DESENVOLVIMENTO DE APLICAÇÕES MULTIPLATAFORMA COM KIVY[1]

RICARDO AUGUSTO DA SILVA MATTOS[2], FELIPE DINIZ DALLILO[3]

Universidade de Araraquara (UNIARA)

Resumo: O presente trabalho busca apresentar as vantagens, desvantagens e usabilidade da biblioteca Kivy do python para o desenvolvimento de aplicações que podem ser executadas em diferentes plataformas e aceita até cinco toques/cliques simultâneos por meio de uma aplicação desenvolvida que utiliza um CRUD para gerenciar idéias de forma rápida e eficiênte a fim de incluir essa tarefa na rotina pessoal.

Palavras-chave: Python. Kivy. Aplicação. Desenvolvimento. Componentes.

Abstract: *The present work seeks to present the advantages, disadvantages and usability of the Kivy library of python for the development of applications that can be executed in different platforms and accepts up to five simultaneous touches / clicks through a developed application that uses a CRUD to manage ideas quickly and efficiently in order to include this task in the personal routine.*

Keywords: *Python. Kivy. Application. Development. Components.*

---

[1] Trabalho apresentado na Universidade de Araraquara – UNIARA.

[2] Graduando no curso de Sistemas de Informação da Universidade de Araraquara – SP.

ricardo.svmtts@gmail.com

[3] Docente na área de Sistemas de Informação da Universidade de Araraquara – SP.

1 INTRODUÇÃO

A procura por *softwares* que atinjam o maior número possível de usuários obriga os desenvolvedores a procurar por soluções computacionais que permitam o desenvolvimento ágil alinhado com a necessidade de atender ao maior número possível de plataformas uma vez que seus espectadores se concentram nas mais diversas aplicações *multiplataforma* como: *Firefox*[1], *Google Chrome*[2], *TeamViewer*[3] (EMPORIO, 2010).

Para este propósito, existem diversas abordagens e tecnologias que apoiam o desenvolvimento do *software* com o intuíto de facilitar o trabalho. Uma dessas formas, é a utilização da linguagem de programação *Python*[4] junto a Biblioteca *Kivy*[5].

Kivy é uma ampla biblioteca *open-source*[6] do *Python* criada para o desenvolvimento de *interface*[7] de usuário e pode ser utilizada no desenvolvimento de utilitários e jogos. Aplicações desenvolvidas com essa biblioteca aceitam até cinco toques simultâneos, o que a torna interessante para o desenvolvimento de jogos e podem ser executado em diversas plataformas como: *Linux, Windows, OS X, Android e iOS* (KIVY, 2016).

Atualmente há dificuldade em processar e armazenar um volume tão grande de dados devido ao excesso de informação, porém, o simples hábito de fazer anotações pode ajudar pois as vantagens são várias: libera a mente para racionar sem precisar lembrar de coisas importantes, permite o resgate de informações sem esforço, facilita a priorização do que é importante e reduz drasticamente esquecimento e lapsos de memória (FALANDODEGESTAO, 2011).

Este artigo propõe o desenvolvimento uma aplicação para estimular o hábito fazer anotações, por meio de um CRUD (*Create Read Update Delete*) utilizando *Kivy* para demonstrar o uso desta tecnologia na criação de *interfaces* de usuário e destacar as suas vantagens e desvantagens.

A aplicação desenvolvida, *Thinking Manager*, possui um *layout* dinâmico alternando suas cores estimulando a criatividade. Ela permite o cadastro de ideias de forma fácil e rápida para que

---

[1] *www.mozilla.org/firefox*
[2] *www.google.com/chrome*
[3] *www.teamviewer.com/pt/*
[4] *www.python.org/*
[5] *https://kivy.org/#home*
[6] *Open-source* – Código aberto
[7] *Interface* - Usuário consegue, usando um computador, interagir com um programa ou com um sistema operacional

depois possam ser consultadas. Após o cadastro, essas ideias estarão registradas e não serão perdidas. Dessa forma, o usuário não se preocupa em esquece-las, deixando a mente mais livre e pode planeja-las como julgar necessário.

