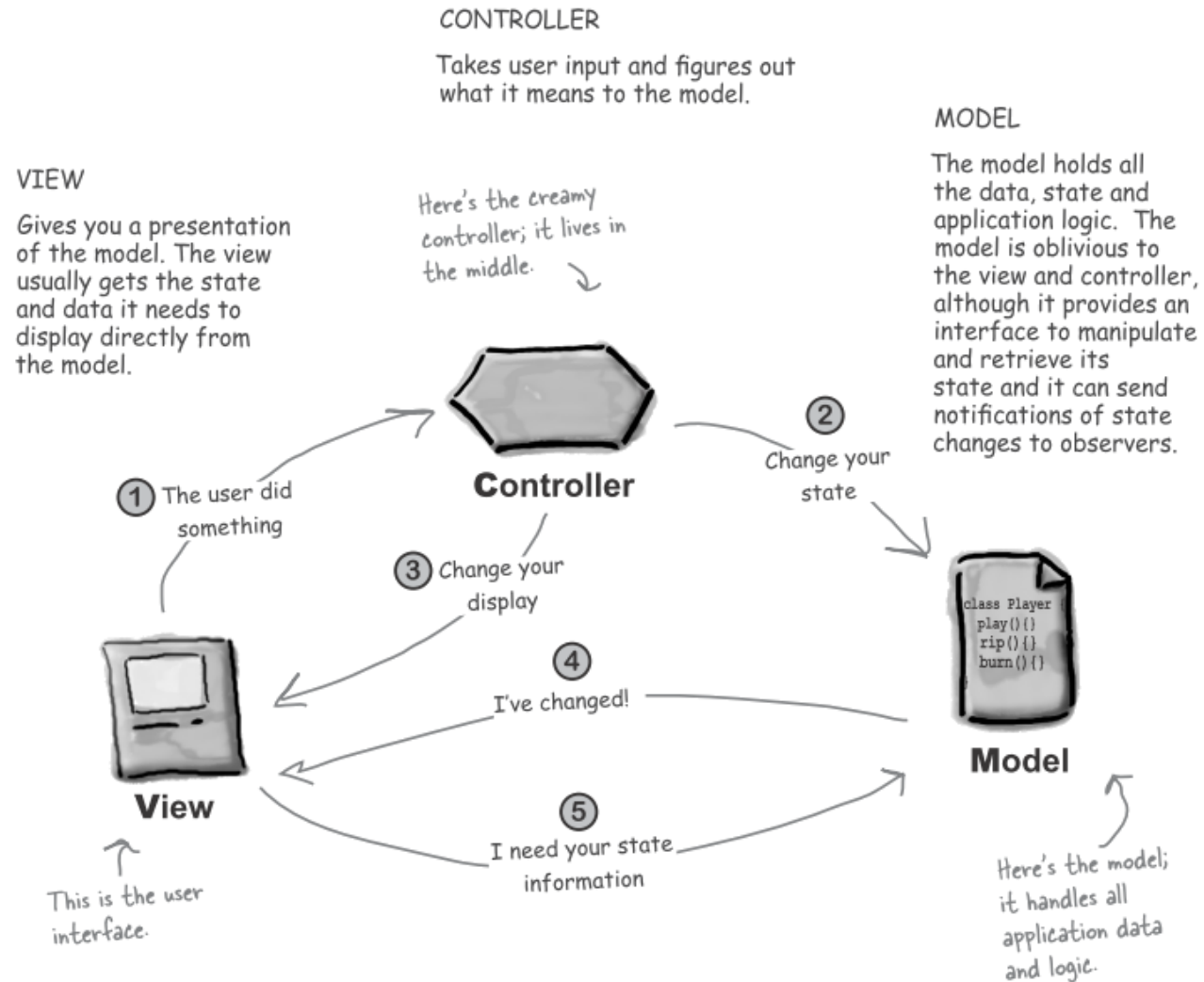


Design Patterns



- How to build systems with good OO design qualities
 - Reusable, extensible, maintainable
- Patterns: Proven solutions to recurrent problems
 - Design problems
 - Programming problems
- Adopt and combine the OO constructs
 - Interface, inheritance, abstract classes, information hiding, polymorphism, objects, statics, ...
- Help dealing with changes in software
 - Some part of a system is free to vary, independently from the rest

