



MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DA AMAZÔNIA
INSTITUTO CIBER ESPACIAL
CURSO BACHARELADO EM SISTEMAS DE INFORMAÇÃO

NATANAEL SILVA CARDOSO
THAMIRYS MARTHA DA SILVA BISPO

**UM ESTUDO COMPARATIVO ENTRE OS PRINCIPAIS FRAMEWORKS DE
DESENVOLVIMENTO WEB EM LINGUAGEM PYTHON**

BELÉM
2019



MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DA AMAZÔNIA
INSTITUTO CIBER ESPACIAL
CURSO BACHARELADO EM SISTEMAS DE INFORMAÇÃO

NATANAEL SILVA CARDOSO
THAMIRYS MARTHA DA SILVA BISPO

**UM ESTUDO COMPARATIVO ENTRE OS PRINCIPAIS FRAMEWORKS DE
DESENVOLVIMENTO WEB EM LINGUAGEM PYTHON**

Trabalho de Conclusão de Curso apresentado
ao curso de Sistemas de Informação como
requisito para obtenção do grau de Bacharel
em Sistemas de Informação.

Orientador: Prof. Dr. João Ferreira de Santana
Filho

BELÉM
2019

NATANAEL SILVA CARDOSO
THAMIRYS MARTHA DA SILVA BISPO

**UM ESTUDO COMPARATIVO ENTRE OS PRINCIPAIS FRAMEWORKS DE
DESENVOLVIMENTO WEB EM LINGUAGEM PYTHON**

Trabalho de Conclusão de Curso apresentado à Comissão de Trabalho de Conclusão de Curso
e Estágio Supervisionado (CTES) como requisito para obtenção do grau de Bacharel em
Sistemas de Informação.

Data da aprovação. Belém – PA: ____/____/____

Banca Examinadora

Prof. Dr. João Ferreira de Santana Filho - Orientador
Universidade Federal Rural da Amazônia

Prof. MSc. Alex de Jesus Zissou - Examinador
Universidade Federal Rural da Amazônia

Prof.^a MSc. Decíola Fernandes de Sousa - Examinadora
Universidade Federal Rural da Amazônia

BELÉM
2019

AGRADECIMENTOS

Natanael:

A Deus;
A minha esposa Josa Cardoso;
Aos meus pais;
Ao professor João Santana por ter nos orientado;
Aos meus professores de todas as etapas até aqui e
Aos meus colegas de classe.

Thamirys:

A Deus;
A minha Mãe Marta Bispo;
Aos professores do Curso de Sistemas de Informação da UFRA;
Ao professor João Santana por ter nos orientado;
Aos meus colegas de classe.

RESUMO

O comparativo realizado nesta pesquisa tem por objetivo analisar qual a carga mínima de trabalho necessária para desenvolver uma aplicação WEB com Python através do framework Django e do microframework Flask. Aplica os recursos estudados sobre a linguagem de programação Python a fim de identificar um roteiro lógico para escolha de ferramentas utilizadas para construção de aplicações WEB nas disciplinas práticas da faculdade de sistemas da UFRA. Explora os parâmetros de prazo, qualidade e desempenho para analisar qual a melhor opção neste contexto. Aborda os principais conceitos relacionados a linguagem de programação python, a programação Web, ao fluxo de requisições na Web e aos frameworks. Descreve as principais características do Django e do Flask e o passo a passo para implementar os requisitos mínimos determinados na fase de planejamento. Conclui que, contexto universitário, Flask é a melhor ferramenta para desenvolver aplicações funcionais nas disciplinas práticas do curso.

Palavras-chave: Django. Flask. Sistemas WEB. Frameworks. Python.

ABSTRACT

The comparative post in the research analysis in the minimum analysis of work required for the development application WEB with Python the Django framework and the Flask microframework. Applies to a set of data on the programming language for a programming language for the choice of tools for the construction of WEB applications in the practical disciplines of the faculty of systems of UFRA. It explores the parameters of term, quality and performance to analyze the best option in this context. It covers the main concepts related to the programming language, the Web, the flow of requests in the Web and the frameworks. Describe it as the main process factors and steps for planning. Concludes that, in the university context, Flask is the best tool to become more functional in the practical disciplines of the course.

Keywords: Django. Flask. WEB Systems. Frameworks. Python

LISTA DE ILUSTRAÇÕES

Figura 1 - Atributos de qualidade de software	13
Figura 2 - Popularidade de Python entre 2002 e 2018	19
Figura 3 - Códigos HTTP	23
Figura 4 - Popularidade dos frameworks Python.....	25
Figura 5 - biblioteca assíncrona do tornado	26
Figura 6 - Hello World em bottle	26
Figura 7 - Hello World em pyramid	27
Figura 8 - Fluxo de uma requisição em Django	28
Figura 9 - arquivos iniciais de um projeto em Django	30
Figura 10 - Interesse em Django e Flask em 2018	31
Figura 11 - página estática em Flask	32
Figura 12 - Classe ProdutoDao.....	34
Figura 13 - Diagrama de caso de uso	40
Figura 14 - Diagrama de classes dos protótipos	41
Figura 15 - Instalação do Django na máquina virtual.....	42
Figura 16 - configuração da CBV para página inicial	43
Figura 17 - Lista de produtos em Django	43
Figura 18 - Modelo da classe Produtos	44
Figura 19 - View de atualização de dados	45
Figura 20 - Sistema de URLs no Django	45
Figura 21 - Página inicial em Flask	46
Figura 22 - função salvar produto.....	47
Figura 23 - Tela de login	48
Figura 24 - rota para autenticar o usuário.....	49
Figura 25 - Listagem de produtos em Flask	49
Figura 26 - Template em laço na página inicial.....	50
Figura 27 - Tela de cadastro de produtos.....	51
Figura 28 - rota para atualizar produtos	51
Figura 29 - rota para atualizar produtos	52

LISTA DE ABREVIATURAS E SIGLAS

ABNT	ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS
CBV	CLASS-BASED-VIEW
CRUD	CREATE READ UPDATE DELETE
DAO	DATA ACCSESS OBJECT
HTML	HYPERTEXT MARKUP LANGUAGE
HTTP	HYPERTEXT TRANSFER PROTOCOL
IDE	INTEGRATED DEVELOPMENT ENVIRONMENT
IEEE	INSTITUTO DE ENGENHEIROS ELETRICISTAS E ELETRÔNICOS
IP	INTERNET PROTOCOL
MVT	MODEL-VIEW-TEMPLATE
NBR	NORMA BRASILEIRA REGULAMENTADORE
PPC	PROJETO PEDAGÓGICO DO CURSO
SGBD	SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS
SSL	SECURE SOCKETS LAYER
TI	TECNOLOGIA DA INFORMAÇÃO
TSL	TRANSPORT LAYER SECURITY
UFRA	UNIVERSIDADE FEDERAL RURAL DA AMAZÔNIA
UML	UNIFIED MODELING LANGUAGE
URL	UNIFORM RESOURCE LOCATOR
WSGI	WEB SERVER GATEWAY INTERFACE

SUMÁRIO

1 INTRODUÇÃO	8
1.1 JUSTIFICATIVA	9
1.2 OBJETIVOS DO TRABALHO	11
1.3 METODOLOGIA DE DESENVOLVIMENTO DO SOFTWARE	12
1.4 ORGANIZAÇÃO DO TRABALHO	16
2 REFERENCIAL TEÓRICO	18
2.1 PYTHON	18
2.2 PROGRAMAÇÃO WEB	21
2.2.1 Fluxo de requisições na WEB	22
2.3 FRAMEWORKS PYTHON	24
2.4 DJANGO	27
2.5 FLASK	30
2.5.1 Persistência	33
2.6 MODELAGEM DE SISTEMAS	34
2.7 UML	36
2.8 FERRAMENTAS UTILIZADAS	36
3 MODELAGEM E IMPLEMENTAÇÃO	39
3.1 REQUISITOS	39
3.2 MODELAGEM DO SISTEMA	40
3.3 IMPLEMENTAÇÃO DJANGO	41
3.4 IMPLEMENTAÇÃO EM FLASK	46
4 CONCLUSÕES	53
4.1 SOBRE O USO DOS FRAMEWORKS	53
4.2 SOBRE AS FERRAMENTAS DE DESENVOLVIMENTO	60
4.3 LIMITAÇÕES DO TRABALHO	61
4.4 TRABALHOS FUTUROS	61
REFERÊNCIAS	63
APÊNDICE A – COMMIT DOS ARQUIVOS DO PROTÓTIPO NO GITHUB	66

1 INTRODUÇÃO

A inserção de recursos tecnológicos em áreas essenciais para o desenvolvimento humano (educação, economia, saúde, segurança...) gerou demanda por produtos e serviços na área de TI que podem representar oportunidades de atuação em diversos segmentos - independentemente da área de especialização dos profissionais que atuam nesse meio (programação, ciência de dados, engenharia de software, análise de sistemas, banco de dados...). Para tanto, é necessário verificar como atender essa demanda com as premissas que ela pressupõe (prazo, desempenho e qualidade).

