



ISEL – INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
ADEETC – ÁREA DEPARTAMENTAL DE ENGENHARIA DE
ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

LEIM

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA
UNIDADE CURRICULAR DE PROJETO

Gerador de Aplicações Jupyter/Voilà



Tiago Oliveira (45144)

Tiago Gil (46296)

Orientador

Professor João Beleza

julho, 2021

Resumo

Este projeto centra-se na criação de uma plataforma Low-Code, em Python, que permita ao utilizador criar o front-end das suas aplicações web de forma mais rápida e eficiente.

Estas aplicações web têm a restrição de necessitarem de ser criadas como Jupyter Notebooks e transformadas em web apps com a utilização da biblioteca Voilà.

Seguimos o conceito de Single Page Application, ou seja, a nossa plataforma cria aplicações só com “uma página” cujo conteúdo vai sendo atualizado. Utilizamos a tecnologia JSON como ferramenta de exportação e importação.

A própria plataforma é uma aplicação desenvolvida no mesmo paradigma, isto é, é uma aplicação Python/Jupyter Notebook Voilà.

Além da plataforma também produzimos uma biblioteca auxiliar para fornecer suporte ao utilizador, onde o utilizador tem o poder de criar toda a lógica da sua aplicação de forma a complementar o que é produzido com a nossa plataforma.

Motivação

Porquê este projeto?

Na escolha de um projeto a realizar procurámos não só algo do nosso interesse, mas também trabalhar com um orientador que se adequasse ao nosso método de trabalho e com o qual nos identificámos ao longo do curso.

Posto isto, os pontos que nos motivaram a avançar com a proposta foram os seguintes:

- A proposta focar-se em desenvolver uma aplicação que gere aplicações;
- Não existência ou pouca existência de nenhuma plataforma Low-Code em Python nem nada que assemelhe ao que a proposta pretendia;
- Possibilidade de expansão fora âmbito académico e até mesmo comercialização da plataforma;
- A possibilidade de explorarmos melhor os Jupyter Notebooks, utilizados ao longo do curso, e aprender a utilizar a biblioteca Voilà;
- O facto do trabalho ser focado em Python e ser orientado pelo engenheiro João Beleza.

Índice

Resumo	i
Motivação	iii
Índice	v
Lista de Tabelas	ix
Lista de Figuras	xi
1 Introdução	1
2 Trabalho Relacionado	3
2.1 O que é “Low-Code”	3
2.2 OutSystems	3
2.3 Unity	4
3 Tecnologias Utilizadas	5
3.1 Jupyter Notebook	5
3.2 Voilà	5
3.3 Ipywidgets	6
3.4 Ipyvuetify	6
3.5 Css	7
3.6 JSON	7
4 Modelo Proposto	9
4.1 Primeiros Passos	9
4.1.1 Síntese de Objetivos	9
4.1.2 Clientes	10

4.1.3	Metas a Alcançar	10
4.2	Requisitos	10
4.3	Fundamentos	12
4.4	Abordagem	13
5	Implementação do Modelo	15
5.1	“Cabeça”	16
5.1.1	VoilApp	16
5.1.2	Export	17
5.1.3	Loader	19
5.2	“Tronco”	19
5.2.1	Graphics	19
5.3	“Membros”	23
5.3.1	Widget Manager	23
5.3.2	Widget - 21 Variações	24
6	Validação e Testes	25
7	Documentação da Aplicação	29
7.1	Instalação	29
7.1.1	Python	29
7.1.2	Jupyter Notebook	29
7.1.3	Voilà	30
7.1.4	Ipywidgets	30
7.1.5	Ipyvuetify	30
7.1.6	Markdown	31
7.1.7	Nbformat	31
7.2	Primeira utilização	31
7.2.1	Utilizar a Interface	32
7.2.2	Utilizar a Biblioteca	36
7.3	IPyWidget - Incorporação	40
7.4	Widgets - Listagem e Utilização	41
7.4.1	Button	41
7.4.2	CheckBox	43
7.4.3	ColorPicker	44
7.4.4	DatePicker	45

7.4.5	Dropdown	47
7.4.6	FileUpload	48
7.4.7	FloatLogSlider	49
7.4.8	Float Progress	50
7.4.9	Float Text	51
7.4.10	HTML	52
7.4.11	Image	53
7.4.12	Int Progress	54
7.4.13	Int Slider	55
7.4.14	Int Text	56
7.4.15	Label	57
7.4.16	Markdown	58
7.4.17	Password	59
7.4.18	Radio Buttons	60
7.4.19	Text Box	61
7.4.20	Valid	62
8	Conclusões e Trabalho Futuro	63
	Bibliografia	65

Lista de Tabelas

4.1	Funções e atributos do sistema	11
-----	--	----

Lista de Figuras

3.1	Widgets da biblioteca ipyvueify	6
4.1	Cenários de Utilização	12
4.2	UML sistema	13
5.1	Diagrama UML - Geral	15
5.2	Diagrama UML - Cabeça	16
5.3	Classe export	18
5.4	Exportar um Notebook	19
5.5	Classe loader	20
5.6	Diagrama UML - Tronco	21
5.7	Layout appLayout	22
5.8	Inspector - VoilApp	22
5.9	Diagrama UML - Membros	23
6.1	Main Page - Réplica	26
6.2	Main Page - Original	26
6.3	About - Réplica	26
6.4	About - Original	26
6.5	Pre-Answer - Réplica	26
6.6	Pre-Answer - Original	26
6.7	Answer - Réplica	27
6.8	Answer - Original	27
6.9	Código Funcional - VoilApp	28
7.1	Interface VoilApp	32
7.2	Preview VoilApp	32
7.3	Inspetor VoilApp	33

7.4	Alteração atributos widget VoilApp	34
7.5	Ecrãs VoilApp	35
7.6	Menu VoilApp	35
7.7	Função addWidget - VoilApp	36
7.8	Função getScreenNumber - VoilApp	37
7.9	Função getWidgetIDs - VoilApp	37
7.10	Função getWidget - VoilApp	37
7.11	Função setCurrentScreen - VoilApp	38
7.12	Função widgetLoader - VoilApp	40
7.13	Widget Button - VoilApp	41
7.14	Disabled atributo - VoilApp	41
7.15	ButtonOnClick método - VoilApp	42
7.16	Widget Checkbox - VoilApp	43
7.17	Style atributo - VoilApp	43
7.18	Widget ColorPicker - VoilApp	44
7.19	Widget DatePicker - VoilApp	45
7.20	Value atributo- VoilApp	46
7.21	Widget Dropdown - VoilApp	47
7.22	Widget FileUpload - VoilApp	48
7.23	Widget FloatLogSlider - VoilApp	49
7.24	Widget FloatProgress - VoilApp	50
7.25	Widget FloatText - VoilApp	51
7.26	Widget Image - VoilApp	53
7.27	Widget IntProgress - VoilApp	54
7.28	Widget Int Slider - VoilApp	55
7.29	Widget IntText - VoilApp	56
7.30	Widget Password - VoilApp	59
7.31	Widget RadioButtons - VoilApp	60
7.32	Widget TextBox - VoilApp	61
7.33	Widget Valid - VoilApp	62

Capítulo 1

Introdução

A mudança de paradigma em desenvolvimento de software para uma metodologia que se foca em linguagens de alto nível, fez com que a necessidade de plataformas que diminuam o tempo de produção e aumentem a eficiência tenha crescido.

Com isto, as plataformas Low-Code encontraram o seu lugar no mundo informático, pois não só permitem aos seus utilizadores navegar e desenvolver aplicações com um sistema flexível de “drag and drop”, como também tornam possível um utilizador que não tenha grande conhecimento de linguagens de programação crie aplicações simples.

Tendo isto em mente e acreditando que este seria um projeto não só ambicioso como também não muito explorado no mundo de Python, aceitámos a proposta de muito bom grado.

Este projeto gira em torno da linguagem Python, utilizando aplicações web como auxílio do produto final.

Através de documentos no formato Jupyter Notebooks, conseguimos criar documentos capazes de conter tanto código como texto markdown. Uma grande particularidade destes documentos é a possibilidade de criar código em parcelas/células, dando ao utilizador a possibilidade de apenas correr o código que pretender, o que não acontece num ficheiro com um programa Python.

A biblioteca Voilà, que transforma estes documentos em aplicações web, o que previamente seriam células de código e/ou texto corrido, é agora uma página web. Esta biblioteca corre a aplicação num kernel, onde em conjunto com o Jupyter Notebook (que cria todo o JavaScript, CSS e HTML

necessários), apresentam o resultado final num browser.

A aplicação desenvolvida em Python, utiliza bibliotecas como ipywidgets, ipyvuetify e JSON para obter as partes gráficas e também a configuração da aplicação. Como os widgets ipywidgets funcionam como output, foi possível manipulá-los e criar o front-end da nossa própria aplicação

Neste documento relatamos os passos seguidos para a investigação, concepção, desenvolvimento e resultados deste projeto.

Capítulo 2

Trabalho Relacionado

Sendo a nossa plataforma de geração de aplicações Jupyter Voilà, uma plataforma Low-Code, procurámos desde cedo perceber melhor este conceito e explorar não só o que já existia no mercado como também se já existia algo com a mesma finalidade que a proposta que escolhemos.

2.1 O que é “Low-Code”

- O termo que conhecemos nos dias de hoje como “Low-Code”, surge em 2014 para designar plataformas que utilizavam interfaces com base em GUI, o que permite ao utilizador trabalhar e desenvolver código mesmo sem conhecer explicitamente a linguagem de programação específica.
- Cada vez mais as empresas começam a transitar para plataformas de desenvolvimento Low-Code pelas suas interfaces estilo ‘drag and drop’, sendo fácil e intuitiva por permitir arrastar e colocar módulos.