2 TECNOLOGIAS ADOTADAS NA APLICAÇÃO

Nesta sessão, serão apresentadas algumas características das tecnologias *Python*, *SQLite* e *Kivy* utilizadas no desenvolvimento da aplicação.

2.1 *Python*

Python é uma linguagem de programação criada no fim da década de 80 por Guido Van Rossum. A primeira ideia de implementar o *Python* surgiu em Amsterdã, Holanda, enquanto Guido participava de um projeto no CWI (*Centrum Wiskunde & Informatica*, Centro de Matemática e Ciência da Computação) num time do qual iria desenvolver a linguagem ABC(MINDBENDING, 2014).

Segundo Guido(2014), a nova linguagem deveria preencher o vazio entre o C[8] e o *Shell Script*[9]. Esse foi o objetivo principal na criação do *Python*.

Python é uma linguagem alto nível, de script, tipagem dinâmica e orientada a objetos. Pode ser aprendida facilmente tanto por programadores experiêntes em outras linguagens quanto por programadores iniciantes por sua sintaxe limpa.

A figura 1 demonstra o uso da linguagem:

[8] C – Linguagem de programação estruturada
[9] *Shell Script*- Linguagem de programação usada em sistemas operacionais como interpretador de comandos

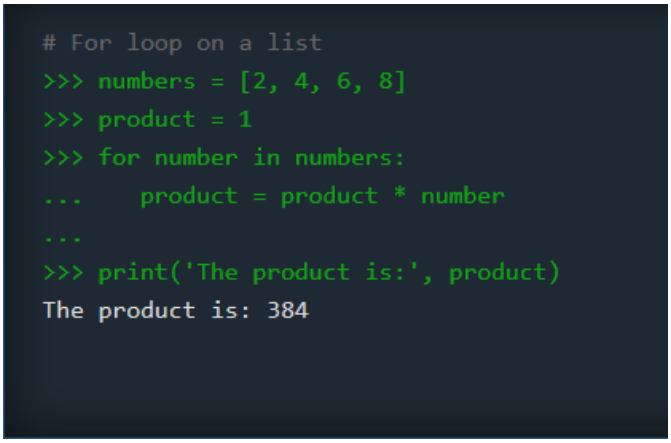

Figura 1 – Exemplo de uso da linguagem Python.
Fonte: Site oficial Python.

A linguagem está presente em grandes empresas. Segundo Cuong(2007), arquiteto de *software* do YouTube, "*Python* é rapido o bastante para nosso site, e nos permite dar manutenção em tempo recorde com o mínimo de desenvolvedores". Segundo Norvig, diretor de qualidade de buscas no *Google*, "*Python* sempre foi uma importante parte do *Google* e continua até hoje nos sistemas que o envolvem".

2.2 SQLite

SQLite [10]é uma biblioteca multiplataforma escrita na linguagem C que ao ser implementada no projeto, permite acesso a um amplo banco de dados SQL. Seu código de domínio publico sendo livre para uso em qualquer propósito, seja comercial ou privado(SQLITE, 2016).

É o banco de dados mais utilizado no mundo com milhares de aplicações ainda que seja voltado para aplicações menos complexas por não fazer uso de um Sistema Gerenciador de Banco de Dados(SGBD) como *Oracle* [11]e *MySQL*(SQLITE, 2016).

O *SQLite* não exige instalação, apenas sua integracação como uma biblioteca no programa em desenvolvimento. Isso se torna bastante útil quando é preciso entregar um sistema integrado a um banco de dados, mas não há possibilidade de organizar um servidor de banco de dados (REVISTABW, 2013).

