Shiny for Python: Reactive Web Apps Made Simple

Pycon Finland 2025

Mohamed El Fodil Ihaddaden, 2025/10/17

A few words about me

- R/Python developer at HDI Global SE
- Algerian based in Germany
- Open source R developer

Shiny?

- Reactive web framework
- Originated in the R ecosystem in 2012
- No web development knowledge needed

Shiny Core and Shiny express

- **Shiny Core:** full control over UI, server logic, and layout using Shiny's reactive model.

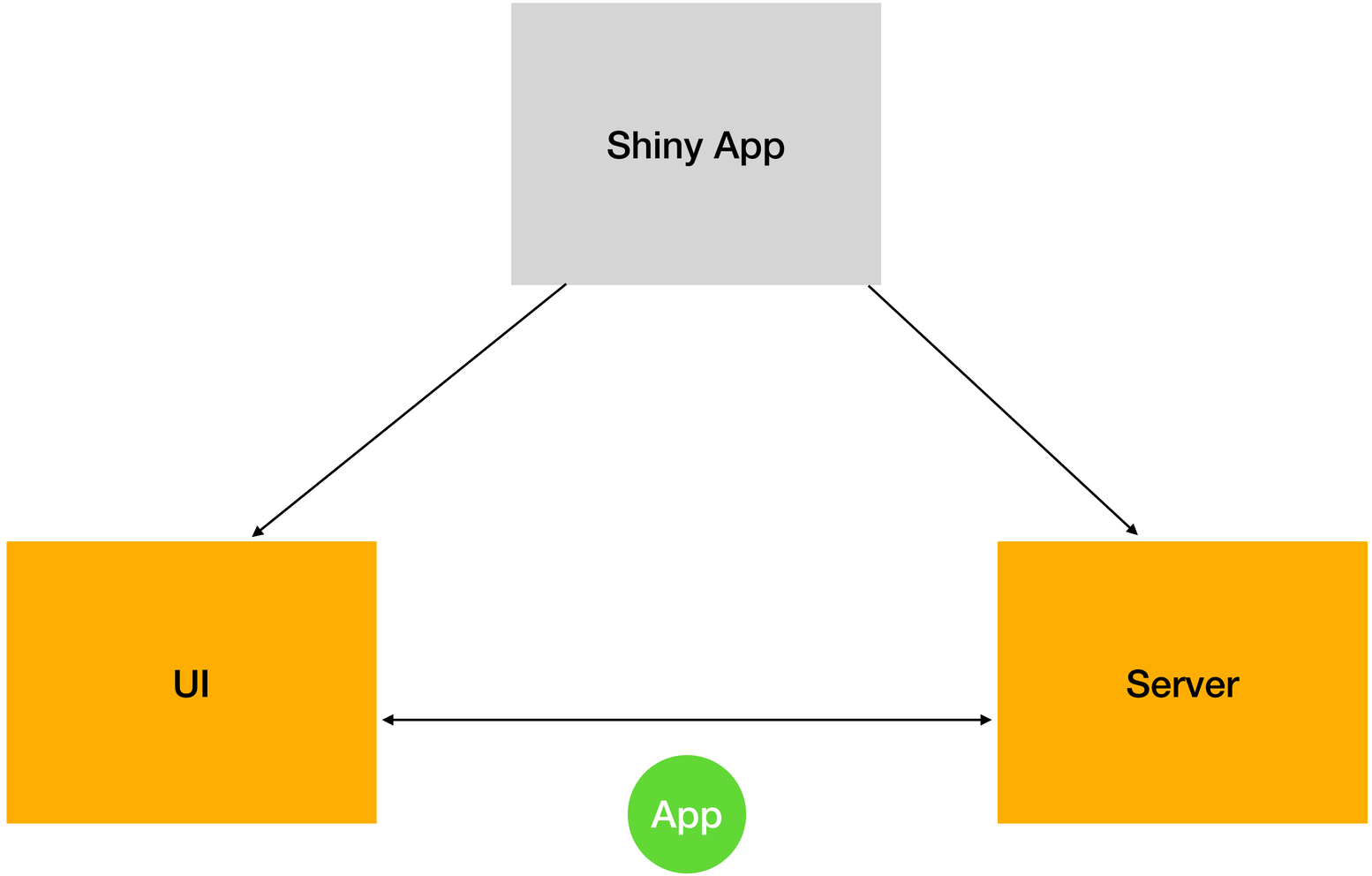
- **Shiny Express:** less code needed, "notebook-friendly" syntax