2.2 OutSystems

- Fundada em 2000, a OutSystems é uma empresa portuguesa com uma “plataforma de desenvolvimento Low-Code que possibilita a criação de aplicações utilizando pouco código”.
- Em fevereiro de 2021 a empresa estava valorizada em 9.5 mil milhões de dólares e em 2019 contava com 1228 empregado e tem alcançado diversos prémios na área da tecnologia nos últimos anos.

2.3 Unity

- Unity é um motor de jogos que permite a criação de jogos em multi-plataforma, é utilizado para a criação de plataformas, plugins, video-jogos e até mesmo animações e filmes. Unity é desenvolvido por Unity Technologies e foi lançado em 2005.
- O Unity apresenta um sistema que nos oferece uma série de assets/objetos pré-existentes dentro da plataforma e ao mesmo tempo oferece a possibilidade de modificar todos os seus atributos ou até mesmo desenvolver código complementar a este objeto, sendo este o tipo de inspiração que nos permite o desenvolvimento da nossa aplicação.

Capítulo 3

Tecnologias Utilizadas

3.1 Jupyter Notebook

Um Jupyter Notebook é um documento que pode conter tanto código Python como texto corrido (linguagem Markdown).

Enquanto os Notebooks são documentos, a aplicação que os produz é uma aplicação web (cliente-servidor) que nos dá a possibilidade de editar, correr, criar e apagar estes documentos. Pode ser executada tanto localmente como também pode ser instalada num servidor remoto.

O formato destes documentos é baseado em JSON, onde dados como código, kernel, células e versão da aplicação estão guardados. Possuem “kernels que são processos que correm código interativo numa linguagem de programação particular e devolvem o output apresentado ao utilizador”.

Podemos dividir o conceito de Jupyter Notebook em três componentes:

- Código em tempo real;
- Conteúdo visual in-line (widgets, imagens, gráficos, tabelas);
- Texto explicativo ao longo do documento.

3.2 Voilà

Como o formato notebook não é adequado a todos os públicos-alvo, nomeadamente aos não-programadores, que não querem ver o código, a necessidade da criação de um módulo que escondesse o código, transformando um notebook numa aplicação web, surgiu naturalmente.

Com isto surge esta biblioteca, e.g., [Voilà, 2019], que esconde todas as células que não sejam quer código quer markdown e as corre num kernel que apresenta numa página web. As células de código apesar de serem executadas não são apresentadas no sítio web final, apenas código com output e markdowns são apresentados.

3.3 Ipywidgets

Também conhecidos como Jupyter-widgets, os ipywidgets são widgets interativos em HTML e JavaScript utilizados em Jupyter Notebooks e IPython kernel.

Através dos Notebooks e da biblioteca que transformam o código que normalmente seria estático em código dinâmico, podendo utilizar estes elementos para alterar os valores e visualizar as alterações de dados em tempo real.

Existem dezenas de opções desde botões, sliders, inputs de texto e drop-downs.

3.4 Ipyvuetify

Tal como os “ipywidgets”, a “ipyvuetify” também é uma biblioteca de widgets, com a distinção de terem um aspeto mais moderno.

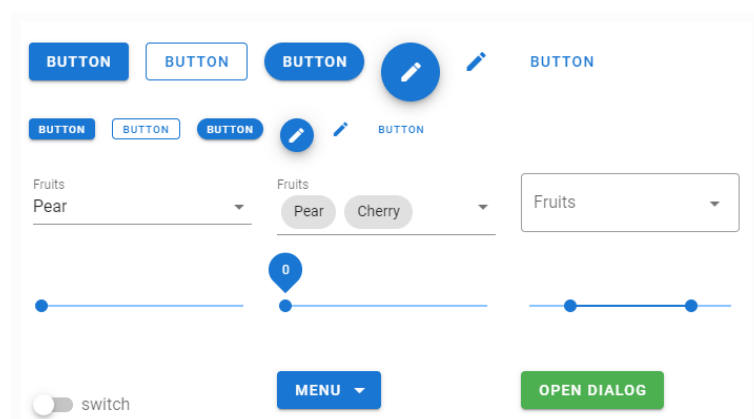


Figura 3.1: Widgets da biblioteca ipyvuetify

Na figura 3.1 observamos que os widgets são baseados na filosofia do design de materiais da Google mais conhecida pela interface do Android.

Como o nome indica, a biblioteca combina a base do ipywidgets com a biblioteca de JavaScript “Vuetify”, sendo que neste projeto utilizamos tanto Jupyter Notebooks como Voilà, estes widgets tiram o máximo de partido da nossa plataforma.

3.5 Css

CSS é a sigla para “Cascading Style Sheets”, Folhas de Estilo em Cascatas em português. É fácil de aprender e é utilizando em conjunto geralmente com HTML.

Foi desenvolvido por W3C (World Wide Web Consortium) em 1996, com o intuito de organizar os estilos numa página web.

Neste caso iremos customizar os widgets que serão adicionados na nossa aplicação e customizar a página da app também.

3.6 JSON

Tendo em questão como iríamos eventualmente conseguir guardar as configurações desenvolvidas e com isso fizemos uma pesquisa sobre os padrões usados neste momento para não tentar reinventar algo já bem definido.

Sendo assim, deparamos com JSON, JavaScript Object Notation, lançado em 2002, que segue uma metodologia de pares chave/valor, ou seja, funciona como se fosse um dicionário, este tipo de notação facilita a serialização dos objetos com que a nossa aplicação lida.

Capítulo 4

Modelo Proposto

4.1 Primeiros Passos

Além da pesquisa das tecnologias que serão utilizadas no projeto, também foi necessário perceber tanto as necessidades da nossa plataforma como também as limitações aplicacionais que iríamos enfrentar.

Como não existe nenhuma aplicação que replicasse o que nos é pedido neste trabalho, precisámos de visar o que já existia e como era executado, desde modelação de dados e vertente visual da aplicação a limitações da linguagem de programação e exemplos a seguir.

Como já foi referido no capítulo 2.3, tentámos replicar o que existe na plataforma Unity, tendo a existência não de componentes 3D/2D mas sim de widgets, que podem ser tanto manipulados na aplicação como posteriormente pelo programador com a biblioteca. Foi também retirada a ideia de inspetor deste mesmo sistema.

Realizámos ainda um documento presente no repositório (01 Analise) que contém uma pequena síntese dos objetivos, o nosso cliente final e as metas que queremos alcançar no final.

4.1.1 Síntese de Objetivos

- Neste projeto será desenvolvida uma aplicação em Jupyter Notebook e Voilà que tem o objetivo de auxiliar o desenvolvimento gráfico(widgets) de futuras aplicações Jupyter/Voilà;
- Será feito seguindo alguns conceitos como "Low-Code" e "single appli-

cation page". Serão desenvolvidas bibliotecas auxiliares para complementarem os Jupyter Notebooks.

4.1.2 Clientes

- Programadores com conhecimento básico e/ou avançado em Python e Jupyter Notebook;
- Utilizador comum com conhecimento mais gráfico que apenas pretenda utilizar a aplicação e não realizar código Python.

4.1.3 Metas a Alcançar

- Ver o código fonte da aplicação;
- Adicionar, editar e remover widgets;
- Guardar o estado de uma aplicação não acabada;
- Utilizar a biblioteca criada para criar novos widgets;
- Gerar aplicações Jupyter Notebook;
- Editar aplicações previamente criadas.
- Permitir a alteração da aplicação sem estar limitado à plataforma.

4.2 Requisitos

Para desenvolver o modelo que estamos a propor e para além das metas delineadas no início da produção do projeto, identificámos os requisitos que seria necessário implementar, quer sejam visíveis ou não.

Como podemos observar na figura 4.1, existem três tipos de categorias, visível, invisível e adorno. As funções visíveis são funções em que o utilizador conhece não só a sua existência como também é visível para o mesmo; tem conhecimento que o sistema oferece essa funcionalidade.

Enquanto uma função invisível apesar de estar implementada no sistema acontece “por detrás do pano”, ou seja, o utilizador não tem noção de que ela acontece mas há a existência de um pedido à plataforma para a executar.

Os adornos são funções que têm o critério opcional e serão como um acessório à aplicação final.

Funções / Atributos do Sistema (Relação entre funções e atributos do sistema)		
Requisito	Função	Categoria
R1	Exportar a Aplicação configurada pelo utilizador	Visível
R2	Editar uma Aplicação previamente exportada	Visível
R3	Antevisão do estado corrente da aplicação	Adorno
R4	Adicionar um Widget específico	Visível
R5	Configurar os detalhes de um Widget	Visível
R6	Apagar um Widget	Visível
R7	Escrita do ficheiro de texto de configuração da aplicação	Invisível
R8	Escrita do ficheiro notebook final da aplicação	Invisível
R9	Exportação do ficheiro de texto de configuração	Invisível
R10	Exportação do ficheiro notebook	Invisível
R11	Exportação da biblioteca desenvolvida para complementar a aplicação	Invisível
R12	Utilização da biblioteca	Invisível
R13	Utilização da biblioteca externamente pelo utilizador caso necessário	Visível

Tabela 4.1: Funções e atributos do sistema

4.3 Fundamentos

Com os requisitos e a estrutura do nosso sistema já definidos, criámos um caso de utilização para a nossa plataforma, pois desta forma conseguimos criar narrativas que simulam uma sequência de eventos provocados pelo autor, ao utilizar o nosso sistema. É-nos também possível demonstrar cenários alternativos a este caso de utilização, caso alguma das ações não vá de acordo com o caso, como podemos ver na figura 4.1.