O quadro 1 apresenta as principais vantagens e desvantagens do banco de dados oferecido pela biblioteca SQLite:

[10] *https://sqlite.org/*
[11] *www.oracle.com/pt*

Vantagens	**Desvantagens**
Baseado em arquivo: O banco de dados consiste num único arquivo no disco, tornando-o extremamente portável.	Sem gerenciamento de usuários: Não possui suporte a usuários(ex.: conexões gerenciadas por um conjunto de privilégios de acesso ao banco e tabelas) visto que seu uso não terá altos níveis de concorrência multicliente.
Atento aos padrões: Apesar da simples implementação, o banco de dados SQLite utiliza SQL.	Falta de possibilidade de alteração para obter performance adicional: Com SQLite, não é possível alterações para um ganho de performance excepcional. Apesar da simplicidade de configuração e uso da biblioteca, não é tecnicamente possível faze-lo mais performático do que ele incrivelmente já é.
Ótimo para desenvolvimento e para testes: SQLite oferece a simplicidade de se trabalhar com um único arquivo e mais do que necessário para o desenvolvimento e testes da aplicação.	

Quadro 1 – Vantagens e Desvantagens do SQLite.
Fonte: Blog Desenvolvedor(2015).

De maneira geral, a melhor utiização do *SQLite* seria em aplicações embarcadas(todas aplicações que não requerem expansão, por exemplo, aplicativos locais com um único usuário, aplicaticos móveis ou jogos) e na substituição de acesso à disco (aplicações que precisem ler/escrever informações diretamente no disco podem se beneficiar da mudança para o *SQLite*, com a adição de funcionalidade e simplicidade do SQL) (BLOG DESENVOLVEDOR, 2015).

2.3 Kivy

Kivy é uma biblioteca de código aberto do *Python* que oferece ferramentas para um rápido desenvolvimento de aplicações que fazem uso de interfaces de usuário. As aplicações desenvolvidas com essa ferramenta podem ser executadas nas plataformas *Linux*, *Windows*, *OS X*, *Android* e *iOS*. O motor gráfico é construído sobre *OpenGL ES 2* para se obter um rápido processamento visual, possui mais de 20 *Widgets*(componentes de interface) para uso no desenvolvimento e aceita até 5 toques/cliques simultâneos na tela, oferecendo assim, diversas formas de uso como jogos e instrumentos musicais(*KIVY*, 2016).

```
from kivy.app import App
from kivy.uix.button import Button

class TestApp(App):
    def build(self):
        return Button(text='Hello World')

TestApp().run()
```

Figura 2 – Exemplo de uso do Kivy
Fonte: Site oficial Kivy.

A figura 2 apresenta um exemplo de uso da biblioteca *Kivy*. São feitos os *imports* necessários com o uso do comando *from*, cria-se a classe *TestApp* com um método construtor responsável por criar um botão assim que a mesma for utilizada e por fim, é criado uma instância(objeto) da classe e a aplicação(figura 3) é iniciada, através do método *run*. A figura 3 apresenta a interface da aplicação, gerada pelo codigo acima:

Figura 3 – Exemplo de interface desenvolvida com Kivy.
Fonte: Site oficial Kivy.

3 MATERIAIS E MÉTODOS

Nesta sessão estão descritos os métodos e materiais utilizados para o desenvolvimento da aplicação desenvolvida para ilustrar a integração das tecnologias mostradas anteriormente. Esse exemplo consiste de um sistema que pode ser executado nas plataformas citadas anteriormente a

fim de demonstrar uma interface desenvolvida com a biblioteca Kivy do Python. A aplicação permite que o usuário cadastre ideias da forma como preferir para que sejam armazenadas e consultadas quando necessário.

A metodologia utilizada para o desenvolvimento do trabalho envolveu primeiramente uma revisão bibliográfica das linguagens e das tecnologias aplicadas no desenvolvimento do artigo. Observa-se que para o sucesso do trabalho é uma etapa importante, pois segundo Silva e Menezes (2005, p. 30), "a revisão de literatura é fundamental, porque fornecerá elementos para evitar a duplicação de pesquisas sobre o mesmo enfoque do tema, e favorecerá a definição de contornos mais precisos do problema a ser estudado".