MVC pattern defined



Normal life-cycle of interaction

- ① **You're the user — you interact with the view.**
The view is your window to the model. When you do something to the view (like click the Play button) then the view tells the controller what you did. It's the controller's job to handle that.
- ② **The controller asks the model to change its state.**
The controller takes your actions and interprets them. If you click on a button, it's the controller's job to figure out what that means and how the model should be manipulated based on that action.
- ③ **The controller may also ask the view to change.**
When the controller receives an action from the view, it may need to tell the view to change as a result. For example, the controller could enable or disable certain buttons or menu items in the interface.
- ④ **The model notifies the view when its state has changed.**
When something changes in the model, based either on some action you took (like clicking a button) or some other internal change (like the next song in the playlist has started), the model notifies the view that its state has changed.
- ⑤ **The view asks the model for state.**
The view gets the state it displays directly from the model. For instance, when the model notifies the view that a new song has started playing, the view requests the song name from the model and displays it. The view might also ask the model for state as the result of the controller requesting some change in the view.

Mapping concepts to Python

- View: presenting the UI
 - View class, which instantiates controllers defined by FLET
 - interacts with the controller
- Controller: reacting to user actions
 - Set of event handlers
 - Local variable to handle the interface status
- Model: handling the data
 - Class(es) including data
 - Persistent data in Data Bases

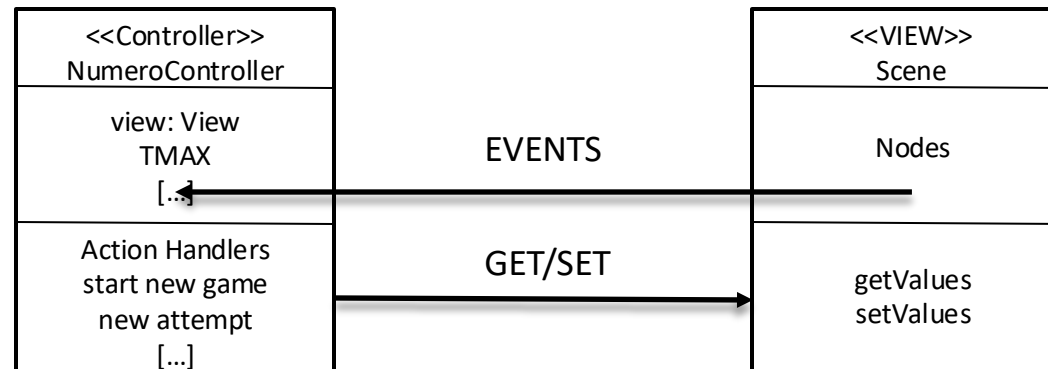


Exercise

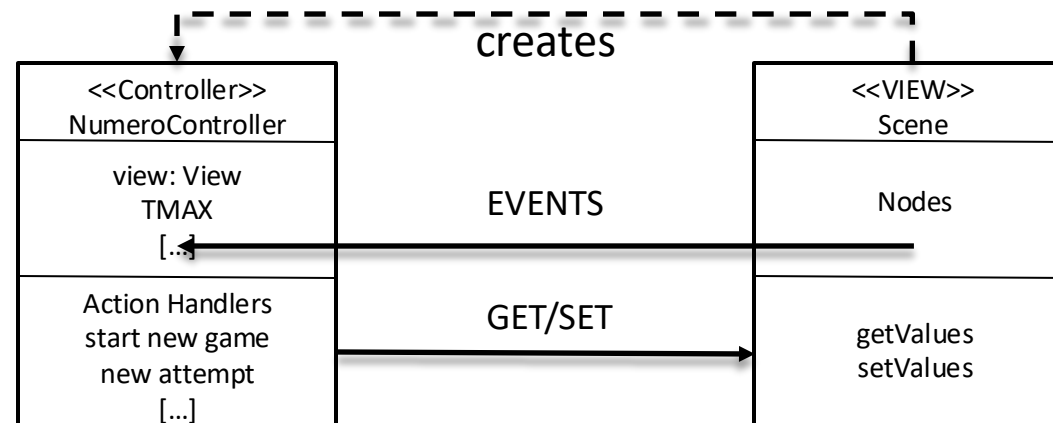


- Update the «IndovinaNumero» by using the MVC pattern
- Where do you declare the data class?
- Which class should have access to which?
- Who creates what objects?