Casos de Utilização - Definição		(Definição dos casos de utilização)	
ID:	1	Nome:	Desenvolver aplicação
Resumo:	O utilizador utilizando a nossa aplicação, cria a sua interface gráfica, através da adição de widgets suportados pelo sistema desenvolvido, e podendo, ou não, exportá-la.		
Refs.:	R1,R3,R4,R5,R6,R9,R10,R11		
Cenário Principal:			
Acção do Actor		Resposta do sistema	
1	O Caso de Utilização inicia quando o utilizador pretende criar uma nova aplicação		
2	Utilizador adiciona um Widget		
4	Utilizador selecciona um widget a editar	3	Instancia um novo Widget e exhibe o mesmo
6	Utilizador altera informações de um widget	5	Expõe as informações relacionadas com o widget
8	Utilizador exporta/guarda a nova aplicação	7	As informações da instância são alteradas
		9	Geração dos ficheiros referentes à aplicação
Cenário Alternativo 1:			
Número de sequência		Alternativa	
1	Utilizador já possui uma aplicação		Carregamento da aplicação previamente criada
Cenário Alternativo 2:			
Número de sequência		Alternativa	
6	Utilizador apaga um widget		Terminação da instância do widget

Figura 4.1: Cenários de Utilização

4.4 Abordagem

Por fim, e já tendo não só as funções e requisitos do sistema como também um cenário de utilização representativo do uso geral da nossa plataforma, passámos à estrutura funcional do nosso projeto. Definimos as classes necessárias e alguns métodos base para realizarem o que explorámos no capítulo 4.2.

Abaixo, na figura 4.2, conseguimos observar a estrutura inicial do nosso projeto, que na altura da implementação foi completada com as necessidades que fomos encontrando.

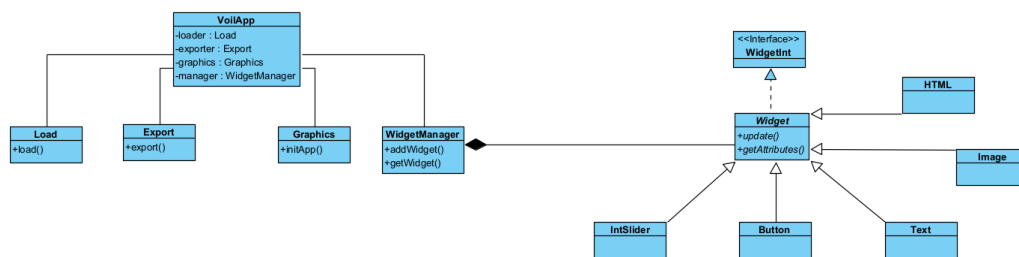


Figura 4.2: UML sistema

Esta divisão em módulos, permite-nos dividir o trabalho em fases e permite que cada uma seja independente da outra. Primeiramente focamo-nos na classe graphics, para termos uma perceção visual de como será de facto a nossa aplicação e juntamos também o widgetManager e as classes dos widgets. Com estes dois módulos prontos temos já uma perceção do nosso produto final, sendo que as classes de export e load em nada influenciam o rumo do trabalho.

Com uma classe VoilApp não finalizada conseguimos testar todas as funcionalidades implementadas.

Escolhemos esta estrutura modular pois não ficamos dependentes de outras implementações. Com a base já implementada o último ponto será focarmo-nos no export e load da nossa aplicação, complementando com a finalização da classe principal VoilApp. Isto permite-nos também ter um desenvolvimento incremental.

Capítulo 5

Implementação do Modelo

Tomando por base o UML apresentado no capítulo anterior o rumo que seguimos a nível de implementação toma por base quase como uma analogia do corpo humano, uma divisão entre cabeça, tronco e membros.

Esta separação leva então a uma divisão de classes como a seguinte:

- “Cabeça” - VoilApp, Loader e Export.
- “Tronco” - Graphics
- “Membros” - Widget Manager e Widgets.

Cada um destes pontos vai estar encarregue de diferentes funções dentro do sistema sendo a cabeça o ponto de junção entre todos.

Na figura 5.1 conseguimos ver o diagrama UML atualizado.

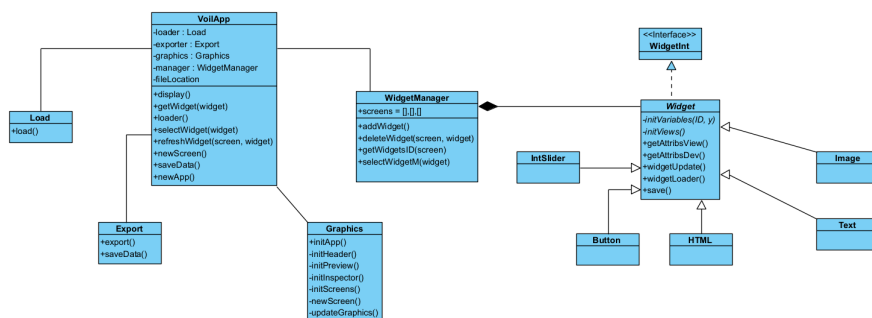


Figura 5.1: Diagrama UML - Geral

5.1 “Cabeça”

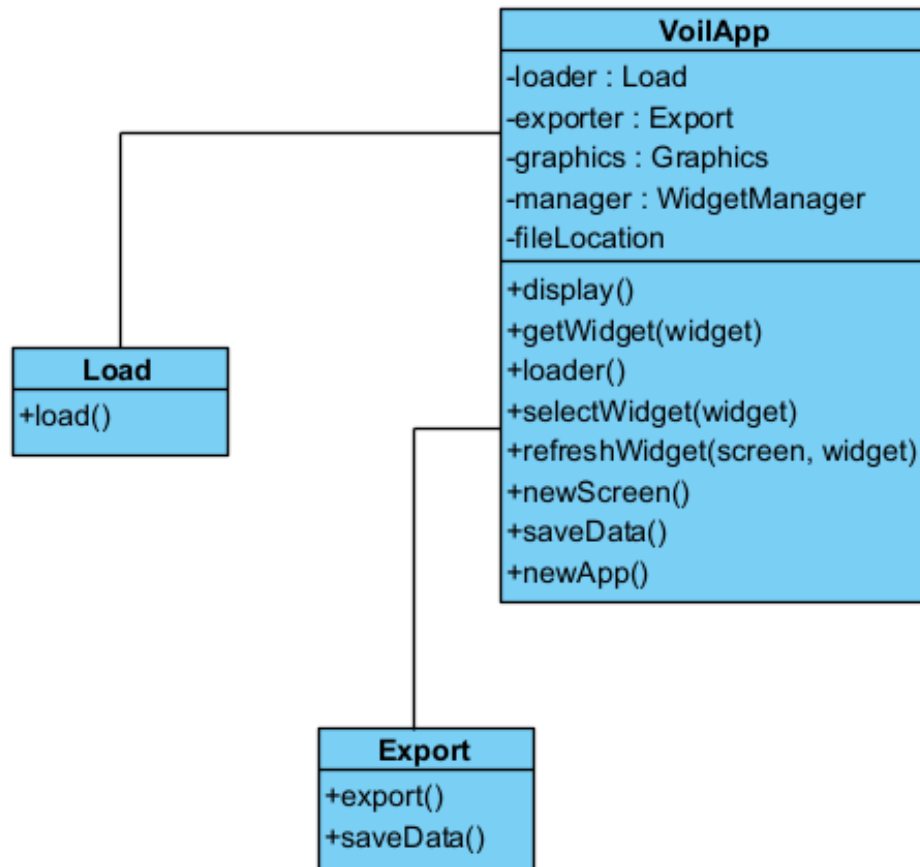


Figura 5.2: Diagrama UML - Cabeça

5.1.1 VoilApp

A classe VoilApp é o ponto central da nossa aplicação, este é o nosso ponto de partida na inicialização da aplicação e de junção de partes.

Com a inicialização de um objeto desta classe todos os outros módulos irão ser sequencialmente chamados conforme um flow definido, além disso esta classe serve também de ponte entre o developer e a aplicação, dando ao utilizador a capacidade de aceder a certos módulos e introduzir novas capacidades á aplicação que produziu com a aplicação.

Para atingir este ponto de papel central foi necessário então uma construção de classe que levasse a tal, daí esta classe começar por ser constituída por um construtor que vai determinar o nível de utilização, com isso queremos dizer que existe uma necessidade de dois parâmetros.

O primeiro remete a uma localização de ficheiro, isto é um pedido para um ficheiro de configuração de uma aplicação já existente caso o utilizador tenha já uma configuração realizada ou será limpo e a aplicação apresenta uma configuração vazia

O segundo parâmetro recebemos um booleano onde indica se a aplicação se encontra em modo de edição ou não, este efeito será relatado mais à frente a nível gráfico. Este parâmetro é necessário para a existência de dois modos, permitindo ter um modo que apresente a interface de edição e permitindo ter outro que não apresente a interface mas apenas o resultado final que será a aplicação web apresentada aos clientes.

Com a inicialização de classe é também feita a inicialização de todos os outros módulos que constituem a aplicação.

Como presente no UML apresentado esta classe implementa diversos métodos alguns destes servindo de comunicação interna na aplicação como uma ligação entre módulos e outros sendo então a tal ponte entre a aplicação e o utilizador. É a partir daqui que podemos também chamar a visualização da interface gráfica que a aplicação irá apresentar.

5.1.2 Export

Antes de se abordar no tópico de load é necessário perceber como o export da aplicação funciona, com isso, uma das primeiras vertentes desta aplicação começou pela ideia de exportar um Jupyter Notebook novo que apresentasse já loaded um ficheiro de configuração, no entanto, esta ideia foi depois modificada, pois se um utilizador quisesse ir gravando alterações à medida que produz a aplicação ia levar a uma criação constante de Notebooks.

Quando inicializado, o módulo export recebe o VoilApp e recebe a localização de ficheiro onde tem de guardar a configuração de sistema, foi também definido que se a localização se encontrar vazia existe um ficheiro default para o qual se irá exportar “config.json”. Assim com acesso a essa informação e a chamada do método saveData() o módulo irá aceder ao widgetManager

através do VoilApp será definido uma raiz para o ficheiro JSON, tendo por base a ideia que iremos ter presente no ficheiro JSON um array por ecrã existente.