Shiny Core and Shiny express

- **Shiny Core:** Building your perfect own sauna.

- **Shiny Express:** Using a public ready to use sauna.

App structure



App structure

```python
from shiny import App, ui

app_ui = ui.page_fluid()


def server(input, output, session) -> None:
    return


app = App(app_ui, server)
```

Static components

- HTML elements that are rendered at run time and won’t change during the app’s session.

Demo: base_app.py

```python
app_ui = ui.page_fluid(
    ui.br(),
    ui.h2("Coffee Consumption in Finland"),
    ui.br(),
    ui.p("Finland drinks more coffee per person than any other country in the world."),
    ui.p("On average, a Finn drinks about 4 cups of coffee per day."),
    ui.br(),
    ui.img(
        src="https://upload.wikimedia.org/wikipedia/commons/4/45/A_small_cup_of_coffee.JPG",
        alt="Cup of coffee",
        width="300",
        height="300",
    ),
    ui.br(),
    ui.p("Data source: International Coffee Organization"),
)
```

Dynamic components

- Dynamic elements that are rendered **or not** at run time, and which **can** change during the app's session.
- Mainly texts, tables and graphs

Demo:

```python
app_ui = ui.page_fluid(
    ...some code before ...
    ui.output_data_frame("coffee_tbl"),
    ..some code after ...
)

def server(input, output, session) -> None:
    @output
    @render.data_frame
    def coffee_tbl():
        coffee_data = get_coffee_data()
        return coffee_data.head(10)


app = App(app_ui, server)
```

Pattern

```
# Within the UI part
ui.output_*(id="some_unique_id")

# Within the server part
@output
@render.*
def some_unique_id():
    return something

# *the component that you want to render
```

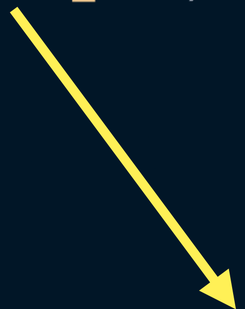
IO Reactivity

Shiny inputs

- Text
- Numeric
- Date
- Switch
- Radio
- File Input

IO Reactivity Pattern

```python
# Within the UI part
ui.input_slider("some_input_id")

# Within the server part
@output
@render.data_frame
def coffee_tbl():
    number_of_rows = input.some_input_id()
    return coffee_data.head(number_of_rows)
```



IO = Input Output

SE Reactivity

Shiny inputs

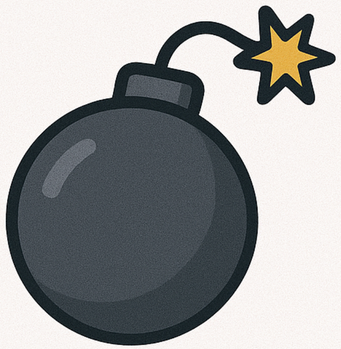
- Action Buttons

SE Reactivity Pattern

```python
# In the UI part
ui.input_action_button("action_button_id", "Click me")

# In the server part
@reactive.effect
@reactive.event(input.action_button_id)
def _():
    ... do anything you want ...
```

