

Apresentação do Web APP “Próximo”

APLICATIVO PARA AGILIZAR PEDIDOS DE REFEIÇÕES

Gabriel Brito Bastos

¹Departamento de Engenharia Eletrônica – Universidade Federal do Rio de Janeiro
(UFRJ)

gabriel.bastos@poli.ufrj.br

Resumo. *O trabalho se trata de um web app direcionado a agilizar a o pedido e a preparação de comidas para os restaurantes do CT (Centro de Tecnologia). Onde, através de uma página web, o cliente poderá visualizar o cardápio de diversos restaurantes e fazer seu pedido via internet, com intuito de menores filas e de um almoço mais rápido.*

1. Motivação

É notável a quantidade de problemas existentes no dia-a-dia de um brasileiro. Tais como filas, burocracia, ineficiência de estocagem, entre outros. Com a intenção de amenizar um problema cotidiano, resolvemos trabalhar em um aplicativo visando a agilidade no processo de compra de refeições, utilizando orientação a objetos em Python em conjunto com um banco de dados criado no MySQL.

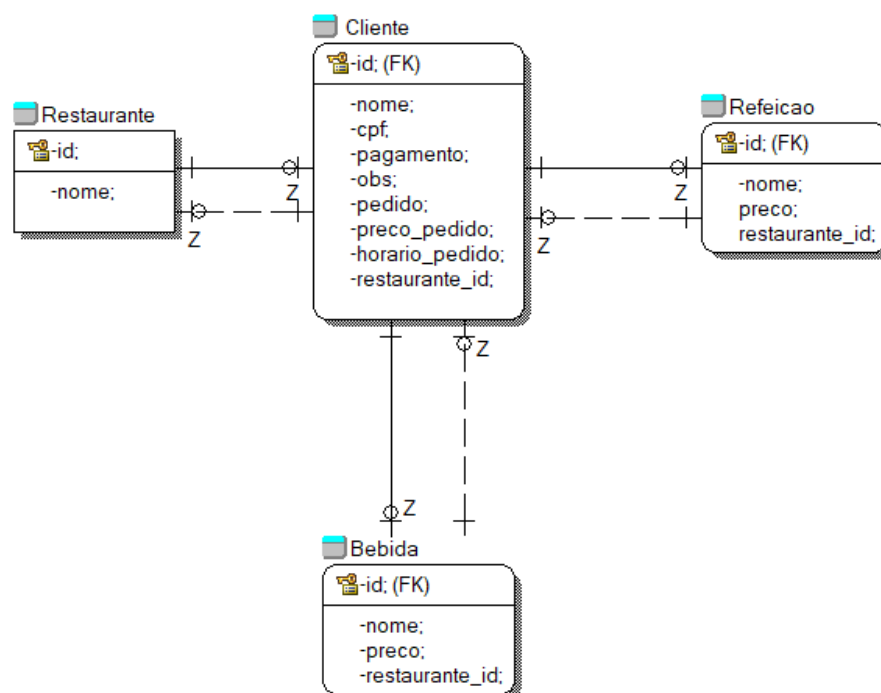
2. Objetivo

Tendo em vista as extensas filas que nos cercam recorrentemente, propusemo-nos a criar um aplicativo, chamado Próximo, para reduzir este problema. Este consiste basicamente em um processo de autoatendimento para restaurantes e praças de alimentação. Onde o cliente, ao chegar no local desejado, realiza seu pedido pelo celular. Sem precisar se locomover de sua mesa. O restaurante receberá esta informação, produzirá a refeição, e anunciará ao cliente quando a mesma estiver disponível para retirada. O aplicativo, portanto, traz conforto ao cliente e ao mesmo tempo reduz gastos com funcionários para uma empresa.

3. Modelo de Entidade e Relacionamento – Classes UML

O modelo abaixo ajuda a entender melhor como todo o programa vai funcionar e como as classes se intercomunicam e se relacionam.

O modelo UML de classes se assemelha muito ao modelo que fizemos para entidade relacionamento, segue o diagrama:



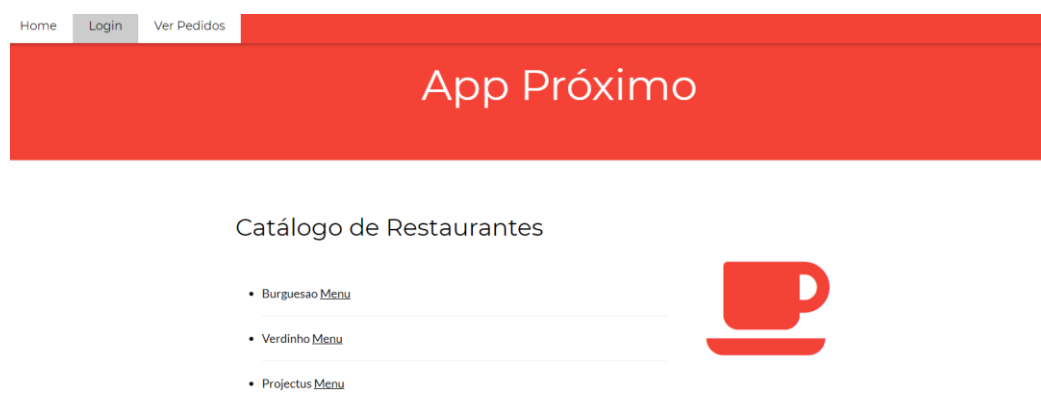
4. Código fonte

No código fonte, existe a área de templates (arquivos em HTML) e a área onde estão os arquivos Python. O template utilizado no app foi retirado de um site aberto, e o link para o mesmo se encontra na bibliografia deste documento.