O estudo das tecnologias e das linguagens foi de natureza aplicada e com objetivo exploratório. A pesquisa aplicada "objetiva gerar conhecimentos para aplicação prática dirigidos à solução de problemas específicos", e a pesquisa exploratória "visa proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou a construir hipóteses, envolve levantamento bibliográfico"(Silva e Menezes (2005, p. 20)).

4 DESENVOLVIMENTO

O projeto foi desenvolvido seguindo o padrão MVC[12] sendo dividido em camadas que se comunicam entre si onde cada uma é responsável por uma tarefa dentro do projeto. A divisão por camadas tem como objetivo deixar o projeto da forma mais organizada possível e em caso de evolução ou alteração, é possível alterar ou substituir a camada específica sem comprometer o resto da aplicação.

A figura 4, demonstra a interface de cadastro da aplicação desenvolvida:

[12] MVC: Padrão de desenvolvimento de *software*

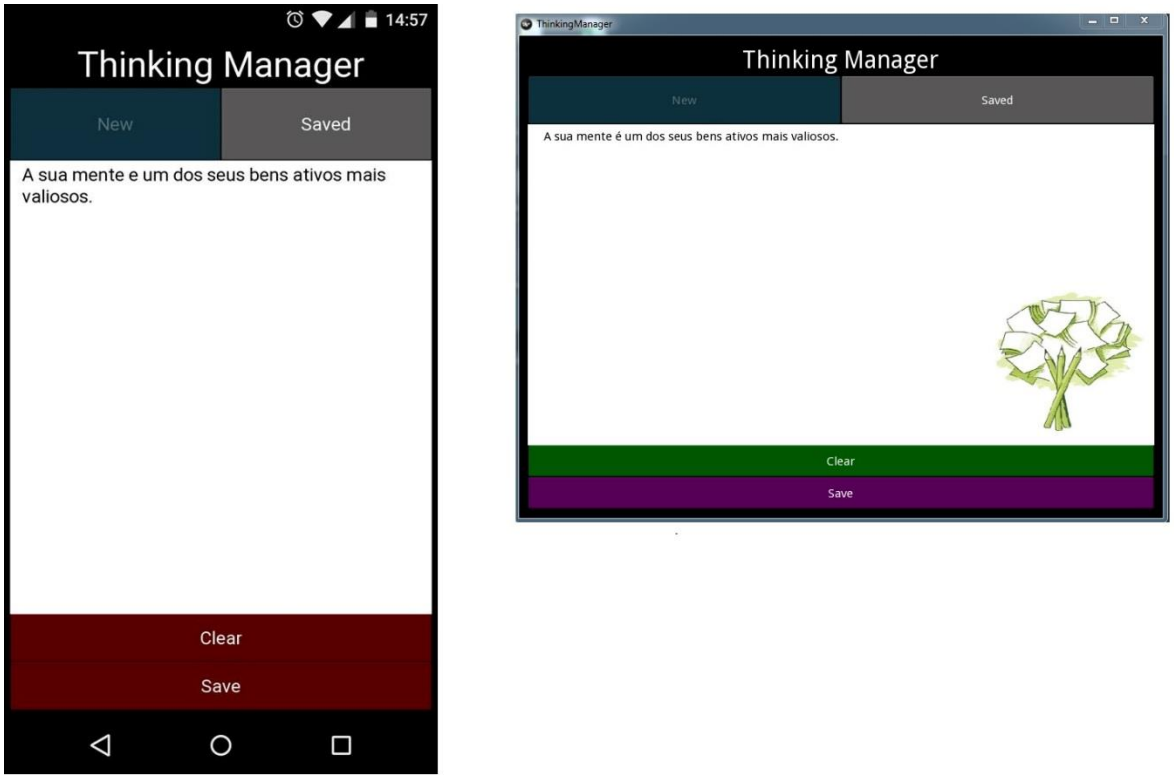

Figura 4: Aplicação desenvolvida sendo executada no *Android*[13](esquerda) e no *Windows*[14](direita).