Ao percorrermos estes ecrãs presentes no widgetManager vamos entrar dentro de cada um deles e aceder aos widgets presentes, onde através do uso de uma função auxiliar presente em cada um destes widgets, que se encontra definida como `save()`, vamos conseguir que nos seja retornada uma string com a serialização do objeto widget em questão. Esta informação em formato JSON é depois então guardada localmente para que possa ser lida.

```
def saveData(self):
    if(self.fileLocation == ""):
        fileLocation = "config.json"
    else:
        fileLocation = self.fileLocation + ".json"

    #1st Create Array of Export
    self.manager = self.application.widgetManager
    data = {}
    data['Screens'] = []

    #2nd Go Trough all of Screens
    screens = self.manager.screens

    for x in range(len(screens)):
        screen = []
        for y in range(len(screens[x][0])):
            #3rd One Widget at a time get the info necessary
            screen.append(screens[x][0][y].save())
        data['Screens'].append(screen)

    with open(fileLocation, 'w') as f:
        json.dump(data, f)
```

Figura 5.3: Classe export

Ao mesmo tempo que oferecemos a opção de guardar apenas os dados temos também, com a utilização da biblioteca de Python `nbformat`, a ex-


```
#Exports a new Jupyter Notebook ready to be used
def exportNotebook(self):
    nb = nbformat.new_notebook()

    code = "from voilapp import VoilApp\n\napp = VoilApp('#' + self.fileLocation\n+ '', False)\n\napp.display()\n\napp.display()"
    codeCell = nbformat.new_code_cell(code)
    nb['cells'].append(codeCell)
    fname = 'myApplication.ipynb'

    self.saveData()

    with open(fname, 'w') as _:
        nbformat.write(nb, _)
```

Figura 5.4: Exportar um Notebook

Agora com o conhecimento de como o módulo de export funciona e sabendo a existência de um input da localização de um ficheiro de configuração na classe VoilApp, quando VoilApp é iniciado este módulo vai verificar a existência de informação no ficheiro de configuração.

Sendo o ficheiro existente então este vai iterar perante cada posição do array (cada ecrã existente na configuração) e irá na aplicação criar um ecrã por posição, de seguida verificar cada widget que se encontra guardado e inicializá-lo no sistema.

5.2 “Tronco”

5.2.1 Graphics

Chegando à parte de interface gráfica este módulo é encarregue de fazer todo o tipo de processamento visual para a nossa aplicação seja em modo de edição ou não. Aqui é o ponto onde mais se tirou vantagem a nível de utilização das tecnologias como iPywidgets e iPyvuetify, e sendo também onde se verifica

```
def load(self):
    if(self.fileLocation == ".json"):
        self.fileLocation = "config.json"

    if(path.exists(self.fileLocation) == False):
        return
    f = open(self.fileLocation,)

    data = json.load(f)

    counter = -1
    for p in data['Screens']:
        counter += 1
        if(counter > 0):
            self.application.newScreen()
        for x in p:
            widget = self.manager.addWidget(counter,x[0],0,0)
            x.pop(0)
            widget.widgetloader(counter, x)
            self.application.graphics.minID += 1

    self.application.setCurrentScreen(0)
```

Figura 5.5: Classe loader

notavelmente a influencia de inspiração de sistemas como unity.

Para a realização da parte gráfica tiramos proveito de algo que já nos era disponibilizado pelo iPywidgets denominado por AppLayout, este layout permite a fácil construção de interfaces disponibilizando-nos um header, um foco central, duas barras de lado e por fim um footer. Com esta estrutura sendo nos disponibilizada dividimos em diferentes métodos a inicialização deste layout, deste modo também nos permitindo que se houvesse a necessidade de atualizar apenas partes da aplicação o pudéssemos fazer chamando apenas os métodos necessários.

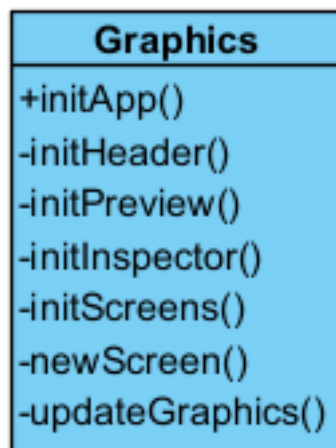


Figura 5.6: Diagrama UML - Tronco

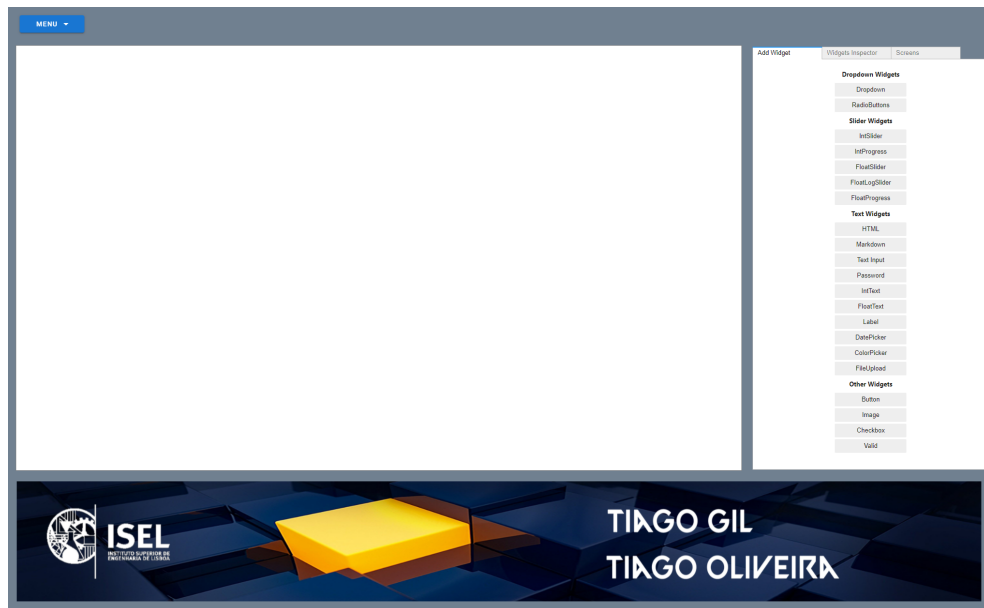


Figura 5.7: Layout appLayout

Com o conhecimento do layout é importante também conhecer as limitações existentes em cada parte do mesmo, começando pela zona de preview que é o ponto central do layout aqui é onde nos é visível aquilo que será o produto final após exportação onde vemos a orientação, posicionamento e até mesmo descrições dos widgets que se encontram inicializados na aplicação. Esta preview encontra-se limitada de momento num sistema de uma box de 50 por 50 entradas.



Figura 5.8: Inspector - VoilApp

No graphics é também onde acontece a distinção entre uma aplicação

terminada e uma aplicação em edição com isto e como dito na classe VoilApp existe aquilo que é um boolean de modo de edição em que este se for True o ecrã apresentará tudo o que foi descrito até agora, no entanto se este se encontrar a False a única coisa que será apresentada no ecrã será a aplicação com as configurações que foram feitas pelo utilizador.

5.3 “Membros”

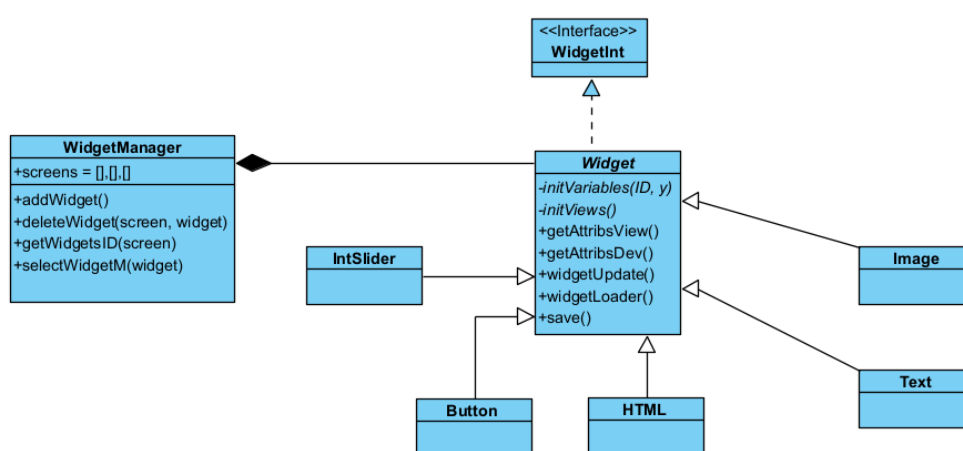


Figura 5.9: Diagrama UML - Membros

5.3.1 Widget Manager

Com todas partes que mantêm a aplicação uma peça única falta aquilo que é a parte crucial a nível de interação e será a produção das aplicações. É necessária a criação de uma classe que trate da manipulação de todos os widgets que vão estar na aplicação e nos diferentes ecrãs presentes nessa mesma.

Para esse efeito foi criado esta classe que é constituída por 3 array tridimensionais que vão definir o ecrã em que estamos, que tipo de widget estamos a lidar com e depois efetivamente qual o widget, respetivamente. Existe a necessidade desta informação pois cada widget será constituído por 3 pontos diferentes, a classe widget definida pelo VoilApp, o widget visual pela qual tomamos partido do iPyWidgets e por fim a representação que se trata de um

botão que aparece no inspector para podermos visualizar as os seus atributos.

É apartir destes arrays que depois a nível gráfico é realizado o preenchimento visual. Mas para além destes arrays esta classe constitui a manipulação dos mesmos, possibilitando através de diferentes métodos a adição, remoção e até edição dos diferentes widgets presentes na aplicação tal como a adição de novos ecrãs.