Na área de programação, especificamente, o desenvolvimento de soluções para atender esta demanda é amparado por diversos recursos em um processo que envolve desde a escolha de uma linguagem de programação até a gerência de recursos externos como a integração com banco de dados através de um SGBD.

Esses recursos são essenciais para determinar a qualidade, o desempenho, e o tempo que envolvem o desenvolvimento de software e a complexidade gerada durante todo o processo. Na WEB eles podem ser navegadores com seus diferentes motores e ferramentas de desenvolvedor, sistemas operacionais, interações em redes de computadores, protocolos que estabelecem a comunicação entre clientes e servidores, a própria internet, servidores WEB, configurações de frameworks (MOLINARI, 2016) ou, ainda, ferramentas relacionadas a controle de versionamento e hospedagem que envolvem, também, uma série de outros recursos.

Nesse contexto, ainda que voltado para aplicações estáticas ou dinâmicas, direcionado para estudo, trabalho ou formação do conhecimento e com diferentes graus de utilização; o desenvolvimento de páginas para WEB é uma ação que está sempre revestida de significação complexa. Essa complexidade se dá por considerá-la, a programação WEB, não somente como um insumo, ou seja, não é simplesmente usada para produzir um bem ou serviço é, também, o próprio produto, visto que é consumível, revestido de valor, mutável e, no que se refere a sua forma, é variável, pois seu significado e sua aplicação mudam de acordo com o contexto em que está inserida ou com os objetivos do projeto em questão.

A problemática em torno desse cenário é o desafio para determinar qual a melhor ferramenta para utilizar nos projetos de desenvolvimento e que combinação de recursos equaliza a relação entre demanda e entrega de software. O desenvolvimento de software, nesse contexto, torna-se algo complexo, pois exige do aluno ou profissional de TI o conhecimento de vários mecanismos que possibilitam a construção de aplicações, tais como as linguagens de programação e suas estruturas de dados, servidores de aplicações, ambientes de desenvolvimento e outros.

Na faculdade de sistemas da UFRA essa problemática está intrinsecamente relacionada a dificuldade que os alunos possuem nas disciplinas relacionadas a programação, desde o início do curso (quando se estuda a parte lógica). Para os que continuam no curso essa dificuldade se transforma em baixo desempenho no decorrer dos semestres, pois afeta a capacidade de terminar um projeto cumprindo todos os requisitos que foram estipulados.

Sobre a UFRA é importante destacar que: criada em substituição à Faculdade de Ciências Agrárias do Pará – FCAP, a UFRA foi instituída pela Lei nº 10.611, DE 23/12/2002, e tem como missão formar profissionais de nível superior, desenvolver e compartilhar cultura técnico-científica através da pesquisa e extensão, oferecer serviços à comunidade e contribuir para o desenvolvimento econômico, social e ambiental da Amazônia. (PPC, 2011).

1.1 JUSTIFICATIVA

A tomada de decisão em projetos de desenvolvimento de software, na Universidade, está intrinsecamente ligada ao conhecimento compartilhado em sala de aula, principalmente quando o aluno não tem experiência na área em que estuda. Por isso, foi perceptível durante a realização do curso uma grande preocupação em associar os conceitos estudados as práticas que darão subsídios para atuação do aluno no mercado.

A prática no ambiente externo a Universidade está, obviamente, ligada a diversos fatores subjetivos (atividades complementares, por exemplo). No entanto, uma das funções da informação disseminada em sala de aula é combinar as diversas áreas do conhecimento que estão divididas em diferentes disciplinas de forma que se possa facilitar e fundamentar a

tomada de decisão no âmbito acadêmico e profissional. Entretanto, o caminho para concretizar essa função não é linear e tão pouco fácil.

Tendo isto em vista, a principal justificativa para realização desta pesquisa foi a percepção da dificuldade apresentada pelos alunos do curso de Sistemas de Informação da UFRA em determinar como definir que ferramentas utilizar para desenvolver softwares na parte prática de disciplinas como laboratório de software, banco de dados ou, ainda, como integrar os conhecimentos adquiridos nas disciplinas correlatas para gerar um produto funcional (um sistema WEB, no caso desta pesquisa).

Nesse sentido, verificou-se que a partir desse trabalho de conclusão de curso é possível, também, disponibilizar e organizar o conhecimento sobre dois importantes frameworks Python de forma que as informações apresentadas proporcionem o alcance do perfil do egresso expresso no PPC do curso que envolve:

- a) domínio de novas tecnologias da informação e gestão da área de Sistemas de Informação, visando melhores condições de trabalho e de vida;
- b) conhecimento e emprego de modelos associados ao uso das novas tecnologias da informação e ferramentas que representem o estado da arte na área;
- c) conhecimento e emprego de modelos associados ao diagnóstico, planejamento, implementação e avaliação de projetos de sistemas de informação aplicados nas organizações;

Mas para isto ocorrer efetivamente e para que o curso consiga cumprir sua missão é necessário utilizar mecanismos que estejam de acordo com a ementa desenvolvida no decorrer do curso, por isso, Python foi a linguagem escolhida para dar segmento a pesquisa, pois é uma das linguagens trabalhadas nos módulos iniciais do curso e oferece suporte para o desenvolvimento WEB através de vários frameworks dentre os quais o Django e Flask entre os microframeworks. Além disso, Python é uma linguagem de propósito geral o que a torna útil em várias disciplinas do curso (inteligência artificial, técnicas de programação, laboratório de software...).

Django e Flask foram escolhidos para o comparativo, porque são tecnologias utilizadas por desenvolvedores o que garante a manutenção de suas bibliotecas pela comunidade. Grandes empresas utilizam recursos de Python através deles (Youtube, Instagram, Spotify, Google e Redit, por exemplo).

Além disso, considerando, por exemplo, a necessidade do mercado por profissionais fullstack e de produzir mais com menos percebe-se que essas ferramentas se adequam tanto ao objetivo da Universidade quanto do mercado, pois disponibilizam uma série de bibliotecas e diferentes padrões que guiam o desenvolvedor do início ao fim de pequenos ou grandes projetos.

Com o estudo realizado nesta pesquisa o aluno poderá verificar o que ele precisa conhecer para determinar a melhor forma de aplicar os conhecimentos formados em sala de aula na prática de desenvolvimento de projetos com Python e apesar de ser a abordagem de uma linguagem específica com ferramentas específicas ele poderá utilizar a pesquisa como parâmetro para avaliar frameworks que possuem como base outras linguagens, pois os frameworks têm o mesmo objetivo geral e os processos são muito semelhantes.

1.2 OBJETIVOS DO TRABALHO

Tendo em vista que o foco desta pesquisa é verificar qual o conjunto de ferramentas mais adequado para construir uma aplicação funcional de acordo com os parâmetros que foram exigidos em disciplinas práticas como laboratório de software e banco de dados II o objetivo geral desta pesquisa é:

- analisar qual a carga mínima de trabalho necessária para desenvolver uma aplicação WEB com Python através dos framework Django e do microframework Flask.

Para alcançar o objetivo geral foram definidos os seguintes objetivos específicos:

- Criar a interface de uma aplicação WEB em Django e Flask;
- Implementar um protótipo de um sistema de gerenciamento de produtos em Django e Flask;
- Apresentar as principais diferenças entre Django e Flask;
- Comparar critérios de qualidade voltados para o desenvolvimento;
- Verificar entre estas ferramentas onde o processo de programação é mais rápido;
- Identificar em qual aplicação o processo de reuso é mais flexível.

Dessa forma com esta análise pretende-se entregar protótipos que demonstrem qual destas ferramentas de desenvolvimento WEB baseado em Python se adequa melhor a realidade que os alunos na faculdade de Sistemas da UFRA.