4.1 Projeto e Análise

O projeto foi desenvolvido em etapas a fim de organizar o desenvolvimento da aplicação, sendo elas:

- Definição;
- Estudo de viabilidade;
- Pesquisa;
- Seleção das técnologias;
- Implementação/Construção;

Para a demonstração das tecnologias propostas, foram levantados alguns requisitos funcionais como:

[13] *Android*: Sistema operacional *mobile* da Google
[14] *Windows*: Sistema operacional da *Microsoft*

- O sistema deve permitir que o usuário cadastre um ideia;
- O sistema deve apresentar um texto instrucional para orientar o usuário sobre o cadastro de ideias;
- O sistema deve apresentar uma mensagem de sucesso ou erro após a tentativa de cadastro;
- O sistema deve permitir consulta com o conteúdo preenchido pelo usuário;
- O sistema deve permitir editar uma ideia cadastrada;
- O sistema deve permitir excluir uma ideia cadastrada;
- O sistema deve permitir a estilização do texto para consulta.
- O sistema deve adaptar sua estrutura às variadas resoluções de tela, para que seja de fácil utilização em dispositivos como *smartphones* e *tablet*s.

4.2 Desenvolvimento em camadas

A aplicação foi desenvolvida em camadas. Essa divisão proporciona ao *software* um alto grau de flexibilidade. Cada camada é formada por um conjunto de classes que possuem determinado propósito. Este conjunto preocupa-se com detalhes específicos que serão ocultos nas demais camadas.

Essa arquitetura facilita a manutenção no código e o reaproveitamento, mantendo uma codificação limpa e promovendo versatilidade para a adição de novas funcionalidades.

4.2.1 Camada *Screen*[15]

A camada *Screen* do projeto, é responsável por toda a *interface* de aprensentação da aplicação. Nela é definido o *layout* e a forma como os *Widgets* [16]se comunicam entre si.

O quadro 2 apresenta os *widgets* utilizados para o desenvolvimento:

[15] *Screen*: Camada da aplicação
[16] *Widget*: Componente de interface da biblioteca *Kivy*

Nome	Descrição
Label	O *widget Label* é utilizado simplemente para renderizar texto.
Button	No *Kivy*, o *widget Button* é uma *Label* associado a uma série de ações que são disparadas quando pressionado.
ToggleButton	Assim como *Button*, o *widget ToggleButton* possui uma série de ações que ao ser pressionado são disparadas, porém, permanece nos estados 'normal' e '*down*'.
Popup	O *widget Popup* é utilizado basicamente para criar pop-ups na aplicação. Por padrão, este irá sobrepor a janela "pai" da aplicação. Para uso, deve-se informar o título e o conteúdo a ser apresentado.
TextInput	*TextInput* fornece uma caixa de texto editável para o preenchimento de informações.
CheckBox	*CheckBox* é um botão de dois estados: "*checked*" e *"unchecked"*. Utilizado para determinar opções dentro da aplicação.
Carousel	O *widget Carousel* fornece um classico visualizador em carrosel onde é possível deslizar entre os *slides*. Neste componente é possível adicionar qualquer conteúdo e vê-lo ser movimentado nas forma horizontal ou vertical.
BoxLayout	*BoxLayout* é um dos layouts oferecidos pelo *Kivy*. Ele organiza os componentes em caixas podendo ser disposto de forma horizontal ou vertical.
GridLayout	O *widget GridLayout* organiza os componentes em uma matriz, calculando o espaço disponível na tela do aparelho utilizado. Dessa forma, o conteúdo é aprensentado em linhas e colunas.