5.3.2 Widget - 21 Variações

Este é o que acaba por ser a nossa oferta para criar as estruturas das aplicações que os utilizadores vão poder executar, com isso, logo de início foi criada uma estrutura que permitia o suporte de um sistema de edição e ao mesmo tempo de um suporte de visualização final de uma aplicação sem termos de realizar alterações de código ou criar diferentes versões do mesmo widget.

Com isso, um widget tem como atributos o widget de ipywidgets e tem também um widget Button que será o ponto de acesso no inspetor aos atributos de representação.

Os atributos de representação são todos os atributos que o ipywidgets utiliza para poder editar visualmente o Widget, além desses atributos referentes à biblioteca de ipywidgets temos também os 3 atributos bases da aplicação que são referentes ao X, Y e ID, em que estes representam a posição do widget no array de preview e o seu identificador. Através deste mesmo identificador permite depois ao utilizador fazer manipulação através do uso deste.

Existe também um método que possibilita ao Graphics ir buscar a visão de edição de atributos referentes a cada widget com os valores apropriados esta denominada por `getAttribsView()`.

Capítulo 6

Validação e Testes

No início deste projeto foi nos fornecido um exemplo de utilização das tecnologias que constituem a nossa aplicação e ao mesmo tempo algo que servia como exemplo do mínimo que a nossa aplicação deveria ser capaz de atingir no fim de desenvolvimento. Este exemplo trata-se de uma plataforma realizada pelo Engenheiro João Beleza para a realização de trabalhos de casa por parte dos alunos das cadeiras de MDP (LEIM), MCG (LEIM), PG (LMATE) e POO (LMATE) do Instituto Superior de Engenharia de Lisboa.

Esta plataforma é chamada de *questiom* e pode ser acedida através de *questiom.com*, tendo sido este exemplo fornecido pelo nosso orientador, a melhor validação que podemos realizar na verificação dos objetivos definidos no início deste projeto era a capacidade de replicar as páginas e o flow que a plataforma original apresenta. Com isso, decidimos então executar uma réplica da plataforma, esta réplica encontra-se no repositório, na pasta de testes, com o Jupyter Notebook e o ficheiro de configuração json que se encontra associado.

Para começo de validação o primeiro ponto que tivemos em consideração foi usar a interface gráfica do VoilApp para replicar as diferentes páginas presentes na plataforma, nas imagens seguintes vamos referenciar a versão original de *questiom.com* (figuras 6.2, 6.4, 6.6, 6.8) e a nossa versão de réplica (figuras 6.1, 6.3, 6.5, 6.7).

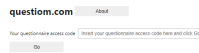


Figura 6.1: Main Page - Réplica

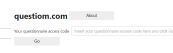


Figura 6.2: Main Page - Original

About questionom.com

[Home](#)

Question stands for Question Mesh

Question is:

- a Web Application to answer questionnaires
- a Desktop Application to build questionnaires

The questions are linked together in a huge mesh that extends to all knowledge areas

Anyone can use Question

Figura 6.3: About - Réplica

About questionom.com

[Home](#)

Question stands for Question Mesh.

Question is:

- a Web Application to answer questionnaires
- a Desktop Application to build questionnaires

The questions are linked together in a huge mesh that extends to all knowledge areas.

Anyone can use Question!

Figura 6.4: About - Original

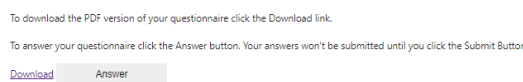


Figura 6.5: Pre-Answer - Réplica

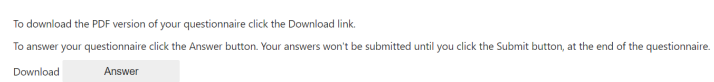


Figura 6.6: Pre-Answer - Original



Considere a classe `CorRGB`, implementada durante as aulas, no módulo `cor_rgb_XXXXX.py` (onde `XXXXX` representa o seu número de aluno), e cuja especificação está disponível no moodle.

Considere também o programa Python 3 que se segue, na mesma pasta/diretória do módulo `cor_rgb_XXXXX.py` (de forma a que o `import` seja executado sem erros). Ignore a variável `seed` e a função `random_float`, que se destinam exclusivamente à geração de números pseudo-aleatórios.

```
from cor_rgb_XXXXX import CorRGB

seed = 1362681
def random_float():
    global seed
    seed = (16807*seed) % 2147483647
    return seed / 2147483646

o1 = []
o2 = []
o3 = []
for n in range(19157):
    o1.append(CorRGB(random_float(), random_float(), random_float()))
    o2.append(CorRGB(random_float(), random_float(), random_float()))
    o3.append(random_float())
```

Acrescente a este programa:

- A lista `o4`. O elemento da lista `o4`, em cada índice, é um objeto `CorRGB` que resulta da soma dos objetos `CorRGB` das listas `o1` e `o2`, no mesmo índice.
- A lista `o5`. O elemento da lista `o5`, em cada índice, é um objeto `CorRGB` que resulta do produto dos objetos `CorRGB` das listas `o1` e `o2`, no mesmo índice.
- A lista `o6`. O elemento da lista `o6`, em cada índice, é um objeto `CorRGB` que resulta do produto do objeto `CorRGB` na lista `o1` pelo `float` da `o3`, no mesmo índice.

Indique se é verdadeiro ou falso.

O valor do atributo `r`, do objeto na lista `o4`, no índice 1135, arredondado com 3 casas decimais usando a função `built-in round`, é 1.1.

- ☒ Verdadeiro
☐ Falso

O valor da lista `o3`, no índice 12258, arredondado com 3 casas decimais usando a função `built-in round`, é 0.454.

- ☒ Verdadeiro
☐ Falso

O valor do atributo `b`, do objeto na lista `o6`, no índice 9941, arredondado com 3 casas decimais usando a função `built-in round`, é 0.598.

- ☒ Verdadeiro
☐ Falso

O valor do atributo `b`, do objeto na lista `o1`, no índice 16956, arredondado com 3 casas decimais usando a função `built-in round`, é 0.789.

- ☒ Verdadeiro
☐ Falso

O valor do atributo `g`, do objeto na lista `o5`, no índice 9005, arredondado com 3 casas decimais usando a função `built-in round`, é 0.372.

- ☒ Verdadeiro
☐ Falso

Submit

Figura 6.7: Answer - Réplica



Considere a classe `CorRGB`, implementada durante as aulas, no módulo `cor_rgb_XXXXX.py` (onde `XXXXX` representa o seu número de aluno), e cuja especificação está disponível no moodle.

Considere também o programa Python 3 que se segue, na mesma pasta/diretória do módulo `cor_rgb_XXXXX.py` (de forma a que o `import` seja executado sem erros). Ignore a variável `seed` e a função `random_float`, que se destinam exclusivamente à geração de números pseudo-aleatórios.

```
from cor_rgb_XXXXX import CorRGB

seed = 1362681
def random_float():
    global seed
    seed = (16807*seed) % 2147483647
    return seed / 2147483646

o1 = []
o2 = []
o3 = []
for n in range(19157):
    o1.append(CorRGB(random_float(), random_float(), random_float()))
    o2.append(CorRGB(random_float(), random_float(), random_float()))
    o3.append(random_float())
```

Acrescente a este programa:

- A lista `o4`. O elemento da lista `o4`, em cada índice, é um objeto `CorRGB` que resulta da soma dos objetos `CorRGB` das listas `o1` e `o2`, no mesmo índice.
- A lista `o5`. O elemento da lista `o5`, em cada índice, é um objeto `CorRGB` que resulta do produto dos objetos `CorRGB` das listas `o1` e `o2`, no mesmo índice.
- A lista `o6`. O elemento da lista `o6`, em cada índice, é um objeto `CorRGB` que resulta do produto do objeto `CorRGB` na lista `o1` pelo `float` da `o3`, no mesmo índice.

Indique se é verdadeiro ou falso.

O valor do atributo `r`, do objeto na lista `o4`, no índice 1135, arredondado com 3 casas decimais usando a função `built-in round`, é 1.1.

- ☐ Verdadeiro
☐ Falso

O valor da lista `o3`, no índice 12258, arredondado com 3 casas decimais usando a função `built-in round`, é 0.454.

- ☐ Verdadeiro
☐ Falso

O valor do atributo `b`, do objeto na lista `o6`, no índice 9941, arredondado com 3 casas decimais usando a função `built-in round`, é 0.598.

- ☐ Verdadeiro
☐ Falso

O valor do atributo `b`, do objeto na lista `o1`, no índice 16956, arredondado com 3 casas decimais usando a função `built-in round`, é 0.789.

- ☐ Verdadeiro
☐ Falso

O valor do atributo `g`, do objeto na lista `o5`, no índice 9005, arredondado com 3 casas decimais usando a função `built-in round`, é 0.372.

- ☐ Verdadeiro
☐ Falso

Submit

Figura 6.8: Answer - Original

Após a réplica destas páginas todas foi necessário assegurar o flow das páginas e então já com o Jupyter Notebook exportado e a utilização da biblioteca desenvolvido através da definição de funções que utilizam a capacidade de mudança de ecrã que o VoilApp oferece demos aos Buttons a capacidade de mudança de ecrã e por fim o print dos resultados de um utilizador a

```
from application import Application
app = Application('', False)

def goAbout(b):
    app.setCurrentScreen(1)

def getExercise(b):
    app.setCurrentScreen(2)

def answerExercise(b):
    app.setCurrentScreen(3)

def goInitial(b):
    app.setCurrentScreen(0)

def submitExercise(b):
    app.setCurrentScreen(0)
    print(app.getWidget(3, "quest1").widget.value)
    print(app.getWidget(3, "quest2").widget.value)
    print(app.getWidget(3, "quest3").widget.value)
    print(app.getWidget(3, "quest4").widget.value)
    print(app.getWidget(3, "quest5").widget.value)

GoAbout = app.getWidget(0, "1")
GoAbout.widget.on_click(goAbout)

GoBTN = app.getWidget(0, "3")
GoBTN.widget.on_click(getExercise)

homeAbt = app.getWidget(1, "5")
homeAbt.widget.on_click(goInitial)

answer = app.getWidget(2, "10")
answer.widget.on_click(answerExercise)

submit = app.getWidget(3, "23")
submit.widget.on_click(submitExercise)

display(app.display())
```

Figura 6.9: Código Funcional - VoilApp

Capítulo 7

Documentação da Aplicação

7.1 Instalação

Para a utilização da VoilApp é necessária a instalação de outros pacotes Python dos quais VoilApp é dependente. Com isso, se estes pacotes não forem instalados irão ocorrer erros ao longo do uso da aplicação.