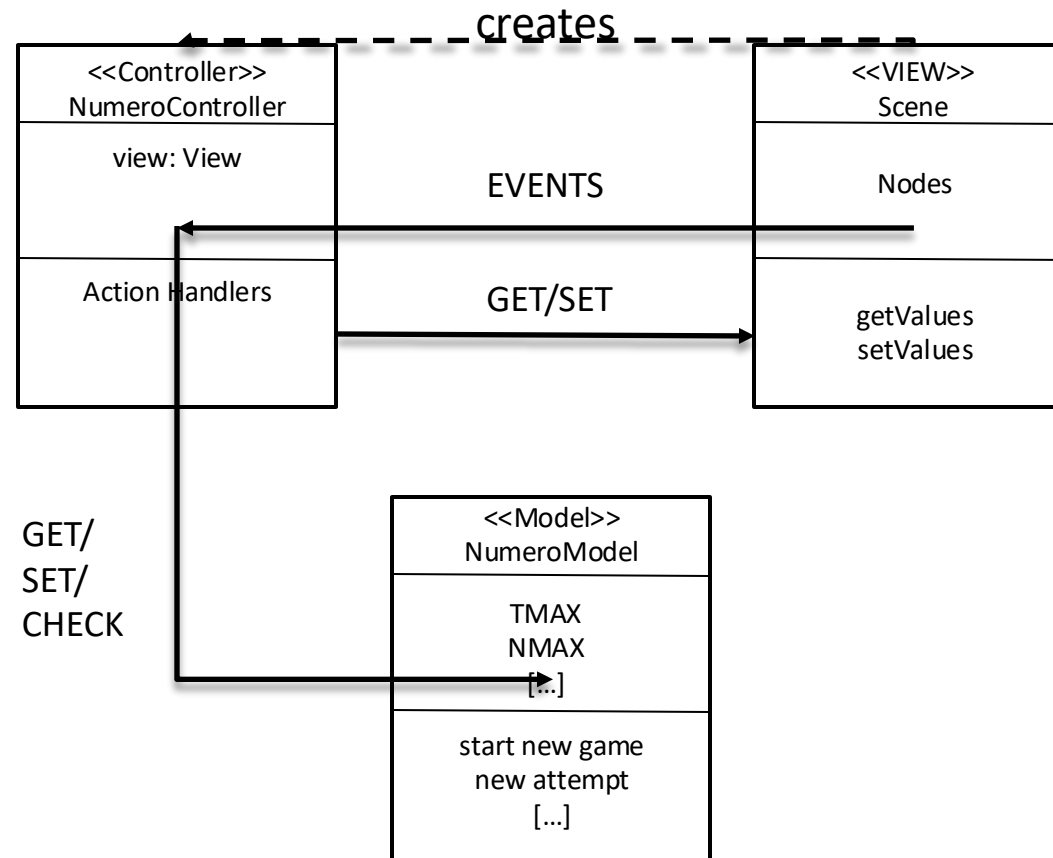
A possible solution



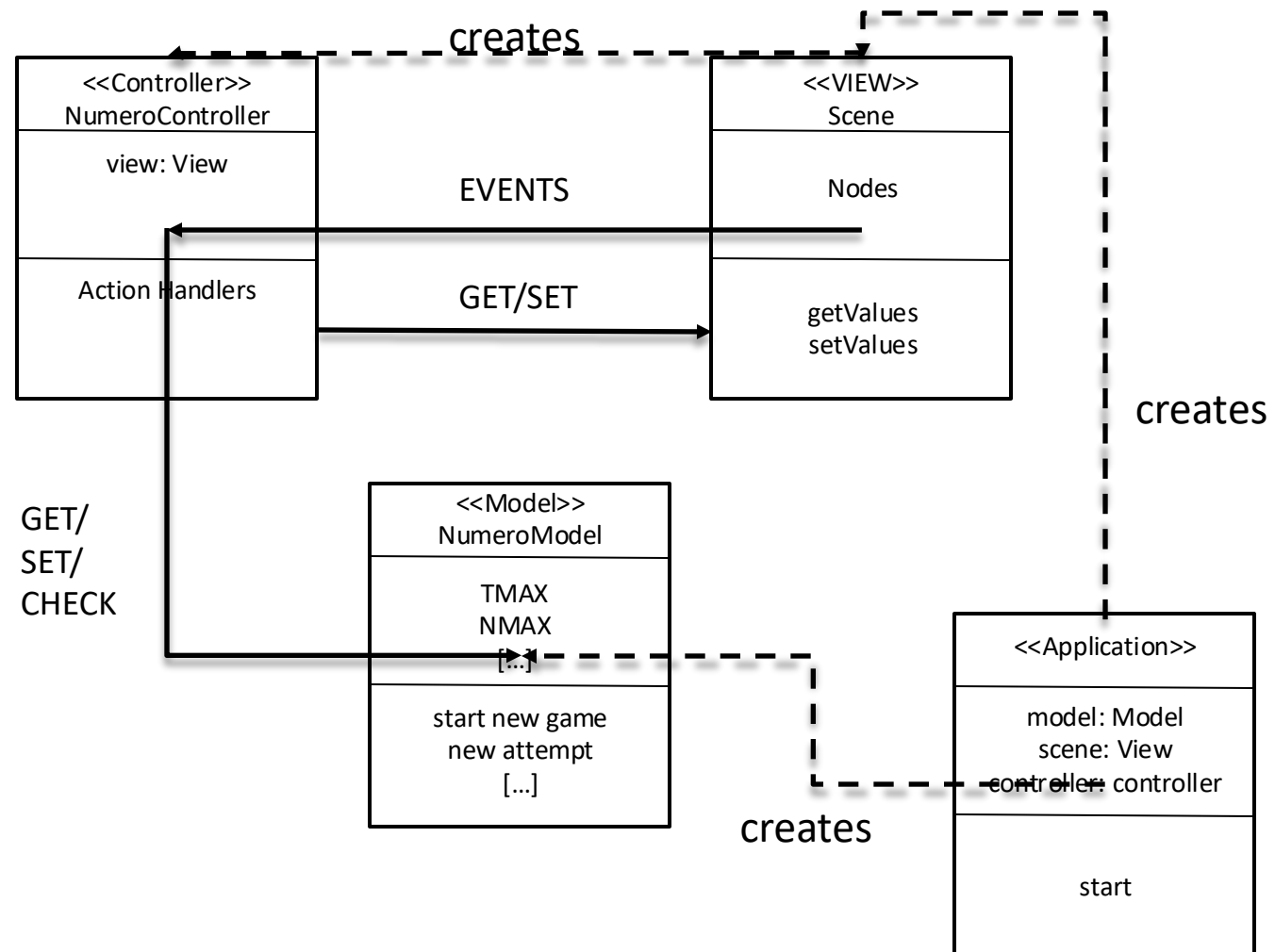
A possible solution



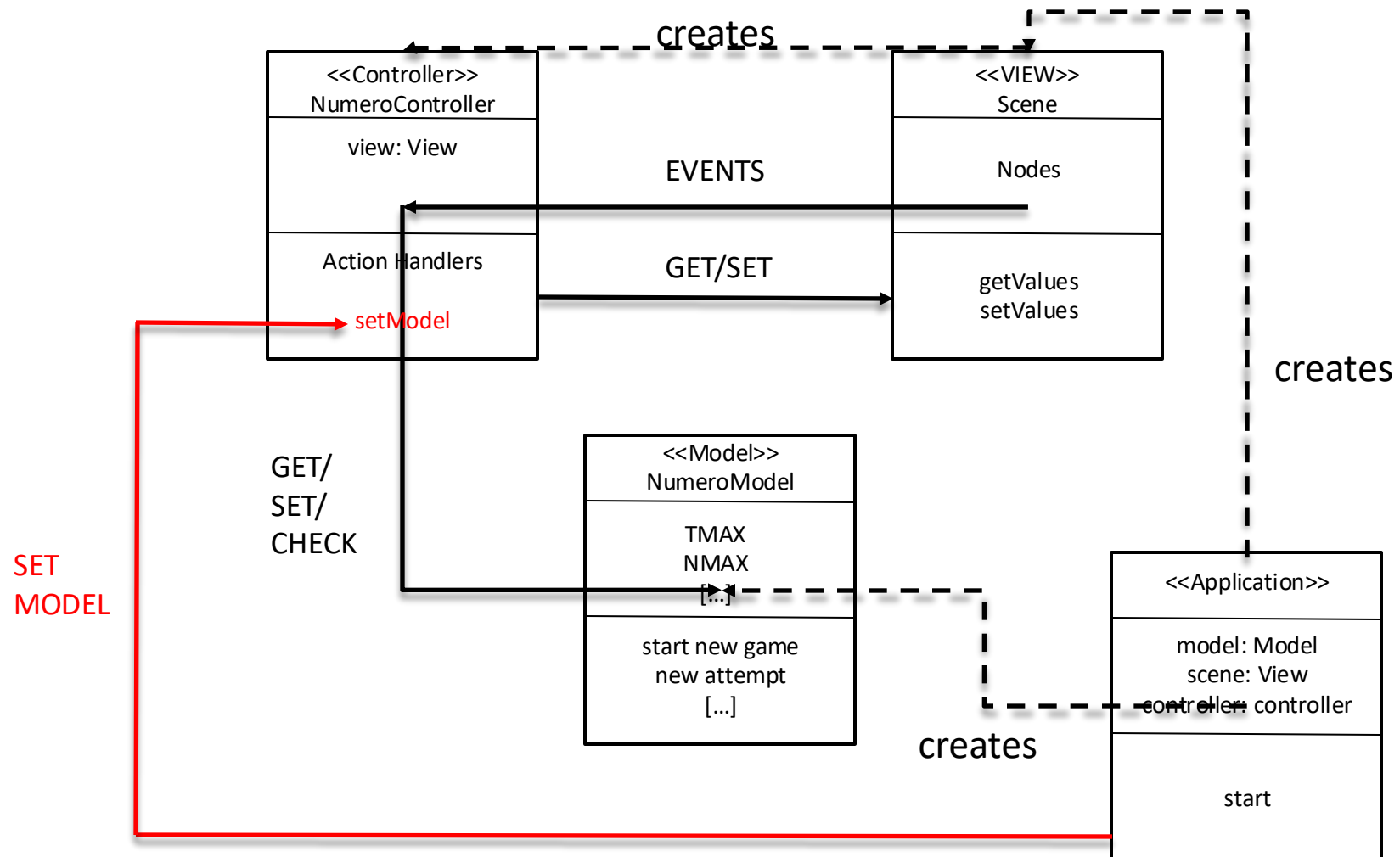
A possible solution



A possible solution



A possible solution



Example: Libretto

TdP with FLET (2024-2025)

☐ Light theme

Il mio Libretto Voti

Lista degli esami finora sostenuti:

- a: 24
- b: 21
- c: 26
- d: 28
- e: 27
- f: 23
- g: 26
- h: 19
- i: 21
- j: 28

Example: Indovina il Numero

Indovina il Numero - edizione TdP 2024

Numero Max
100

Tentativi Rimanenti
2

Tentativi Max
6

Valore

Nuova Partita

Indovina

Indovina a quale numero sto pensando:

Nope! il numero segreto è più grande di 50.

Nope! il numero segreto è più grande di 75.

Nope! il numero segreto è più piccolo di 88.

Grande! il numero segreto era 80.

License



- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
 - **Share** — copy and redistribute the material in any medium or format
 - **Adapt** — remix, transform, and build upon the material
 - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
 - **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
 - **NonCommercial** — You may not use the material for commercial purposes.
 - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
 - **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>