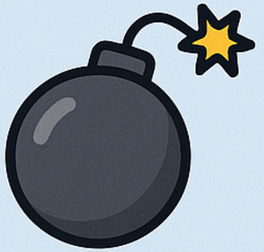
Controlled Reactivity

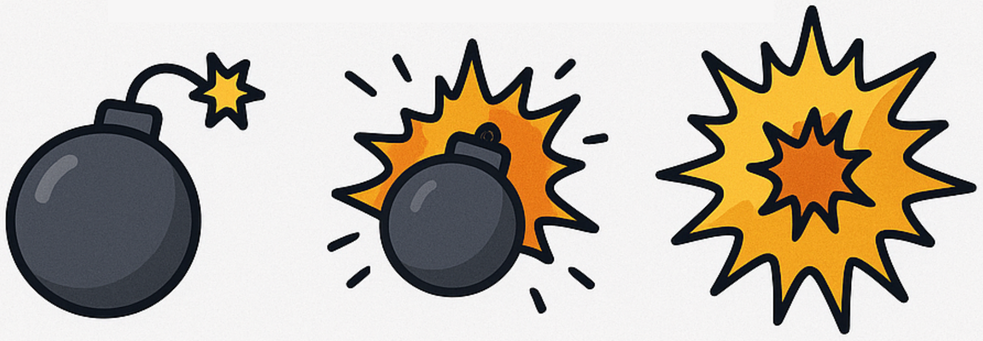
- Reactive Values

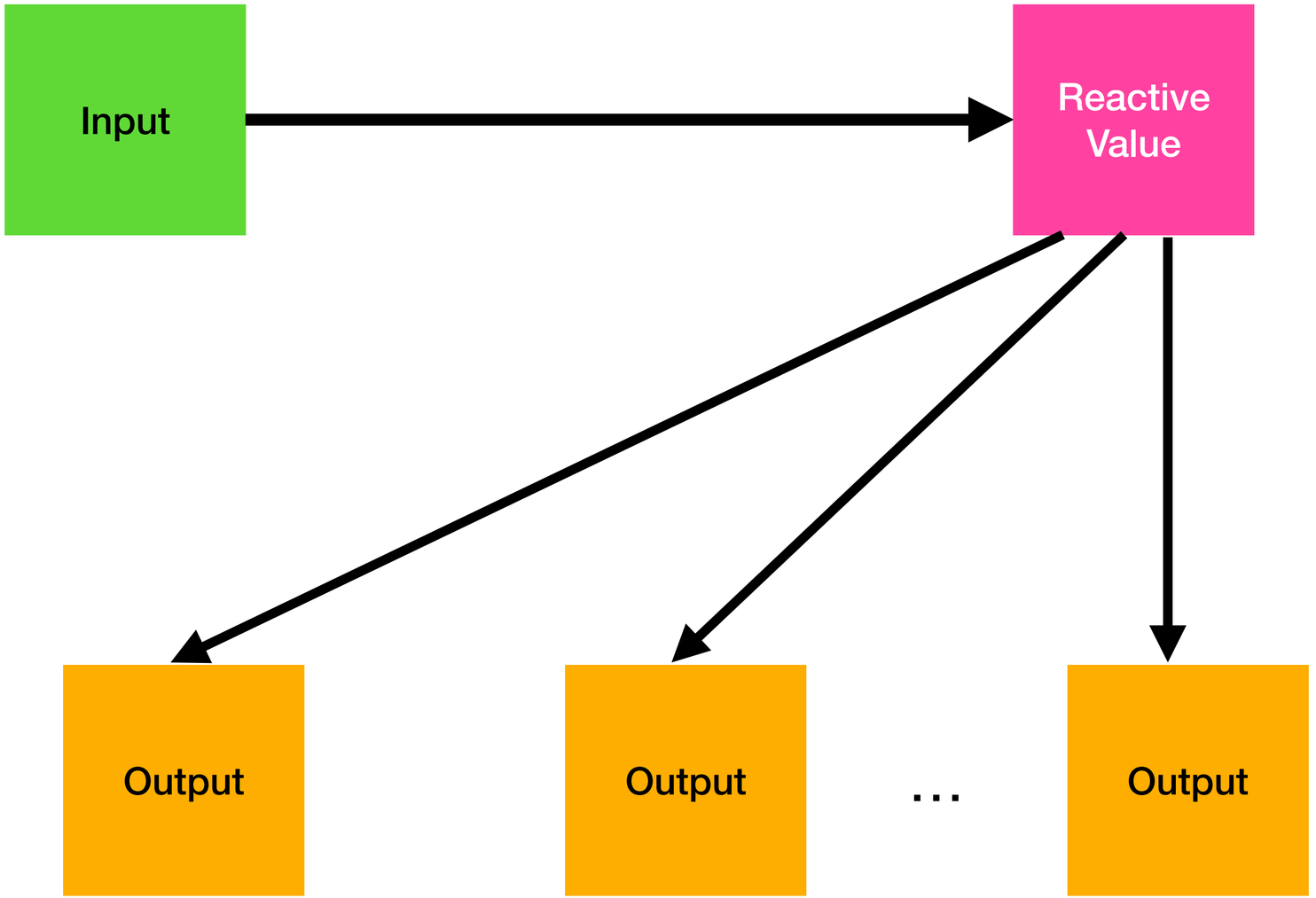


initial

reactive component



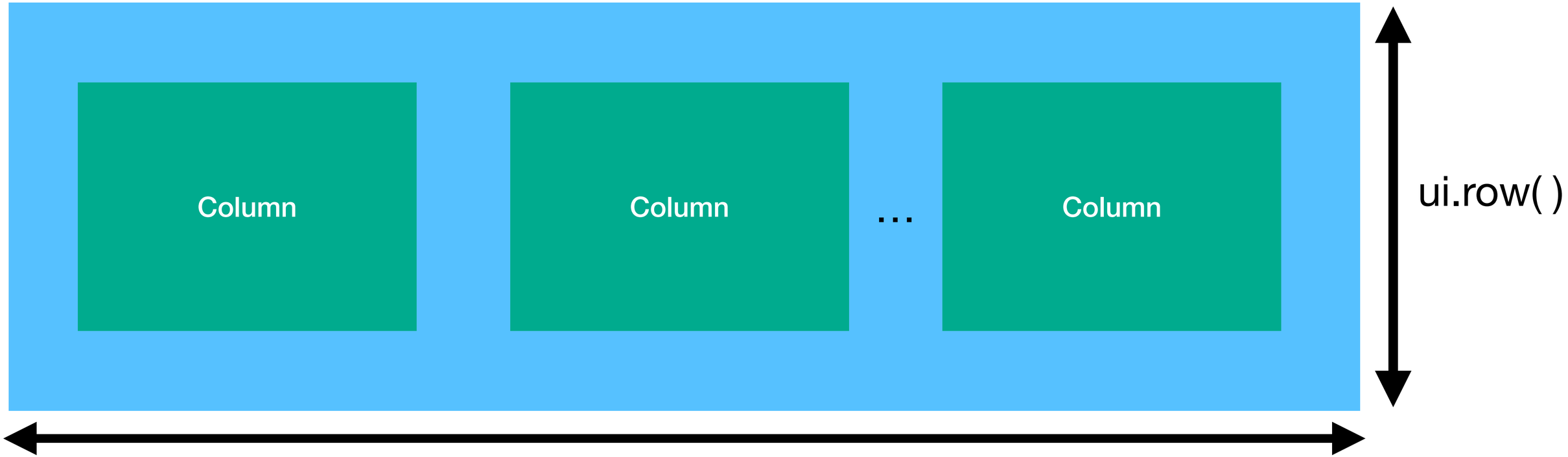



Input
Reactive Value
Output
Output
...
Output

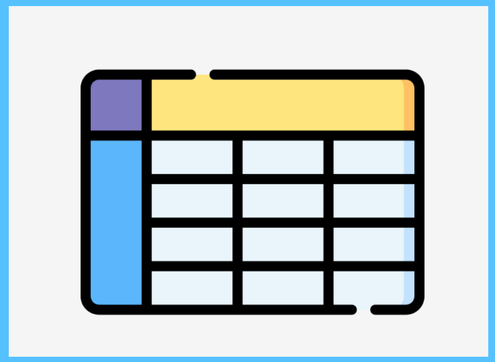
Controlled Reactivity Pattern

```
# Withing the server

reactive_value = reactive.Value(an initial value)

@reactive.effect
@reactive.event(the input that will trigger the RV)
def _():
    ...some code...
    reactive_value.set(A new value)

@output
@render.something
def some_id():
    reactive_value()
```