Abaixo estão presentes todas as dependências e como instalar as mesmas.

7.1.1 Python

Como base para todas as outras dependências da aplicação é necessário a presença de Python no sistema com isso a aplicação é funcional com versões de Python 3.7.x, 3.8.x, 3.9.x ou superior.

<https://www.Python.org/downloads/>

7.1.2 Jupyter Notebook

Para o workspace da aplicação é necessária a instalação de Jupyter Notebook.

Para esse efeito através de linha de comandos com o módulo pip do Python podemos correr o seguinte para a sua instalação:

```
pip install notebook
```

Para verificar se a instalação foi bem sucedida podemos fazer:

```
jupyter notebook
```

7.1.3 Voilà

Para permitir o host da aplicação como web application necessitamos do módulo Voilà que faz a transformação de Jupyter Notebook para web app.

Com isso, podemos proceder à instalação do módulo através da linha de comandos com:

```
pip install voila
```

Para verificar se a instalação foi bem sucedida podemos fazer:

```
voila
```

7.1.4 Ipywidgets

O módulo que vai permitir todos os widgets presentes na VoilApp de serem representados pode ser instalado através da linha de comandos com:

```
pip install ipywidgets
```

7.1.5 Ipyvuetify

Este módulo é necessário na vertente do menu do editor VoilApp, pode ser instalado através da linha de comandos com:

```
pip install ipyvuetify
```

E para permitir a sua incorporação com Jupyter Notebook :

```
jupyter nbextension enable --py --sys-prefix ipyvuetify
```

7.1.6 Markdown

```
pip install markdown
```

7.1.7 Nbformat

Por fim, o módulo que permite a exportação da aplicação diretamente para um novo ficheiro .ipynb (Jupyter Notebook) é instalado pela linha de comandos através de:

```
pip install nbformat
```

Com isto, todas as dependências estão agora instaladas. Nos casos onde não é apresentada a versão do módulo a instalar é porque versões recentes irão sempre funcionar, todas as versões que tenham sido lançadas no último ano.

7.2 Primeira utilização

Após a instalação dos módulos necessários será necessário a clonagem do repositório ou realizar o download do mesmo para o sistema local, através do seguinte link:

```
https://github.com/Iselenos/Projeto/tree/main/03\_Implementacao
```

Com a aplicação e biblioteca no nosso sistema local, podemos agora inicializar o ficheiro Voilapp.ipynb que será o ponto de partida na primeira utilização, com este Notebook devemos abrir este com o Jupyter Notebook e de seguida correr a primeira célula onde podemos depois inicializar o Voilá.

Ficamos então com a seguinte interface inicializada (figura 7.1) onde podemos realizar a nossa aplicação.



Figura 7.1: Interface VoilApp

7.2.1 Utilizar a Interface

No uso de interface para o desenvolvimento de uma aplicação vamo-nos deparar com o cenário apresentado na figura 7.1.

Do lado esquerdo temos o que será a representação da nossa aplicação final, ao que chamamos de módulo de Preview (figura 7.2):

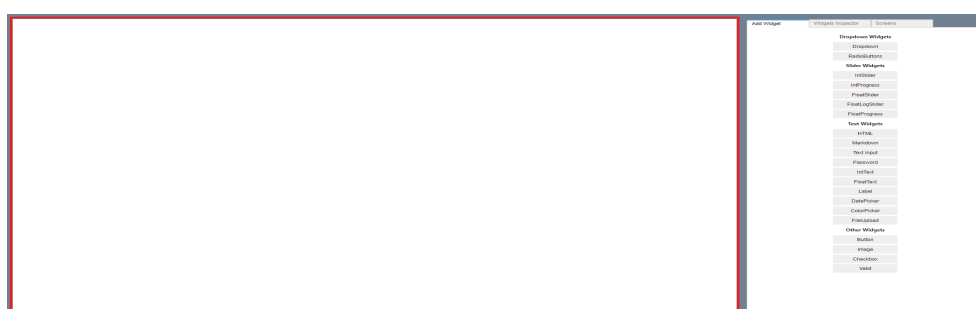


Figura 7.2: Preview VoilApp

Do lado direito é nos apresentado o inspetor (figura 7.3), que contém 3 tabs diferentes, os quais podemos seleccionar cada uma com diferentes fins.

Na primeira tab podemos começar por adicionar todo o tipo de widget que possamos desejar para a nossa aplicação.

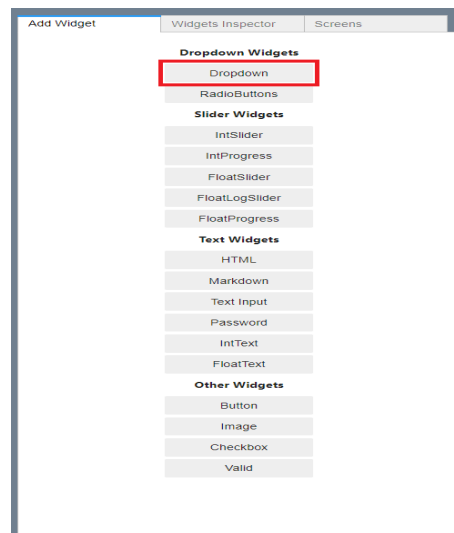


Figura 7.3: Inspetor VoilApp

Com diferentes widgets inicializados na aplicação podemos customizá-los perante as nossas necessidades, podendo alterar diferentes atributos que nos são disponibilizados no inspetor.

Com isso, na segunda tab podemos ver uma lista de todos os widgets presentes no nosso ecrã e seleccioná-los. Quando um widget é seleccionado, são nos apresentados os vários atributos (figura 7.4) e a capacidade de alterar os mesmos. (Cada widget tem a sua gama de atributos que pode ser consultada mais à frente na documentação).

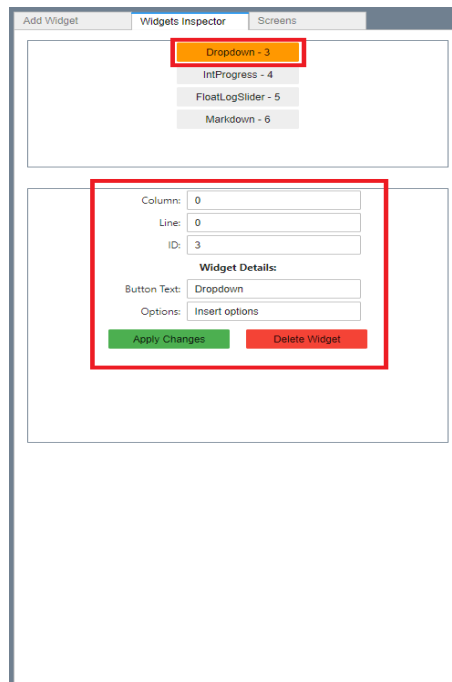


Figura 7.4: Alteração atributos widget VoilApp

Quando terminarmos de realizar alterações podemos clicar em “Apply Changes” que aplica as alterações que realizámos ao widget em questão e visualizar as mesmas na nossa preview.

Outro conceito importante antes de exportarmos ou gravarmos a nossa aplicação é a ideia de suporte a múltiplos ecrãs na aplicação, com isto, a terceira tab no nosso inspetor é referente a este ponto.

Podemos criar um novo ecrã e/ou mudarmos entre ecrãs (figura 7.5 widgets são exclusivos ao ecrã em questão, não são partilhados entre eles).

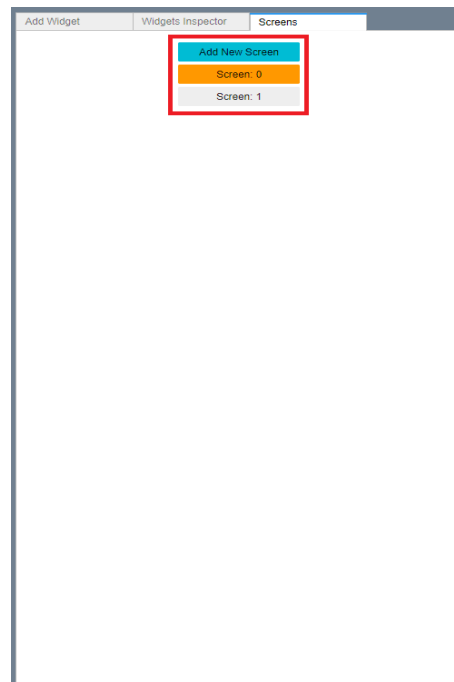


Figura 7.5: Ecrãs VoilApp

Com estes conceitos em mente já podemos construir o front-end da nossa aplicação. Quando terminada podemos guardar o ficheiro de configuração da app ou até mesmo exportar para um novo Jupyter Notebook com referência ao menu que temos presente (figura 7.6).

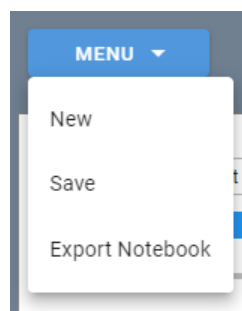


Figura 7.6: Menu VoilApp

Por fim, temos também uma opção presente que é a “New” permitindo começar uma nova aplicação.