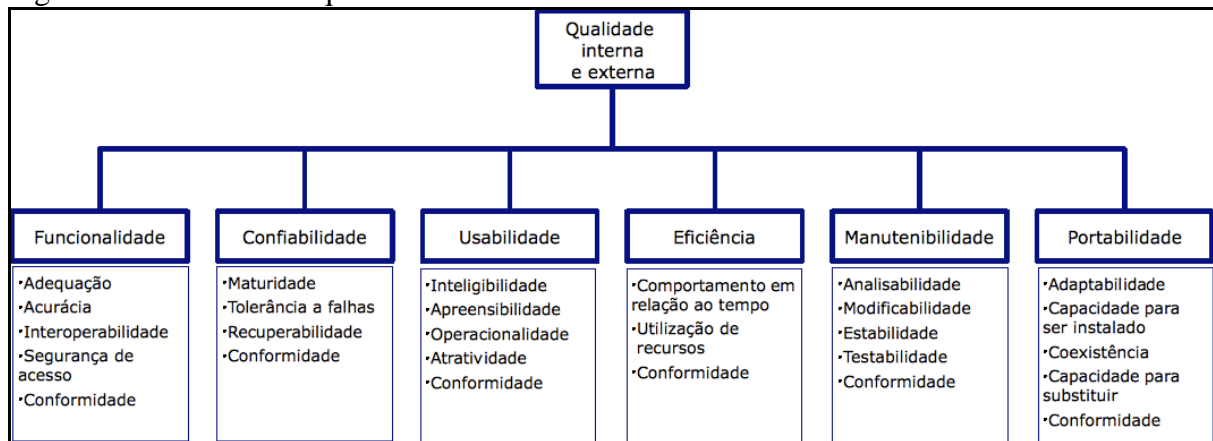
1.3 METODOLOGIA DE DESENVOLVIMENTO DO SOFTWARE

O projeto foi realizado através de pesquisas bibliográficas, da utilização de fóruns de desenvolvimento (stack overflow, fórum do alura e outros), controladores de versão e artigos em repositórios virtuais. Todos os levantamentos de dados foram direcionados para o atendimento das três características de um projeto de software: prazo, qualidade e desempenho.

Essas características foram adaptadas para o contexto da Universidade. Dessa forma, a ideia inicial foi utilizar o período de duração de uma disciplina prática (laboratório de software, nesse caso) para avaliar, por exemplo, a carga mínima de trabalho para desenvolver um módulo de gerenciamento de produtos de acordo com os requisitos que são normalmente exigidos na faculdade (interface e integração com banco de dados).

Já para avaliar o Django e o Flask com base nas características relacionadas à qualidade de software utilizou-se como parâmetro as especificações propostas pela NBR ISO/IEC 9126-1 cuja estrutura geral é mostrada na figura 1:

Figura 1 - Atributos de qualidade de software



Fonte: Google imagens (2018)

A funcionalidade é definida como a “capacidade do produto de software de prover funções que atendam às necessidades explícitas e implícitas, quando o software estiver sendo utilizado sob condições especificadas”. (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2003, p. 8). Nesse atributo foram marcados como “não se aplica” os critérios de interoperabilidade e conformidade.

A confiabilidade é a “Capacidade do produto de software de manter um nível de desempenho especificado, quando usado em condições especificadas” (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2003, p. 8). Nesse atributo foram marcados como “não se aplica” os critérios de interoperabilidade e conformidade. Nesse atributo foi marcado como “não se aplica” o critério de conformidade.

A usabilidade “Capacidade do produto de software de ser compreendido, aprendido, operado e atraente ao usuário, quando usado sob condições especificadas. ” (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2003, p. 9).

A eficiência “Capacidade do produto de software de apresentar desempenho apropriado, relativo à quantidade de recursos usados, sob condições especificadas. ” (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2003, p. 10). Nesse atributo foram marcados como “não se aplica” os critérios de interoperabilidade e conformidade. Nesse atributo foi marcado como “não se aplica” o critério de conformidade.

A manutenibilidade é a “Capacidade do produto de software de ser modificado. As modificações podem incluir correções, melhorias ou adaptações do software devido a

mudanças no ambiente e nos seus requisitos ou especificações funcionais. ” (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2003, p. 10).

A portabilidade é a “Capacidade do produto de software de ser transferido de um ambiente para outro ”. (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2003, p. 10). Nesse atributo foi marcado como “não se aplica” o critério de conformidade.

Para cada critério de qualidade foi atribuído um peso (0-3) tanto para o comportamento do Django quanto para o comportamento do Flask diante das características avaliadas. O peso 0 significa que o critério não se aplica, o peso 1 que não atende, o peso 2 que atende parcialmente, e o peso 3 que atende ao critério avaliado. No final foram somados os pesos para verificar qual das ferramentas atende melhor aos critérios definidos pela NBR de qualidade de software.

O somatório com maior resultado indica a ferramenta que atende melhor ao parâmetro de qualidade. Alguns critérios foram marcados como não se aplica, pois não podiam ser aplicados ao contexto da pesquisa. Os critérios relacionados a conformidade, por exemplo, não se aplicam, pois estão relacionados a regulamentação e legislação vigente direcionadas ao produto de software. Como o foco foi ambiente acadêmico esse critério não se aplica ao contexto da pesquisa.

Além disso, o prazo foi avaliado pelo cumprimento dos requisitos dentro do tempo estipulado no quadro 1 e o desempenho foi medido pelo número de tarefas desenvolvidas dentro do tempo estipulado inclusive as funções adicionais criadas a partir do tempo restante direcionado para o Flask ou o Django.

O projeto foi dividido em quatro etapas (resumidas no Quadro 1). Ao final de cada etapa foram realizadas as análises e revisões necessárias para avaliar os incrementos que foram feitos:

Quadro 1 - etapas de desenvolvimento e avaliação do software

Etapas	Ações desenvolvidas	Objetos de estudo	Tempo mínimo investido
Planejamento	Modelagem do sistema, análise dos requisitos,	Requisitos	68h
Desenvolvimento do sistema em Flask	Implementação das interfaces, integração com o banco de dados,	Microframework Flask	68h
Desenvolvimento do sistema em Django	Implementação das interfaces, integração com o banco de dados,	Framework Django	68h
Análise dos resultados	Análise comparativa do prazo, qualidade e desempenho dos sistemas desenvolvidos	Flask e Django	68h

Fonte: Os autores (2018)

O quadro 1 foi desenvolvido com base no tempo que é disponibilizado pela faculdade em cada disciplina (carga horária em sala de aula). Assim como ocorre na prática o tempo investido em cada etapa foi alterado conforme a necessidade, portanto o tempo apresentado no quadro 1 foi o tempo mínimo previsto para realização de cada atividade descrita.

Para evidenciar a relação entre a metodologia escolhida e os objetivos específicos definidos elaborou-se o quadro 2 conforme apresentado a seguir:

Quadro 2 – Metodologia específica para cada objetivo específico

OBJETIVO ESPECÍFICO	METODOLOGIA ESPECÍFICA
Criar a interface de uma aplicação WEB em Django e Flask	Utilizar recursos do bootstrap (versão 4) nas interfaces.
Implementar um protótipo de um sistema de gerenciamento de produtos em Django e Flask;	Utilizar a IDE pycharm para desenvolver o protótipo em Flask e o editor VS Code para Django.
Apresentar as principais diferenças entre Django e Flask	Fazer o comparativo com base no prazo, desempenho e qualidade durante a evolução dos protótipos desenvolvidos
Comparar critérios de qualidade voltados para o desenvolvimento	Utilizar os critérios da NBR
Verificar entre estas ferramentas onde o processo de programação é mais rápido	Métrica de tempo com base no cumprimento do prazo
Identificar em qual aplicação o processo de reuso é mais flexível	Comparar em que protótipo foi mais simples reutilizar recursos oferecidos por Django e Flask

Fonte: Os autores (2019)

1.4 ORGANIZAÇÃO DO TRABALHO

No primeiro capítulo o referencial teórico da pesquisa, todos os conceitos que foram utilizados para construir o projeto. Nesse tópico, mostra-se uma visão geral da estrutura da linguagem de programação Python e os recursos ela oferece para trabalhar com WEB, tais como seus frameworks pois elas influenciam a escolha das ferramentas utilizadas no desenvolvimento. Também foram mostradas as características gerais da programação WEB e do fluxo de requisições relacionados a ela. Essa abordagem foi feita, pois auxilia no entendimento do fluxo que ocorre nos frameworks. Em seguida foram apresentadas algumas

informações sobre os frameworks (com ênfase nas características dos que são baseados em Python).

As seções 2.4 e 2.5 foram direcionadas para Django e Flask, pois dentro da categoria analisada eles são os mais populares. Posteriormente foi feita a conceituação de modelagem de sistemas e UML e a apresentação geral das principais ferramenta utilizadas no trabalho (PyCharm, VSCode, Astah).