A seguir, uma descrição das abas principais do web app:

Na página “Home”: Há uma tabela de restaurantes que estão cadastrados no banco de dados, onde o usuário vai selecionar um dos restaurantes da lista. Logo depois, há uma tabela com as refeições cadastradas do restaurante que foi selecionado. A seguir, encontra-se a página de bebidas, onde também é chamada a tabela das bebidas do restaurante especificado.

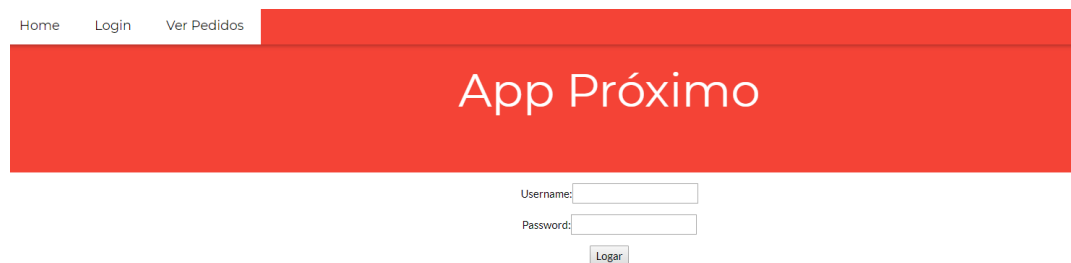
Finalmente, o usuário faz o cadastro de seus dados, como nome, CPF, forma de pagamento e um “Obs” para finalizar o pedido. Então, o cliente visualizará o pedido e confirmará a ação.



Na aba “Ver pedidos”: Nessa aba, acessamos a página onde é possível selecionar um restaurante e ver a lista de pedidos na ordem, para consultar quantos pedidos há na sua frente ou apenas ver a lista de pedidos do restaurante selecionado.



Na aba “Login” : Essa aba foi feita para os administradores. Onde os mesmos podem acrescentar novos restaurantes, pratos e bebidas no banco de dados, e assim ter uma inserção de dados mais fácil rápida do sistema. Nesta aba, também é possível editar e deletar restaurantes do banco de dados. Assim como editar e deletar bebidas e refeições. Isto foi acrescentado com objetivo de facilitar a inserção e edição de dados do banco.



Home Login Ver Pedidos

App Próximo

Username:

Password:

Logar

A seguir, uma descrição dos arquivos .py

- **Flask_app:**
O Flask é onde ocorre a interseção entre os dados e o Framework. Nele temos o flask recebendo o banco de dados que será utilizado para realizar as consultas. Primeiro, há de ser feita uma associação com o banco que será utilizado como fonte de dados. Depois, utiliza-se os decoradores para associar as funções às rotas postas no decorador. Para, então, o mesmo chamar os templates desejados e mandar as informações corretas do banco de dados para seus respectivos templates. Os decoradores, por sua maioria, são para direcionar para páginas da leitura do banco de dados, mas a página após o pedido ser realizado e confirmado, terá todos os dados que foram preenchidos pelo usuário sendo cadastrados na tabela “cliente”, para que possa ser visualizado na página “Ver pedidos”.
- **Classes.py:**
É o arquivo que contém as classes que vão ser chamadas no ‘flask_app’, onde cada classe lê suas respectivas tabelas e associa seus valores a strings para serem utilizadas no ‘flask_app’.
- **Lojas.py:**
Esse arquivo faz a inserção dos dados necessários dentro da tabela, para que eles sejam utilizados no ‘classes.py’ e consequentemente no ‘flask_app.py’. O arquivo é utilizado para, manualmente, preencher o banco de dados com as informações desejadas. Com as features de adicionar, remover e editar bebidas e restaurantes,

este arquivo se mostra útil apenas em caso de que as outras funções não estejam funcionando corretamente.