Quadro 2: *Widgets* utilizados pela aplicação

Com os *widgets* apresentados acima, foi possível demonstrar o uso da biblioteca Kivy do Python através da aplicação desenvolvida.

A figura 5, apresenta um trecho do código encontrado na camada Screen:

```
def newScreen(self, backup_idea = None):

    self.last_screen = 'newScreen'

    '''
    Header
    '''
    lbl_title_app = Label(text = '[ref=tk]Thinking Manager[/ref]',
                          markup = True,
                          font_size = '30sp',
                          size_hint_y = None,
                          height = '40dp')
    lbl_title_app.bind(on_ref_press = partial(self.buildConfig, self.buttons_layout, self.layout, self.move_carousel))

    '''
    Menu
    '''
    btn_new = ToggleButton(text = 'New', state = 'down')
    btn_new.disabled = True

    btn_saved = ToggleButton(text = 'Saved'
                             , state = 'normal')
    btn_saved.bind(on_press = partial(self.buildSaved, self.buttons_layout, self.layout))

    self.buttons_layout.add_widget(btn_new)
    self.buttons_layout.add_widget(btn_saved)

    '''
    Content
    '''
    txt_idea = TextInput(multiline = True
                         , text_size = 50
                         , padding_x = 20
                         , background_normal = 'Capture.JPG'
                         , background_active = 'Capture.JPG')

    if backup_idea != None:
        txt_idea.text = backup_idea
        btn_save_name = 'Overwrite'
        editing = 1
    else:
        txt_idea.text = self.instructions #default idea
        txt_idea.bind(focus = partial(self.txtIdeaFocus, txt_idea))
        btn_save_name = 'Save'
        editing = 0
```

Figura 5: Trecho de código da camada *Screen* do projeto.

A figura 5 mostra um trecho do método *newScreen*, responsável por produzir o layout da tela de cadastro de ideias. Nela é possível ver alguns *widgets* e suas configurações sendo definidas.

Na parte supeior da aplicação, foi definido uma *Label* com o nome da aplicação, "*Thinking Manager*" que quando clicado, abre a tela de configurações. Logo após são definidos dois componentes *ToggleButton* responsáveis pela navegação entre o layout de cadastro e consulta. A seguir, é criada uma caixa de texto com o uso do *widget TextInput* para o preenchimento das ideias e então os botões de ação *Clear* para limpar o texto digitado e *Save* que cadastra o conteúdo da caixa de texto.

4.2.2 Camada *Controller*[17]

A camada *Controller*, é destinada a realizar operações de gerenciamento da aplicação, através de requisições feitas pelo usuário. Algumas delas são: a transferência das informações preenchidas pelo usuário para o banco, atualização das configurações definidas, popular o componente *Carousel* para consulta das ideias.

A figura 6 apresenta um trecho do código contido nessa camada:

```
#Constructor
def __init__(self):
    connection = connectDB()
    if not os.path.exists('DAO/database.db'):
        connection.createDB()

#Insert new idea
def insertIdea(self, idea, editing = 0, idea_id = None):

    if idea != '':
        try:
            connection = connectDB()
            connection.text_factory = str

            if editing == 0:  # insert
                sql = '''INSERT INTO IDEIA(idea_txt) VALUES(:bind)'''
            else:  # update
                sql = '''UPDATE IDEIA SET idea_txt = (:bind) WHERE idea_id = %s ''' % (idea_id)

            if connection.executeSQL(sql, idea):
                print('Ideia cadastrada/atualizada com sucesso!')
                return True

        except Exception as err:
            #self.errorSong()
            print('<!Controller!>Erro:' + str(err))

    else:
        #self.errorSong()
        return False
```

Figura 6: Trecho de código da camada Controller do projeto.

Na figura 6, está presente o método construtor *__init__*. Ele cria o objeto *connection* a partir da classe *connectDB* e verifica se existe um banco de dados criado no diretório '/DAO' e caso não seja encontrado, o método *createDB* é executado e o banco de dados será criado.