No terceiro tópico mostrou-se como o sistema foi modelado e como a implementação em Django e Flask foi executada. Nesse tópico descreveu-se os principais comandos e apresentadas as principais telas dos sistemas. Além disso, mostrou-se como os requisitos foram atendidos. Já no último tópico apresenta-se a análise dos resultados obtidos durante a pesquisa com base nos parâmetros estabelecidos para comparação entre os frameworks.

2 REFERENCIAL TEÓRICO

Uma das principais questões que surgem quando se decide desenvolver para WEB é a escolha da linguagem de programação. Ao verificar, por exemplo, o ranking de linguagens de programação disponibilizado pela IEEE Spectrum pode-se dizer que existem pelo menos 48 linguagens disponíveis no mercado. Para esta pesquisa a linguagem de programação utilizada foi Python dado que é uma das linguagens utilizadas nos módulos iniciais do curso de sistemas na UFRA.

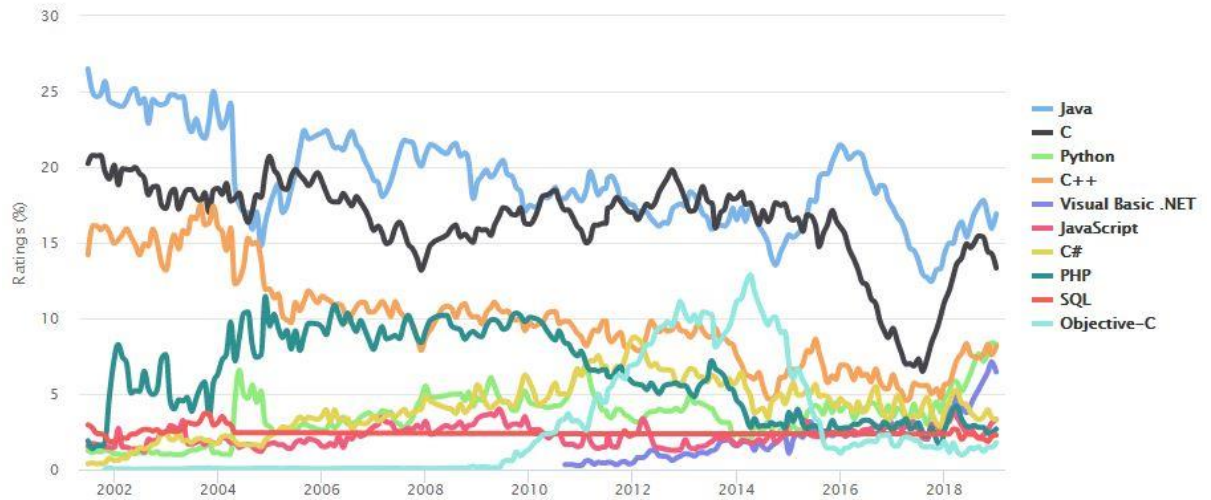
2.1 PYTHON

A linguagem de programação Python foi criada em 1990 por Guido Van Rossum, a partir da linguagem ABC (BORGES, 2010), mas foi publicada em 1991 (SOUSA, 2017) e apesar de, atualmente, essa linguagem ser utilizada em diversos tipos de aplicações, originalmente ela foi associada ao desenvolvimento de tarefas computacionais relativamente pequenas (SEBESTA, 2011).

Python é uma linguagem de programação de alto nível assim como Java, C++ e outras. Essa característica já deixa implícito que parte de sua função é abstrair a complexidade gerada por linguagens de máquina. Os programas em Python são executados através de um interpretador e por definição seus arquivos possuem a extensão “.py” (DOWNEY; ELKNER, MEYERS, 2009).

Por ser uma linguagem de propósito geral (aplicável na WEB, desktop, jogos...) ela vem se popularizando ao longo dos anos. Esse fato foi demonstrado em diversas pesquisas realizadas em 2018 onde ela aparece sempre nas primeiras posições. No ranking apresentado pela IEEE Spectrum, Python aparece como a linguagem mais popular em 2018 (CASS; BULUSU, 2018) já na pesquisa da TIOBE ela aparece em 3º lugar conforme mostra a figura 2:

Figura 2 - Popularidade de Python entre 2002 e 2018



Fonte: (TIOBE, 2018)

O gráfico da figura 2 deixa visível a evolução da popularidade de Python, em relação às demais linguagens, de 2002 até 2018. As características desta linguagem que justificam essa evolução são várias: sintaxe, suporte, tipagem, gerenciamento de memória e outras, conforme mostra o quadro 2:

Quadro 2 - principais características de Python

CARACTERÍSTICA	DESCRIÇÃO
Sintaxe	A sintaxe de Python é simples e o tamanho do código depende mais do programador do que da linguagem, mas normalmente
Suporte	A comunidade de Python é grande e ativa, portanto é fácil conseguir suporte
Tipagem	Possui tipagem dinâmica e não realiza conversão implícita (na versão atual)

Gerenciamento de memória	Python utiliza o Garbage Collector (coletor de lixo) para gerenciar a memória do programa. Dessa forma, o usuário não precisa fazer a remoção do objeto da memória manualmente
Paradigma	Python foi criada para ser uma linguagem orientada a objetos, mas é possível programar de maneira procedural ou imperativa
Strings	Strings em Python são imutáveis. Qualquer função de alteração devolve uma nova string, que representa a alteração. Esse princípio também é chamado de imutabilidade.
Estrutura de dados	O Python facilita a manipulação de estruturas de dados, pois tem diversas funções e operadores que auxiliam as tarefas mais comuns em trabalhar com Listas e Strings, nos tornando programadores muito produtivos.
Flexibilidade	Python possui diversas bibliotecas e a liberdade para desenvolver sem a preocupação de declaração de variáveis e tipos torna a linguagem bastante flexível

Fonte: os autores (2018) adaptado de Sousa (2017) e Steppat (2018)

Entretanto, observe-se que parte das características apresentadas no quadro podem ser encontradas em outras linguagens de programação. A questão principal é qual o contexto de aplicação de cada característica com determinada linguagem. Por exemplo, em comparação com as linguagens que aparecem nas primeiras posições com Python na figura (java e C), Seabra, Drummond e Gomes (2018) realizaram um estudo sobre os problemas clássicos de computação relacionados às linguagens de programação citadas. Primeiramente eles

concluem que a sobreposição de C, em determinado contexto, se deve principalmente por seu desempenho em tempo de execução e afirmam ainda que:

a linguagem Python, apesar de apresentar pior desempenho quando comparada às demais, seria uma boa escolha para implementações nas quais se necessita de um resultado quase que imediato. A escolha do Java se daria mais por conta de experiência e aptidão do desenvolvedor [...] as linguagens de mais alto nível tendem a perder em desempenho por conta da abstração. Ou seja, quando se é realizada a decisão de abstrair um processo em uma linguagem de programação, pode-se resultar na queda de performance computacional juntamente do ganho no tempo de produção.

Sob essa perspectiva entende-se que o campo de aplicação e os parâmetros utilizados determinam o desempenho que a linguagem terá sob determinado aspecto. Nesse sentido, pode-se dizer que se um indivíduo determina que vai utilizar uma linguagem x ou y em seu projeto ele precisa verificar se dentro do contexto em que está inserido essa linguagem pode ser eficaz.

Entretanto, se já existe uma familiaridade com uma linguagem específica e ela dá suporte a área de aplicação de interesse do indivíduo em questão (mesmo que não seja a linguagem que apresente o melhor desempenho na área) é interessante buscar ferramentas relacionadas a ela que maximizem seu desempenho e permitam que os objetivos do projeto sejam atendidos. Esse é caso do desenvolvimento WEB com Python, pois as ferramentas utilizadas para desenvolver para WEB nesta linguagem são relativamente recentes e atingem bons resultados, impulsionados pela comunidade, mas sob determinados aspectos uma linguagem ou outra (principalmente as específicas para WEB, como javascript ou PHP) podem apresentar algumas vantagens em relação a ela.

2.2 PROGRAMAÇÃO WEB

Desenvolver para WEB, em termos gerais, significa criar um conjunto de páginas HTML com recursos adicionais embutidos (folhas de estilo, scripts...) que são acessadas via

URL. O gerenciamento da configuração e comunicação entre essas páginas é realizado com suporte de um servidor WEB que hospeda as aplicações e responde as requisições de determinado usuário (LACERDA; OLIVEIRA, 2014).

Ao solicitar acesso a essas páginas HTML uma série de regras de comunicação são executadas entre o cliente e o servidor. Na WEB essas regras são definidas pelo protocolo de comunicação HTTP (*Hypertext Transfer Protocol*). Esse protocolo garante a conectividade e apesar de não ser o único é um dos mais importantes na WEB (MOLINARI, 2016).