7.2.2 Utilizar a Biblioteca

Esta biblioteca pode ser utilizada em conjunto com o ficheiro Jupyter Notebook exportado, ou, pode ser também utilizada fazendo um programa complementar em Python a ser utilizado com a plataforma/aplicação.

addWidget (screen,widget):

Params:

Screen - número do ecrã a adicionar o widget em (0 a x).

Widget - string a referenciar o tipo de widget a adicionar (WidgetList).

Return:

Widget - instância do widget inicializado.

WidgetList:

"Button","HTML","Markdown","Text","Input","Image","IntSlider","FloatSlider",
"FloatLogSlider","IntProgress","FloatProgress","IntText","FloatText","Checkbox",
"Valid","Dropdown","RadioButtons","Password","Label","DatePicker",
"ColorPicker","FileUpload".

Este exemplo (figura 7.7) mostra a inicialização de um Button widget no primeiro ecrã (0).

```
#Add new Widget  
wid = app.addWidget(0,"Button")
```

Figura 7.7: Função addWidget - VoilApp

getScreenNumber ():

Params: Não tem.

Return:

Length - tamanho do array de ecrãs presentes.

Exemplo (figura 7.8):

getWidgetIDs (screen):

```
app.getScreenNumber()
```

```
3
```

Figura 7.8: Função getScreenNumber - VoilApp

Params:

Screen - número do ecrã a adicionar o widget em (0 a x).

Return:

IDs - array de strings com tipo e ID de widgets presentes no ecrã.

Exemplo (figura 7.9):

```
app.getWidgetIDs(0)
```

```
['Dropdown - 0', 'Radio Button - 1', 'Button - 2']
```

Figura 7.9: Função getWidgetIDs - VoilApp

getWidget (screen, ID):

Params:

Screen - número do ecrã a adicionar o widget em (0 a x).

ID - identificador do widget que se quer.

Return:

Widget - widget com o respetivo identificador.

Exemplo (figura 7.10):

```
wid = app.getWidget(0, "0")|
```

Figura 7.10: Função getWidget - VoilApp

setCurrentScreen (number):

Params:

Number - número do ecrã desejado.

Return:

Nenhum.

Exemplo (figura 7.11):

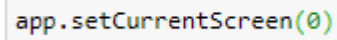
A screenshot of a code editor showing the function call `app.setCurrentScreen(0)`. The text is color-coded: `app` is blue, `.` is red, `setCurrentScreen` is green, and `(0)` is blue. The code is enclosed in a light gray rectangular box.

Figura 7.11: Função `setCurrentScreen` - VoilApp

saveData():

Params:

Nenhum.

Return:

Nenhum.

Exemplo Este método guarda no ficheiro de configuração JSON qualquer alteração que seja feita aos ecrãs ou variáveis de widgets:

getAttribsDev ():

Params:

Nenhum.

Return:

Atributos - retorna array com todos os atributos referente ao widget.

Exemplo Este array pode ser alterado e usado em conjunto com o método widgetLoader do widget.

widgetLoader (screen, attribs):

Params:

Screen - número de referencia ao ecrã do widget.

Attribs - array de atributos obtidos através de getAttribsDev() .

Return:

Nenhum.

Exemplo Este método (figura 7.12) procede à atualização dos dados do

widget consoante o array de atributos, o significado de cada posição do array pode ser visualizado na documentação na página de cada widget diferente.

```
widAttribs[1] = 2 #Update para ser a terceira linha  
widAttribs[4] = "Submit" #E o texto do button passa a ser Submit  
wid.widgetloader(0 ,widAttribs) #Update do button com esta informacao
```

Figura 7.12: Função widgetLoader - VoilApp

7.3 IPyWidget - Incorporação

Pondo em consideração os métodos introduzidos pelo VoilApp o nível de implementação não se encontra limitado ao que foi apresentado, existindo um nível de desenvolvimento mais profundo. Pode ser usado em conjunto com esta biblioteca, a biblioteca de iPyWidgets que foi usada como base para os widgets implementados.

Dito isto, qualquer método permitido por essa biblioteca pode, por sua vez ser, utilizado aqui também, qualquer widget que não se encontre implementado pode também ser implementado seguindo a interface que se encontra definida na biblioteca VoilApp.

7.4 Widgets - Listagem e Utilização

Através da utilização da biblioteca iPyWidgets foram incorporados os seguintes widgets na aplicação.

7.4.1 Button

Utilidade:

Define uma zona de interação à qual pode ser fornecida diversos métodos de execução por parte do programador.



Figura 7.13: Widget Button - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado no Button.

Tooltip [5] - texto de hover.

Style [6] - referencia para texto CSS ou para Styles pré-definidos pela biblioteca ipywidgets.

Atributos (iPyWidgets):

Icon - define um icon que aparece no Button.

Disabled - define se o widget está ativo.

```
wid.widget.disabled = False
```

Figura 7.14: Disabled atributo - VoilApp

Métodos (iPyWidgets):

on_click() - permite definir uma função a ser executada como callback na ação de click.

```
def buttonscreen1(b):  
    app.setCurrentScreen(1)  
  
wid = app.addWidget(0,"Button")  
wid.widget.on_click(buttonscreen1)
```

Figura 7.15: ButtonOnClick método - VoilApp

7.4.2 CheckBox

Utilidade:

Interface de duplo estado, representada por uma caixa que consoante o estado apresenta ou não um símbolo de check.

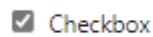


Figura 7.16: Widget Checkbox - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado no Button.

Value [5] - estado da checkbox (true ou false).

Atributos (iPyWidgets):

Disabled - define se o widget está ativo (figura 7.14).

Style - define o css associado ao widget.

```
wid.widget.style = "CSS"
```

Figura 7.17: Style atributo - VoilApp

7.4.3 ColorPicker

Utilidade:

Permite a definição de uma interface que dá ao utilizador a capacidade de escolher uma cor variante nos valores RGB.

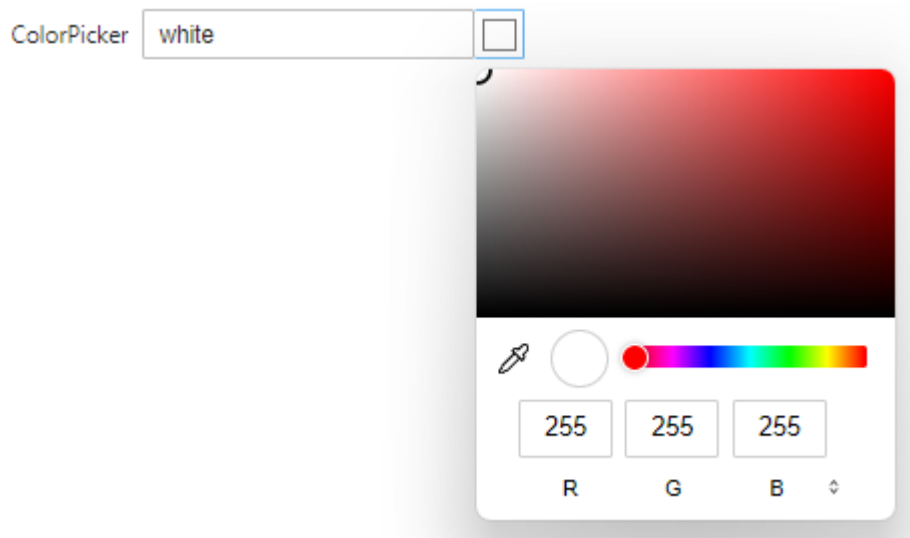


Figura 7.18: Widget ColorPicker - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado no Button.

Value [5] - os valores referentes à cor selecionada.

Atributos (iPyWidgets):

Disabled - define se o widget está ativo (figura 7.14).

7.4.4 DatePicker

Utilidade:

Permite a definição de uma interface que dá ao utilizador a capacidade de escolher uma data. Este é baseado no campo input de data disponibilizado em HTML.

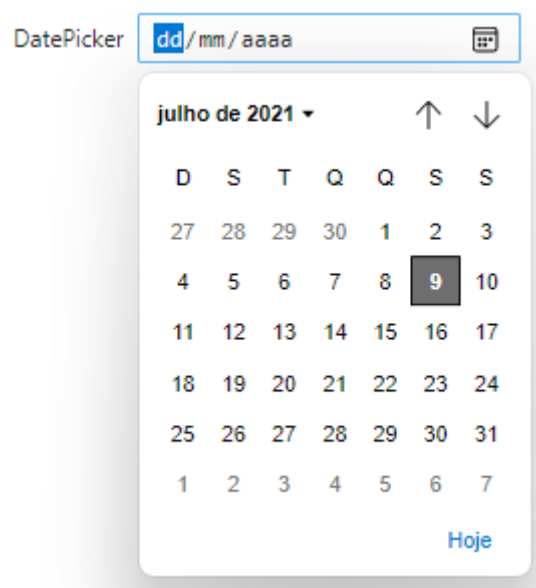


Figura 7.19: Widget DatePicker - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

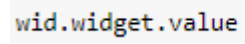
Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado no Button.

Atributos (iPyWidgets):

Value - valor de data escolhido.



```
wid.widget.value
```

Figura 7.20: Value atributo- VoilApp

Disabled - define se o widget está ativo (figura 7.14).

7.4.5 Dropdown

Utilidade:

Interface que permite apresentar diferentes opções de conteúdo a um utilizador para este poder seleccionar determinada opção.

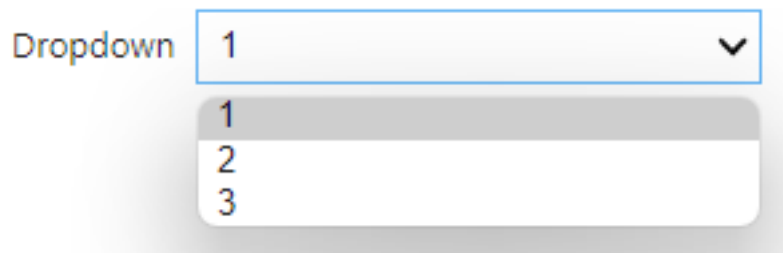


Figura 7.21: Widget Dropdown - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado no Button.