[17] *Controller*: Camada da aplicação.

Outro método que pode ser visto na figura é o *insertIdea*, responsável por promover a camada DAO um comando SQL e o conteúdo do *widget TextInput*, após uma requisição de *Save*(cadastro) ou *Overwrite*(atualizar) gerada pelo usuário. Seu fluxo, parte da validação das informações obtidas pelos parâmetros. O parâmetro *idea* que deve possuir a ideia preenchida na caixa de texto, passa por uma validação que verifica se o seu conteúdo é direfente de vazio. Logo após o valor do parâmetro *editing* é verificado, sendo 0(zero) para cadastro e 1(um) para atualização. Desse forma, é montado o comando SQL preciso para a ação que o usuário deseja, que segue para ser executado.

4.2.3 Camada DAO

A camada DAO(*Data Access Object*) tem como objetivo abstrair e encapsular os mecanismos de acesso ao banco de dados. Sendo assim, esta camada é responsável por todas as operações CRUD(*Create Read Update Delete*), ou seja, fatores que não interessam as demais camadas da aplicação.

Os métodos dessa camada são acessados pelo *Controller* através de requisições feitas pelo usuário. Alguns deles são: *createDB, executeSQL* e *getData.*

A figura 7 apresenta alguns métodos implementados na camada DAO:

```
'''
  EXECUTE SQL
'''
def executeSQL(self, sql, value = ''):

    try:
        conn = sqlite3.connect('DAO/database.db')

        if value != '':
            conn.text_factory = str
            conn.execute(sql, {'bind' : str(value)})
        else:
            return conn.execute(sql)

    except Exception as err:
        print('<!>Erro: ' + str(err))
        return False

    conn.commit()
    return True

'''
  Get Ideas
'''
def getData(self, sql_id, sql_txt):

    list_id = []
    list_idea = []
    matriz = []

    try:
        for row in self.executeSQL(sql_id):
            list_id.append(row[0])

        for row in self.executeSQL(sql_txt):
            list_idea.append(row[0])

        matriz = [list_id, list_idea]

    except Exception as err:
        print('<!>Erro: ' + str(err))

    return matriz
```

Figura 7: Trecho de código da camada DAO do projeto.

O método *executeSQL* é tem como objetivo executar todas operações CRUD no banco de dados. Para isso, ele possui dois parâmetros que são: *sql*, onde será informado o comando SQL a ser executado no banco e o *value*, que armazena os valores do comando SQL como por exemplo a ideia preenchido pelo usuário ou o id da ideia a ser editada.

Outro método apresentado pela figura 7 é o *getData*. Esse por sua vez, popula uma matriz com todos os IDs e ideias cadastradas no banco para preencher o *widget Carousel* e assim possibilitar a consulta das mesmas.

5 KIVY – VANTAGENS E DESVANTAGENS

Dentre as vantagens do Kivy, a maior delas foi permitir o uso do Python para o desenvolvimento *Mobile* de forma que possa existir interação com o banco de dados e controladores gráficos oferecendo diversos a caminhos serem explorados (CADERNODELABORATORIO, 2016).

Kivy também proporcionou a possibilidade de desenvolvimento multiplataforma para quem não conhece outras linguagens ou não deseja aprender.

A seguir, algumas das vantagens da biblioteca *Kivy*:

- O aplicativo pode ser executado no próprio *Desktop*, por isso, não há necessidade de instalar alguns emuladores ou maquinas virtuais para vê-lo funcionar;
- Comunidade ativa disposta a ajudar nos fóruns relacionados a *Kivy*;
- Relativamente simples de implantar com Buildozer(ferramenta utilizada para criar um executável para smarthphones) sem qualquer necessidade de mergulhar muito a fundo nos detalhes de determinada plataforma;
- Possibilita o desenvolvimento de aplicações *Mobile* multiplataforma na linguagem *Python*;