O envio de dados que circulam entre o cliente e o servidor são enviados em texto puro quando o HTTP é utilizado. Nesse caso a comunicação não é segura, por isso é comum a utilização do HTTPS que proporciona mais segurança através das camadas SSL/TSL. A segurança do HTTPS é realizada através da certificação digital, do envio de informações criptografadas por chaves públicas e descriptografadas por chaves privadas no servidor. (PIMENTEL, 2018)

Para realizar uma requisição na WEB utiliza-se uma URL que é traduzido em IP pelo serviço DNS (Domain Name System). Para entender como esse processo funciona é necessário compreender o fluxo de requisições na WEB.

2.2.1 Fluxo de requisições na WEB

De modo geral o desenvolvimento WEB funciona com base em fluxo de requisições. Essas requisições podem ser feitas de diversas formas, mas independente disso quando o fluxo de requisições inicia, o navegador precisa enviar uma resposta para o usuário que solicitou algum serviço de um site (autenticação em uma página, acesso a um recurso específico...)

Para realizar o processo de autenticação, por exemplo, o envio de um formulário cujas informações de login são enviadas via HTTP para o servidor representa um request e o retorno do servidor representa o response (resposta), por isso o fluxo é conhecido como request-response e sempre inicia com uma solicitação do cliente

É importante ressaltar que mesmo que o formulário seja o mesmo, cada requisição deve conter todas informações e enviá-las para o servidor sempre que houver comunicação para que seja possível estabelecer um fluxo de respostas. Esse reenvio é necessário pois o HTTP não mantém o estado das requisições. Quando é necessário guardar os dados no usuário é necessário criar uma sessão, cada sessão possui um identificador temporário (PIMENTEL, 2018).

O fluxo de requisições pode ser acompanhado no navegador através de métodos que representam a ação da requisição. O método GET é acionado quando as requisições não implicam em algum tipo de modificação dos dados do servidor e quando é necessário fazer alguma modificação utiliza-se o método POST (PIMENTEL, 2018). Os status das respostas a estes métodos são representados por códigos padrões no HTTP conforme mostra a figura 3:

Figura 3 – Códigos HTTP

1XX Informational		4XX Client Error Continued	
100	Continue	409	Conflict
101	Switching Protocols	410	Gone
102	Processing	411	Length Required
2XX Success		412	Precondition Failed
200	OK	413	Payload Too Large
201	Created	414	Request-URI Too Long
202	Accepted	415	Unsupported Media Type
203	Non-authoritative Information	416	Requested Range Not Satisfiable
204	No Content	417	Expectation Failed
205	Reset Content	418	I'm a teapot
206	Partial Content	421	Misdirected Request
207	Multi-Status	422	Unprocessable Entity
208	Already Reported	423	Locked
226	IM Used	424	Failed Dependency
3XX Redirectional		426	Upgrade Required
300	Multiple Choices	428	Precondition Required
301	Moved Permanently	429	Too Many Requests
302	Found	431	Request Header Fields Too Large
303	See Other	444	Connection Closed Without Response
304	Not Modified	451	Unavailable For Legal Reasons
305	Use Proxy	499	Client Closed Request
307	Temporary Redirect	5XX Server Error	
308	Permanent Redirect	500	Internal Server Error
4XX Client Error		501	Not Implemented
400	Bad Request	502	Bad Gateway
401	Unauthorized	503	Service Unavailable
402	Payment Required	504	Gateway Timeout
403	Forbidden	505	HTTP Version Not Supported
404	Not Found	506	Variant Also Negotiates
405	Method Not Allowed	507	Insufficient Storage
406	Not Acceptable	508	Loop Detected
407	Proxy Authentication Required	510	Not Extended
408	Request Timeout	511	Network Authentication Required
		599	Network Connect Timeout Error

HTTP STATUS CODES

When a browser requests a service from a web server, an error may occur.
This is a list of HTTP status messages that might be returned.

Fonte: Google images (2018)

Os códigos da categoria “2xx” representam uma mensagem positiva do servidor, nesse caso a requisição foi realizada com sucesso, na categoria 3xx, em geral representam a inexistência de algum recurso solicitado na requisição. Ao receber o código 301, por exemplo, o navegador mostra o status e redireciona para um novo local, pois o código representa uma nova URL. Já os códigos na categoria 4xx representa o envio de dados errados. Na categoria 5xx estão os códigos que representam erros no servidor (PIMENTEL, 2018).

Para abstrair a complexidade gerada pela linguagem de programação “pura” é comum a utilização de uma base que facilite a programação para WEB essa é a ideia por trás da utilização de frameworks

2.3 FRAMEWORKS PYTHON

A partir do momento em que a linguagem de programação é definida surge a seguinte questão: que ferramenta ou conjunto de ferramentas utilizar para agregar valor a determinado projeto a partir da utilização desta linguagem? O tipo, a dimensão e as características gerais do projeto influenciam diretamente ou indiretamente essa decisão.

Nos projetos de desenvolvimento WEB existem processos relacionados a estruturas de dados complexas desenvolvidas para que o hardware ou software sejam capazes de exibir determinado conteúdo independente de suas características. Além disso, é necessário se preocupar com outras questões estruturais como a integração com banco de dados, por exemplo.

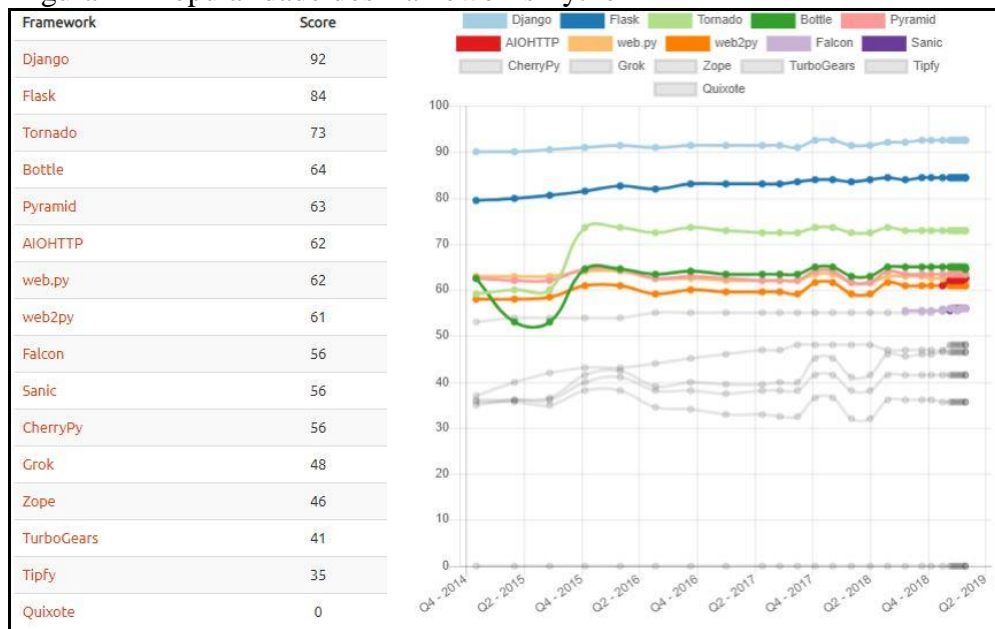
Dessa forma, esses projetos (se feitos com a linguagem de programação “pura”) exigem uma sobrecarga de conhecimento e trabalho exaustivos para o programador e, como consequência, os resultados podem levar mais tempo para serem alcançados. Para simplificar este processo existem softwares que facilitam a configuração de ambientes voltados para WEB, porque oferecem uma base estável para o desenvolvimento, com essas configurações de infraestrutura pré-programadas. Esses softwares são denominados frameworks.

Os frameworks configuram a estrutura geral do projeto “sua divisão em classes e objetos [...], como estas colaboram, e o fluxo de controle. Um framework predefine esses

parâmetros de projeto, de maneira que o projetista/implementador [...], possa se concentrar nos aspectos específicos da sua aplicação. ” (Gamma, et al., 2007, p. 42).