Options [5] - os valores apresentados como opções na interface (estes são uma string dividida por virgulas entre cada opção).

Atributos (iPyWidgets):

Style - define o css associado ao widget (figura 7.17).

Disabled - define se o widget está ativo (figura 7.14).

7.4.6 FileUpload

Utilidade:

Permite o upload de ficheiros.

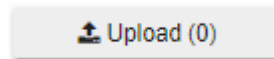


Figura 7.22: Widget FileUpload - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Accept [4] - extensões de ficheiros aceites.

Multiple [5] - variável boolean que define se é aceite ou não múltiplos ficheiros ao mesmo tempo.

Atributos (iPyWidgets):

Style - define o css associado ao widget (figura 7.17).

Disabled - define se o widget está ativo (figura 7.14).

Value - path do ficheiro escolhido (figura 7.20).

Notas:

O ficheiro que é feito o upload vai ter os seus conteúdos armazenados numa memory view de Python, que pode ser acedido indo buscar o atributo de value do widget e, após isso, acedendo ao content desse value.

7.4.7 FloatLogSlider

Utilidade:

Um slider que permite a variação de um valor de tipo float entre um valor máximo e mínimo com representação visual de uma função logarítmica.



Figura 7.23: Widget FloatLogSlider - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado antes do widget.

Value [5] - valor corrente.

Min [6] - valor mínimo do slider.

Max [7] - valor máximo do slider.

Step [8] - valor incremental entre cada passo.

Base [9] - base logarítmica.

Atributos (iPyWidgets):

Style - define o css associado ao widget (figura 7.17).

Disabled - define se o widget está ativo (figura 7.14).

7.4.8 Float Progress

Utilidade:

Representação visual de uma barra de progresso definida por valores de tipo float.



Figura 7.24: Widget FloatProgress - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado antes do widget.

Value [5] - valor corrente.

Min [6] - valor mínimo do widget.

Max [7] - valor máximo do widget.

Bar_style [8] - referencia para texto CSS ou para Styles pré-definidos pela biblioteca ipywidgets.

Orientation [9] - orientação visual do widget(vertical ou horizontal).

Atributos (iPyWidgets):

Disabled - define se o widget está ativo (figura 7.14).

7.4.9 Float Text

Utilidade:

Caixa de texto que apenas permite introdução de texto numérico em tipo integer ou float.



Figura 7.25: Widget FloatText - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado antes do widget.

Value [5] - valor corrente.

Atributos (iPyWidgets):

Disabled - define se o widget está ativo (figura 7.14).

Style - define o css associado ao widget (figura 7.17).

7.4.10 HTML

Utilidade:

Widget que aceita linguagem HTML.

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Value [4] - valor corrente.

Desc [5] - texto apresentado antes do widget.

Placeholder [6] - texto apresentado no lugar do widget.

7.4.11 Image

Utilidade:

Representação de uma imagem que pode ser escolhida através do upload de um ficheiro.



Figura 7.26: Widget Image - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Width [4] - comprimento da imagem.

Height [5] - altura da imagem (este valor não tem influencia na representação por limitações vindas da biblioteca iPyWidgets).

Value [6] - dados referentes à imagem (bytes).

7.4.12 Int Progress

Utilidade:

Representação visual de uma barra de progresso definida por valores de tipo integer.



Figura 7.27: Widget IntProgress - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado antes do widget.

Value [5] - valor corrente.

Min [6] - valor mínimo do widget.

Max [7] - valor máximo do widget.

Bar_style [8] - referencia para texto CSS ou para Styles pré-definidos pela biblioteca ipywidgets.

Orientation [9] - orientação visual do widget(vertical ou horizontal).

Atributos (iPyWidgets):

Disabled - define se o widget está ativo (figura 7.14).

7.4.13 Int Slider

Utilidade:

Um slider que permite a variação de um valor de tipo integer entre um valor máximo e mínimo com representação visual.



Figura 7.28: Widget Int Slider - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado antes do widget.

Value [5] - valor corrente.

Min [6] - valor mínimo do widget.

Max [7] - valor máximo do widget.

Step [8] - valor incremental entre cada passo.

Orientation [9] - orientação visual do widget(vertical ou horizontal).

Atributos (iPyWidgets):

Disabled - define se o widget está ativo (figura 7.14).

Style - define o css associado ao widget (figura 7.17).

7.4.14 Int Text

Utilidade:

Caixa de texto que apenas permite introdução de texto numérico em tipo integer.



Figura 7.29: Widget IntText - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado antes do widget.

Value [5] - valor corrente.

Atributos (iPyWidgets):

Disabled - define se o widget está ativo (figura 7.14).

Style - define o css associado ao widget (figura 7.17).

7.4.15 Label

Utilidade:

Caixa de texto que apenas permite introdução de texto numérico em tipo integer.

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Value [4] - valor corrente.

Atributos (iPyWidgets):

Style - define o css associado ao widget (figura 7.17).

7.4.16 Markdown

Utilidade:

Suporta linguagem markdown para representação visual. Este foi implementado de modo a permitir que o utilizador final não tenha de formatar texto em HTML.

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Value [4] - valor corrente.

Desc [5] - texto apresentado antes do widget.

Placeholder [6] - texto apresentado no lugar do widget.

7.4.17 Password

Utilidade:

Caixa de texto, mas com suporte a confidencialidade de introdução.



Figura 7.30: Widget Password - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado antes do widget.

Placeholder [5] - valor apresentado quando widget se encontra vazio.

Value [6] - valor corrente.

Atributos (iPyWidgets):

Disabled - define se o widget está ativo (figura 7.14).

Style - define o css associado ao widget (figura 7.17).

7.4.18 Radio Buttons

Utilidade:

Múltiplas opções do qual o utilizador pode apenas selecionar uma.

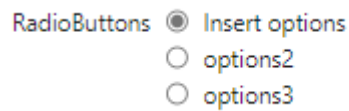


Figura 7.31: Widget RadioButtons - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado no Button.

Options [5] - os valores apresentados como opções na interface (estes são uma string dividida por virgulas entre cada opção).

Value [6] - valor corrente.

Atributos (iPyWidgets):

Style - define o css associado ao widget (figura 7.17).

Disabled - define se o widget está ativo (figura 7.14).

7.4.19 Text Box

Utilidade:

Caixa de texto básica.

Text Input

Figura 7.32: Widget TextBox - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado antes do widget.

Placeholder [5] - valor apresentado quando widget se encontra vazio.

Atributos (iPyWidgets):

Style - define o css associado ao widget (figura 7.17).

Disabled - define se o widget está ativo (figura 7.14).

Value - path do ficheiro escolhido (figura 7.20).

7.4.20 Valid

Utilidade:

Representacao visual de um valor booleano.



Figura 7.33: Widget Valid - VoilApp

Atributos (VoilApp):

Notação - [Posição referente ao array de atributos do widget.]

X [0] - coluna de apresentação.

Y [1] - linha de apresentação.

ID [2] - identificador.

Desc [4] - texto apresentado antes do widget.

Value [5] - valor corrente do widget (boolean).

Atributos (iPyWidgets):

Style - define o css associado ao widget (figura 7.17).

Capítulo 8

Conclusões e Trabalho Futuro

Apesar de terem sido atingidos os objetivos definidos ainda existe muitos pontos que poderiam ser adicionados a este projeto, sendo alguns de complexidade elevada.

Um dos pontos que gostaríamos de ver este trabalho atingir num futuro seja por nós ou um aluno que pegue no nosso projeto e o decida continuar, era a ideia de conseguir transformar o que está definido como a zona de preview da aplicação e adicionar as capacidades de tornar a movimentação de widgets na aplicação em “drag and drop”.

Um ponto como este implica fazer reverse engineering numa das tecnologias usadas que é o iPyWidgets e conseguir entender como é feita a transformação de Python para as linguagens Web que o browser utiliza, conseguindo fazer uma aplicação destas abre também a porta a um número imenso de capacidades que poderiam ser implementadas na aplicação como por exemplo edição de widgets diretamente na preview.

Outro ponto seria também adicionar a ideia de prefabs, uma compilação de widgets num container para criar um widget mais complexo, por exemplo, um widget Login que contenha todos os pontos relevantes a um sistema desses.

Com isto, deixamos em aberto ao docente orientador Engenheiro João Beleza uma eventual continuação futura deste projeto se assim o desejar e a alunos que possam fazer o mesmo, poderem pegar na nossa aplicação e melhorá-lo com a base que já tem.

Concluindo, este projeto deu-nos uma oportunidade de aprender mais sobre as ferramentas que tocámos durante o nosso percurso académico (3.1)

e solidificar aspetos da linguagem Python. Tivemos ainda oportunidade de conhecer bibliotecas que não tínhamos abordado ainda (3.2) e usar noutro âmbito o formato JSON.

Este projeto provou também a possibilidade de desenvolver plataformas web, Low-Code, em Python. Mostrou também a sua eficiência sendo capaz de reproduzir em cerca de 5 minutos uma aplicação que demorou vários dias a ser produzida (questiom.com).

Este sistema permitiu ainda libertar o programador da repetibilidade que é a programação de interfaces de utilizador, tornando-a Low-Code e fácil.

Ficámos satisfeitos com os resultados que obtivemos e pela hipótese de termos produzido este projeto e não outro dentro das opções que nos foram fornecidas.

Achamos que produzimos algo que pode chegar longe no mercado, sendo que não existe nada de parecido na atualidade e com a normalização de web apps nos dias de hoje tem potencial para chegar a uns níveis interessantes, sendo que o Low-Code é “moda” por enquanto.

Bibliografia

[Voilà, 2019] Voilà (2019). Voilà. <https://github.com/voila-dashboards/voila/tree/stable>.