Por outro lado, *Kivy* também possui algumas desvantagens:

- Documentação confusa comparado a demais outras bibliotecas;
- Pouco conteúdo na Internet devido a tecnologia ainda ser recente;
- Em smarthphones, aplicativos desenvolvidos com *Kivy* apresentam um tempo maior que outros para iniciar a execução;

6 CONCLUSÃO

Kivy é uma biblioteca desenvolvida em C para programar em *Python*, focada em aplicações modernas. É simples, veloz, multiplataforma e aceita até cinco cliques/toques simultâneos. Com essas características, *Kivy* vem se mostrando uma tecnologia pensada no desenvolvimento de aplicações inovadoras com *Python*.

Ainda que recente, *Kivy* não para de crescer e já possui qualidades para andar em paralelo com as grandes ferramentas utilizadas atualmente para o desenvolvimento de aplicações multiplataforma.

Neste sentido, este trabalho contribuiu em apresentar a biblioteca *Kivy* por meio de uma aplicação desenvolvida citando suas principais vantagens, desvantagens e demontrando sua usabilidade a fim promover seu uso e disseminação no desenvolvimento multiplataforma.

REFERÊNCIAS BIBLIOGRAFICAS

BlOG DESENVOLVEDOR, **SQLite vs MySql vs PostgreeSQL**. Disponível no site: http://blog.desenvolvedor.org/sqlite-vs-mysql-vs-postgresql. Acesso em: 13 set, 2016.

CADERNODELABOLATORIO**, Kivy, SQLite, ORM e Sqlalchemy na programação multiplataforma.** Disponível no site: http://cadernodelaboratorio.com.br/2016/01/09/kivy-sqlite-orm-e-sqlalchemy-na-programacao-multiplataforma-i/. Acesso em: 20 out, 2016.

CHAPADEV, **My First Experience With Kivy**. Disponível no site: http://cheparev.com/my-experience-with-kivy/. Acesso em: 20 out, 2016.

CUONG, Do, **Quotes About Python**, 2007.

EMPORIO, **As 10 melhores aplicações multiplataforma**. Disponível no site: http://blog.emporio-web.com/lang/pt-pt/2010/05/as-10-melhores-aplicacoes-multiplataforma/. Acesso em: 19 jun, 2016.

FALANDODEGESTAO, **4 Motivos Para Desenvolver o Hábito de Anotar**. Disponível no site: https://falandodegestao.com/2011/12/18/4-motivos-para-desenvolver-o-habito-de-anotar/. Acesso em: 17 jul, 2016.

GUIDO, Van Rossum, **A história do Python**. Holanda, 2014.

KIVY, Disponível em http://kivy.org. Acesso em: 10 jan. 2016.

MINDBENDING, **A história do Python**. Disponível no site: http://mindbending.org/pt/a-historia-do-python. Acesso em: 12 set, 2016.

NORVIG, Peter, **Quotes About Python**, 2007.

Python, **Quotes About Python**. Disponível no site: https://www.python.org/about/quotes/. Acesso em: 12 set, 2016.

REVISTABW, **Conceitos Iniciais e Acesso Via Linha de Comando em SQLite**, Disponível no site: http://www.revistabw.com.br/revistabw/conceitos-iniciais-e-acesso-via-linha-de-comando-em-sqlite/. Acesso em: 13 set, 2016.

SILVA, E. L.; MENEZES, E. M. **Metodologia da Pesquisa e Elaboração de Dissertação**. 4.ed. Florianópolis: UFSC, 2005. Disponível em: https://projetos.inf.ufsc.br/arquivos/Metodologia_de_pesquisa_e_elaboracao_de_teses_e_dissertacoes_4ed.pdf. Acesso em: 15 jul, 2016.

SQLITE, **About SQLite**, Disponível no site: https://www.sqlite.org/about.html. Acesso em: 13 set, 2016.