O que caracteriza o framework com uma ferramenta voltada para produtividade é, portanto, sua reusabilidade, extensibilidade e inversão de controle (PUC-RIO, 2006). No entanto, é importante notar que mesmo que uma linguagem já tenha sido definida e o programador tenha um certo domínio sobre sua sintaxe, ele ainda precisa determinar qual framework se adapta melhor ao conhecimento que ele possui e as características do projeto em que ele estiver inserido já que, em geral, existem diversos frameworks para uma única linguagem. Em Python, por exemplo existem mais de dez frameworks disponíveis, conforme mostra a figura 4:

Figura 4 – Popularidade dos frameworks Python



Fonte: Dados da pesquisa (2018)

O Tornado é um framework para rede assíncrona que pode ser escalado em diversas conexões abertas, pois usa I/O de rede sem bloqueio de processos, ou seja, permite o processamento paralelo. Os recursos assíncronos podem ser acessados através de bibliotecas como asyncio, conforme apresentado na figura 5:

Figura 5 - biblioteca assíncrona do tornado

```

async def post(self):
    cursor = self.get_argument("cursor", None)
    messages = global_message_buffer.get_messages_since(cursor)
    while not messages:
        # Save the Future returned here so we can cancel it in
        # on_connection_close.
        self.wait_future = global_message_buffer.cond.wait()
    try:
        await self.wait_future
    except asyncio.CancelledError:
        return
    messages = global_message_buffer.get_messages_since(cursor)
    if self.request.connection.stream.closed():
        return
    self.write(dict(messages=messages))

def on_connection_close(self):
    self.wait_future.cancel()

```

Fonte: GitHub (2018)

A principal diferença entre ele e a maioria dos frameworks Python é que ele não é baseado no WSGI (TORNADO, 2018) que especifica a forma de comunicação entre um servidor WEB e uma aplicação e como as aplicações processam uma solicitação (WSGI, 2011). Outra diferença é que normalmente o Tornado só utiliza uma thread por processo.

Já o Bottle é, na verdade, um microframework distribuído como um módulo de arquivo único e sua única dependência é a biblioteca padrão do Python. Possui suporte para URLs limpas e dinâmicas. Ele integra suporte para os templates mako, jinja2 e cheetah, além de suporte as requisições HTTP (BOTTLE, 2011). Na figura 6 é possível perceber que é muito simples inicializar uma aplicação em Bootle:

Figura 6 - Hello World em bottle

```

from bottle import route, run, template

@route('/hello/<name>')
def index(name):
    return template('<b>Hello {{name}}</b>!', name=name)

run(host='localhost', port=8080)

```

Fonte: Bootle (2011)

Assim como Bottle a proposta do framework pyramid é proporcionar um início rápido. Entretanto, em sua documentação, fica claro que no pyramid o início é pequeno, mas a estrutura final é grande (como em uma pirâmide), pois ele é focado no acabamento da aplicação. A estrutura inicial dele não é tão “limpa” quanto a do Bottle, mas também é simples como mostra a figura 7:

Figura 7 - Hello World em pyramid

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response

def hello_world(request):
    return Response('Hello World!')