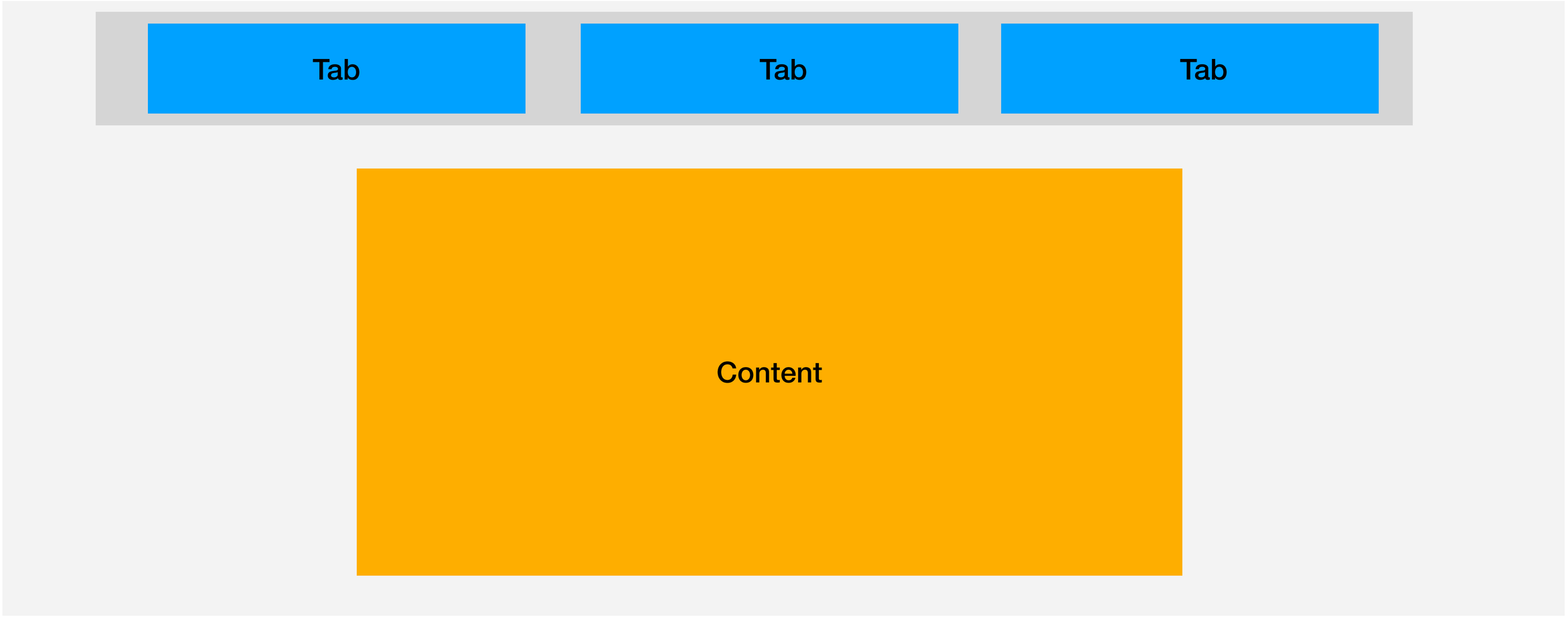
Shiny Apps Layout



Example



Multiple pages



Styling in Shiny

```
css_file = Path(__file__).parent / "file_name.css"

ui.page_fluid(
    ui.include_css(css_file),
    ...some code...
)
```

Thank you