5. Extra – Parte 2

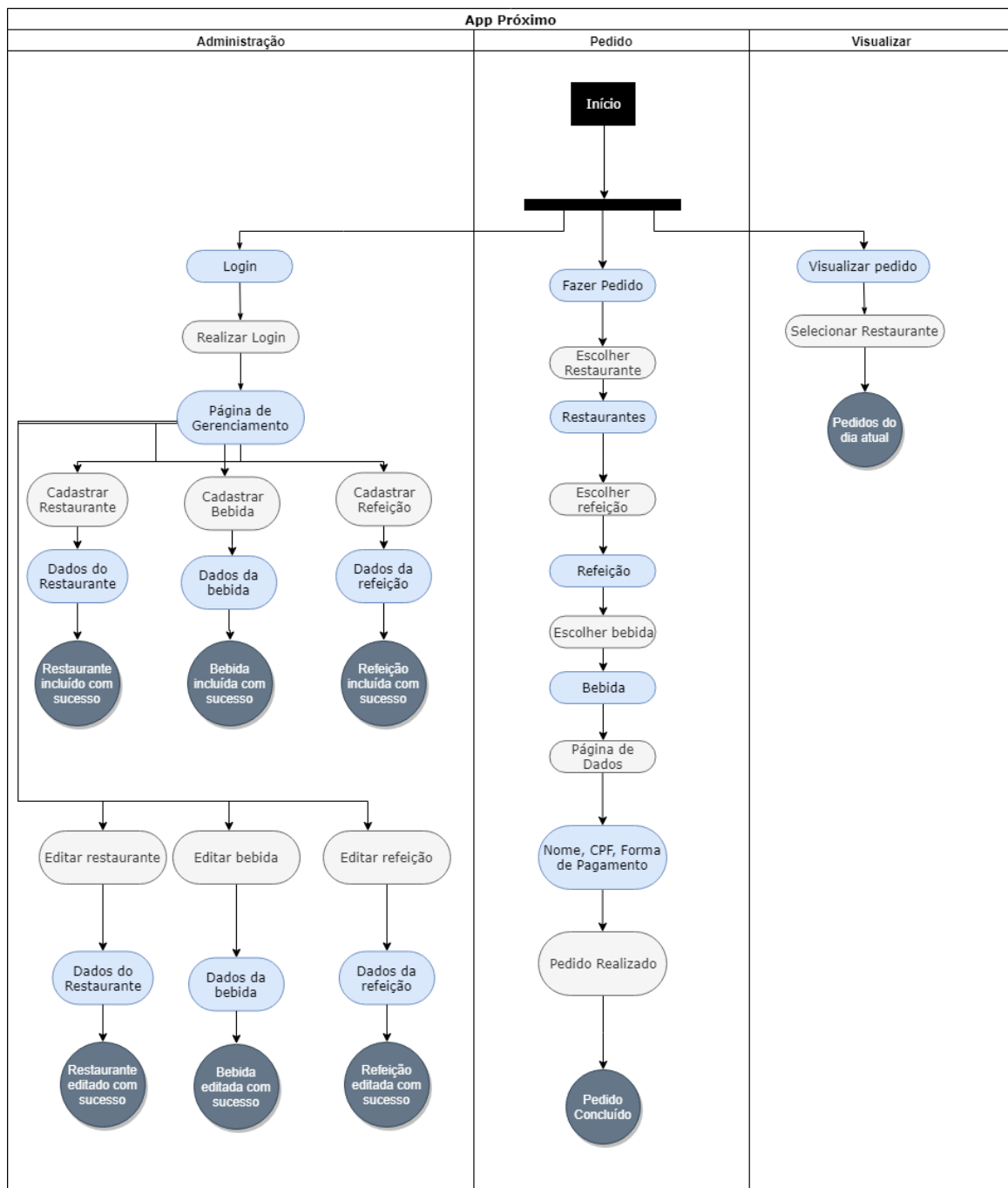
Existe também uma função útil para administradores. Onde, utilizando o Task do PythonAnywhere, diariamente serão salvos os pedidos registrados no banco de dados em um arquivo xml. Assim, administradores poderão analisar os pedidos cotidianamente. Esta feature foi feita utilizando a linha de comando mysqldump, como visto a seguir:

```
“mysqldump -u USUARIO -h LINK-DO-BANCO-DE-DADOS no-create-info --no-create-db --no-tablespaces --no-set-names --no-autocommit --skip-set-charset --xml ‘NOME DA BASE DE DADOS’ TABELA> DIRETORIO DO ARQUIVO.XML”
```

Foi feito também na segunda parte do projeto os diagramas UML, segue o diagrama de atividades em anexo.

Além de termos incluído também na parte de gerenciamento do aplicativo a parte de editar e remover restaurante e bebidas, ou seja, ainda identificamos algumas partes que podem melhorar a fim de tornar útil a aplicação e estaremos evoluindo o projeto para que seja uma plataforma bem finalizada e funcional para os usuários.

O PythonAnyWhere ajudou muito na programação do app, visto que a hospedagem e o suporte ao Flask e ao banco de dados eram viabilidades da ferramenta. Porém encontramos algumas dificuldades por nosso plano ser o mais básico, o plano grátis, como por exemplo, conexões de bancos de dados, não podíamos exceder três conexões ao banco, caso contrário encontramos o erro interno 500.



UML- Diagrama de Atividades

6. Bibliografia

- BORGES, Luiz Eduardo. **Python para Desenvolvedores**. 3 ed. São Paulo. Novatec. 2014
- POLEPEDDI, Lalith. **Uma Introdução a Python Flask Framework**. Envato Tuts +. Disponível em: <https://tutsplus.com>. Acesso em: 21/10/2018
- **Template**
Disponível em: https://www.w3schools.com/w3css/tryw3css_templates_start_page.htm