if __name__ == '__main__':
    with Configurator() as config:
        config.add_route('hello', '/')
        config.add_view(hello_world, route_name='hello')
        app = config.make_wsgi_app()
        server = make_server('0.0.0.0', 6543, app)
        server.serve_forever()
```

Fonte: Pyramid (2018)

Apesar de aparecer na quinta posição, o pyramid é muito popular e tem uma proposta muito interessante, pois ele tenta combinar soluções para as dificuldades apresentadas pelos desenvolvedores ao finalizar suas aplicações sem que fiquem presos a decisões estabelecidas pelo próprio framework.

2.4 DJANGO

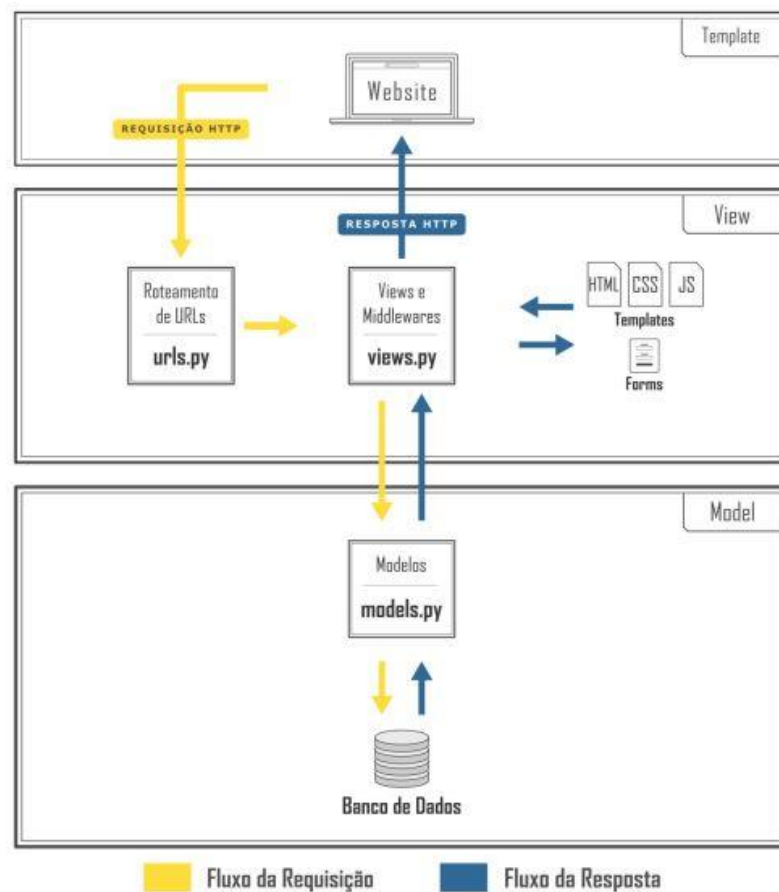
Django é um framework “de alto nível, escrito em Python que encoraja o desenvolvimento limpo de aplicações web” (RAMOS, 2018, p.8). A proposta do Django é “tornar tarefas comuns do desenvolvimento Web rápidas e fáceis” (DJANGO SOFTWARE FOUNDATION, 2017, p. 5), para isso ele encapsula as questões relacionadas ao tratamento de requisições, mapeamento objeto-relacional, as questões básicas de segurança, preparação de respostas e HTTP, por exemplo, para aumentar a produtividade durante desenvolvimento

de aplicações WEB. Essas tarefas básicas podem ser reutilizadas sempre que uma aplicação é criada.

Além disso, para testes, o Django possui um servidor que pode ser utilizado durante o processo de desenvolvimento, dessa forma não é necessário configurar um servidor externo como o apache antes de efetivamente colocá-lo em produção (DJANGO SOFTWARE FOUNDATION, 2017)

Já em relação a metodologia de desenvolvimento de software, ele é definido como um framework MVT (model-template-view), pois segundo Ramos (2018) no Django, uma view descreve a informação que será apresentada sem passar essa função para um controller especificamente, já que é o próprio framework que realiza o processamento e roteamento de requisições para view de acordo com a configuração de URL que for descrita, conforme representado no fluxo da figura 8:

Figura 8 - Fluxo de uma requisição em Django



Fonte: Ramos (2018)

Na figura 8 é possível visualizar que o fluxo de requisição, em um website, no framework inicia com uma requisição HTTP. Quando essa requisição é passada para o Django é realizado o roteamento de URLs através do “*urls.py*”. Após a verificação o Django passa a solicitação para views e, por fim, para o modelo onde. Nesse flux, ficam visíveis as três camadas do Django: models, views e templates.

No models do Django são configuradas as ações de acesso aos dados da aplicação “é nela que descrevemos os campos e comportamentos das entidades que irão compor nosso sistema” (RAMOS, 2018, p. 23). Nessa camada ficam as informações necessárias para integrar a aplicação com o banco de dados, pois cada modelo dá origem a uma tabela onde cada atributo descrito no modelo representa um campo da tabela no banco.

Já na camada de views são processadas as requisições dos usuários, através de funcionalidades como o roteamento de URLs, por isso é comum associar esta camada a efetivação das regras de negócio do sistema. As requisições são tratadas na View através de funções ou classes base do Django (CBVs) que facilitam o desenvolvimento (RAMOS, 2018).

Na camada templates é construída a interface da aplicação com os usuários do sistema. Na documentação, a camada de templates é definida como uma forma de gerar HTML dinamicamente, para isso um template contém a parte estática da saída desejada mais algumas sintaxes “especiais” que descrevem a forma como o conteúdo dinâmico será inserido (DJANGO SOFTWARE FOUNDATION, 2018). Um template em Django é composto por:

Variáveis que podem ser substituídas por valores, a partir do processamento por uma Engine de Templates (núcleo ou “motor” de templates). Usamos os marcadores `{{variável}}`.

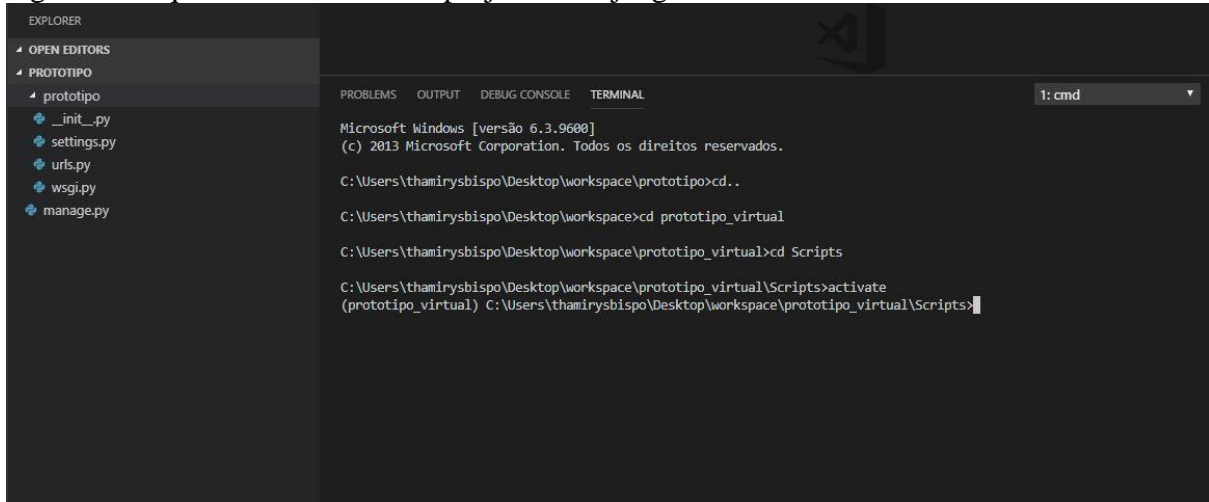
Tags que controlam a lógica do template. Usamos com `{% tag %}`.

Filtros que adicionam funcionalidades ao template. Usamos com `{{ variável|filtro }}`. (RAMOS, 2018, p. 82)

Quando se cria um projeto em neste framework, devem ser geradas algumas configurações básicas. É necessário “auto-gerar um código para iniciar um projeto Django – uma coleção de configurações para uma instância do Django, incluindo configuração do

banco de dados, opções específicas do Django e configurações específicas da aplicação” (DJANGO SOFTWARE FOUNDATION, 2018), conforme mostra a figura 9.

Figura 9 - arquivos iniciais de um projeto em Django



Fonte: Os autores (2018)

O arquivo “settings.py” armazena informações relacionadas as configurações gerais do projeto, tais como quais são os aplicativos que ou quais são as configurações gerais de banco de dados. Já o urls.py contém as configurações de rotas. O wsgi.py contém as configurações de interface entre o servidor e aplicação e o manage.py armazena comandos de manutenção das aplicações.

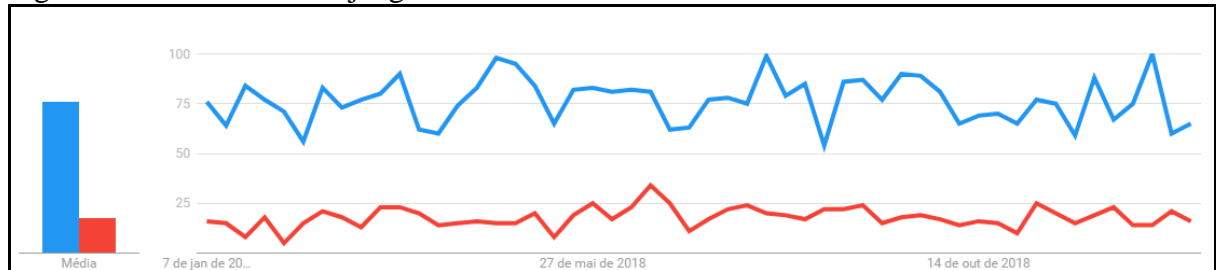
Tendo em vista que o Django é um framework WEB existe em sua composição a preocupação com as questões de segurança nesse ambiente. Exemplo disto é “a tag do Django { % csrf_token % } é obrigatório em todos os forms pois está relacionado à proteção que o Django provê ao CSRF - Cross Site Request Forgery” (RAMOS, 2018, p. 62). Também existem outras bibliotecas voltadas para aplicar segurança em Django.

2.5 FLASK

Em termos de interesse em pesquisas, o framework Django aparece em primeiro lugar entre os frameworks voltados para o desenvolvimento WEB, com Python, no *trend topics* do

Google. Em relação ao microframework Flask o Django apresenta níveis mais elevados de interesse em todos os períodos de 2018 conforme mostra a figura

Figura 10 - Interesse em Django e Flask em 2018



Fonte: Google Trends (2018)

Apesar de a média de interesse do Django ser visivelmente superior que a do Flask é precipitado afirmar que o Django é melhor. O Flask é um microframework, mas isso não é, necessariamente, uma limitação, pois ele possui uma série de recursos e segundo Ronacher (2013) é necessário considerar que o termo micro:

não significa que a sua aplicação web inteira tem que se encaixar em um único arquivo Python, embora certamente pode. Também não quer dizer que o Flask está faltando em termos de funcionalidade. O “micro” no microframework Flask visa manter o núcleo simples, mas extensível. Flask não vai tomar muitas decisões para você, como o banco de dados para usar. Essas decisões que ele faz, como o que motor de templates usar, são fáceis de mudar. Todo o resto é com você, então o Flask que pode ser tudo que você precisa e nada que você não faz.

Apesar do Flask ser um pacote com muitos recursos, em geral apenas parte destes recursos são explorados em uma aplicação, pois depende do contexto do projeto. Existem várias formas de incorporar o Flask em um projeto, a forma utilizada nesta pesquisa foi a utilização do gerenciador de pacotes do Python, o PIP3. Para o desenvolvimento da aplicação que será mostrada e na maioria dos projetos de pequeno porte é suficiente o import da classe Flask que possui o conjunto de recursos necessários para construir uma aplicação WEB com o CRUD completo.

Uma aplicação em Flask inicia com a **criação de um objeto do tipo Flask instanciado com o nome do pacote que será executado.** Esse pacote contém os arquivos que serão

implementados à medida que o projeto é desenvolvido. O conteúdo que será exibido em um navegador, a partir da criação do objeto, pode ser implementado de maneira muito simples, conforme mostra a figura 11:

Figura 11 – página estática em Flask

```
from flask import Flask

app_web = Flask(__name__)

@app_web.route('/')
def ola():
    return 'Página estática em Flask'
app_web.run()
```

Fonte: Os autores (2018)

Com apenas seis linhas de código foi possível gerar um conteúdo estático que pode ser exibido em qualquer navegador. As chamadas de funções para execução são reconhecidas no Flask através de diretivas que constituem as rotas que serão exibidas no navegador (`@app_web.route('/')`, por exemplo). Todo processamento da aplicação ocorre no Flask e para definir o local e a porta que por padrão é a 5000 basta enviar essas informações como parâmetros no método `run()`.

As rotas também são utilizadas para incluir recursos externos e tornar as páginas de um projeto mais dinâmicas. Para isso, o Flask utiliza *helpers* que podem ser importados para renderizar páginas HTML. Além disso, o caminho de acesso a arquivos que estão em locais diferentes também pode se tornar dinâmico através da função `url_for()` que acessa os arquivos dinamicamente ou criar URLs dinâmicas através da busca pelo nome do método que ela contém. Essa função é uma grande aliada no Flask quando se pensa em manutenção do código.

Ao renderizar páginas HTML o Flask permite o envio de conteúdo em variáveis que tornam os templates dinâmicos. A biblioteca que permite esta ação é a Jinja2 que também possibilita a inserção de código Python em templates no formato HTML através de diretivas (“`{{ variavel }}`” para variáveis e “`{% acao %}`” para ações). A Jinja 2 é, segundo Rocha (2014) um dos três pilares do Flask, por definição ela é:

um template engine escrito em Python, você escreve templates utilizando marcações como `{{ nome_da_variavel }}` ou `{% for nome in lista_de_nomes %} Hello {{nome}}!! {% endfor %}` e o Jinja se encarrega de renderizar este template, ou seja, ele substitui os placeholders pelo valor de suas variáveis. O Jinja2 já vem com a implementação da maioria das coisas necessárias na construção de templates html e além disso é muito fácil de ser customizado com template filters, macros etc. (ROCHA, 2014)

A lógica relacionada às requisições neste microframework é a mesma apresentada sobre o desenvolvimento WEB e na prática ela é implementada com as definições de métodos nas rotas do código que por padrão recebe o método GET. O Flask possui algumas convenções que podem ser alteradas pelo desenvolvedor. Por exemplo: “templates e arquivos estáticos são armazenados em subdiretórios dentro da aplicação na árvore fonte do Python, com o nomes templates e static respectivamente” (RONACHER, 2013). Nesse caso, HTMLs ficam no diretório templates, e o diretório static é a pasta padrão para arquivos .css

2.5.1 Persistência

É possível integrar o Flask com uma variedade de SGBDs através de extensões, mas para esta pesquisa foi definido o MySQL. O Flask disponibiliza a extensão MySQLdb para integração com banco de dados no MySQL. A conexão com o banco é realizada através da função *conn()* que recebe como ‘parâmetros’ relativas ao usuário e ao banco de dados que será utilizado.

A lógica necessária para acessar comandos essenciais de banco de dados, tais como CREATE, UPDATE e DELETE no Flask associado a um banco de dados relacional (MySQL, nesse caso) é encapsulada em classes DAO conforme mostra a figura 12:

Figura 12 - Classe ProdutoDao

```

class ProdutoDao:
    def __init__(self, db):
        self.__db = db

    def salvar(self, produto):
        cursor = self.__db.connection.cursor()

        if (produto.id):
            cursor.execute(SQL_ATUALIZA_PRODUTO, (produto.nome, produto.categoria, produto.valor, produto.id))
        else:
            cursor.execute(SQL_CRIA_PRODUTO, (produto.nome, produto.categoria, produto.valor))
            produto.id = cursor.lastrowid
        self.__db.connection.commit()
        return produto

```

Fonte: Os autores (2018)

Os comportamentos das *queries* de acesso ao banco foram abstraídos para facilitar o entendimento do código. Essa abstração não é automática (oferecida pelo Flask). Ou seja, cada query foi programada durante o desenvolvimento da aplicação.

2.6 MODELAGEM DE SISTEMAS

Os riscos gerados pela utilização inconsciente dos recursos computacionais em um projeto de software resultam em falhas desde a concepção da aplicação até a finalização quando o tempo estipulado para finalizá-la é ultrapassado. Segundo Pressman (2011) para entender melhor as necessidades do software e, por conseguinte do projeto de desenvolvimento que irá atender essas necessidades é necessário criar modelos que sirvam como esboço do produto final e que passem uma ideia geral do todo que será construído.

Para Gudwin (2015) esses modelos são parte de uma análise difundida na engenharia de software para universalizar a “linguagem” de construção de softwares entre os desenvolvedores e participantes de sistemas. Ele ainda afirma que essa linguagem é “geralmente uma linguagem visual, que especifica a sintaxe e a semântica de classes de diagramas, utilizados explicitamente para modelar partes de um sistema de software ou de planos de construção de sistemas de software” (GUDWIN, 2015, p. 6)

Com sua utilização objetiva-se representar a visão global que se tem do projeto em modelos que sejam compreensíveis e úteis para o desenvolvimento de cada componente do

sistema, pois percebe-se que cada parte é dependente de algum aspecto comportamental do projeto como um todo. Em suma pode-se dizer que:

A aplicação de técnicas de modelagem no processo de engenharia de software [...] simplifica representações da realidade possibilitando o seu entendimento. Por meio dos modelos, podemos ter a representação de diversas perspectivas para um determinado projeto, melhorando a sua compreensão. Esta possibilidade de compreensão advém da dificuldade de entendimento para a complexidade. A partir do momento que se divide um problema em perspectivas ou visões, estes podem ser controlados e entendidos de uma melhor forma, pois a sua fronteira se torna mais clara. Controlando a sua abstração podemos ter modelos de alto nível ou de baixo nível, direcionando-os para públicos diferentes conforme sua necessidade. (JOÃO, 2015)

Tendo este conceito em vista, percebe-se que no campo de desenvolvimento WEB existem características que diferenciam os WebApps de outros softwares que precisam ser consideradas no processo de modelagem (em etapas específicas). A propagação massificada da informação, por exemplo, remete na modelagem a preocupação como o uso intensivo de redes que aumenta o alcance do software a uma comunidade diversificada de clientes em diversos lugares no mundo (PRESSMAN, 2011). Na modelagem esse aspecto impacta na definição do público alvo do sistema.

Outras características são: a simultaneidade que diz respeito ao número de pessoas que podem acessar o site ao mesmo tempo. A carga não previsível que diz respeito a impossibilidade de quantificar quantas pessoas vão acessar o site em dias ou horários específicos. O desempenho esperado pelas partes interessadas. A disponibilidade que indica a necessidade de manter o WebApp sempre disponível para acesso. A usabilidade e capacidade de evoluir (PRESSMAN, 2011). As características gerais de um sistema podem ser representadas através do padrão UML.

2.7 UML

Para modelar o sistema desenvolvido durante a pesquisa utilizou-se o padrão estabelecido pela Linguagem de Modelagem Unificada (UML) que é adotado internacionalmente. A UML “é uma linguagem visual utilizada para modelar softwares baseados no paradigma de orientação a objetos. É uma linguagem de modelagem de propósito geral que pode ser aplicada a todos os domínios de aplicação” (GUEDES, 2011, p.19).

Para Corrêa e Aniche (2018) “o grande objetivo da UML (a Unified Modelling Language) é facilitar a comunicação entre a equipe de desenvolvimento, entre os stakeholders, e quaisquer outros interessados naquele software”. Esse aspecto é fundamental para garantir que todas as partes interessadas no sistema entendam a ideia geral do projeto.

2.8 FERRAMENTAS UTILIZADAS

Para programar em Flask utilizou-se o PyCharm, uma IDE criada pela empresa checa JetBrains. É uma das IDE's (Integrated Development Environment ou Ambiente de Desenvolvimento Integrado), mais utilizadas para desenvolvimento em Python da atualidade. É largamente usada por programadores profissionais e iniciantes, pois apresenta uma interface limpa e personalizável, dando ao usuário uma integração amigável.

Com uma série de funções nativas, o ambiente do PyCharm, propicia ao usuário dinamismo e facilidades, como a edição de códigos inteligentes, refatoração de códigos, navegação inteligente por códigos, depuração, teste e criação de perfis, ferramentas de banco de dados, entre outros recursos.

O PyCharm, se apresenta em duas versões: PyCharm Community Edition, esta, com código open source e totalmente free, disponibiliza uma série de recursos, o que possibilita uma excelente experiência para programadores iniciantes; e a versão PyCharm Professional Edition, esta, com uma licença proprietária, que, agrega uma série de recursos que não estão presentes na versão anterior, tais como: ferramentas científicas, desenvolvimento web,

frameworks web em Python, perfilador de Python, capacidades de desenvolvimento remotas e suporte a banco de dados e SQL.

Já o projeto em Django foi desenvolvido no Visual Studio code por duas razões: Versão Community do Pycharm não dá suporte ao Django e o projeto foi desenvolvido no sistema operacional Windows. Outros editores de texto e sistemas operacionais poderiam ter sido utilizados na pesquisa, o sublime text3, por exemplo, disponibiliza plugins para programar em Django e o Linux no stack overflow aparece como melhor opção para programar em Django.

Entretanto, o critério adotado para escolha das ferramentas foi buscar coerência com os objetivos da pesquisa, ou seja, escolher ferramentas que contribuíssem com o objetivo de identificar a carga mínima de trabalho para desenvolver aplicações WEB com Python. Dessa forma, decidiu-se utilizar ferramentas que são mais comuns no contexto de aprendizagem.

O VSCode é um editor de código desenvolvido pela Microsoft, mas é multiplataforma e possui suporte à sintaxe de diversas linguagens, inclusive Python. Ele possui comandos do Git embutidos, um sensor inteligente, o “IntelliSense”, que a partir da digitação do código resulta em conclusões lógicas com base nas variáveis e nos demais elementos da linguagem utilizada. Além disso, é extensível e personalizável (MICROSOFT, 2018). Para o desenvolvimento desta pesquisa foi necessário instalar plugins adicionais para ter acesso algumas características da sintaxe de Django.

Para o VSCode reconhecer características do Python, como indentação, foi instalado o **plugin do Don Jayamanne**. Também foi instalada a extensão Django Snippets de templates para Django que oferece um sistema de cores para destacar partes específicas do código e outras funções. Também foi instalado o djaneiro e outros plugins para aumentar a produtividade ao programar para Django no VSCode.

Para modelagem foi utilizado o Astah que é um software cuja sua característica principal e a proposta para qual foi desenvolvido, é a concepção de modelagem UML - Unified Modeling Language. O software traz diversas formas de modelagem em “bloco”, utilizando formas geométricas para roteirizar, ou como o próprio nome sugere, modelar softwares e/ou banco de dados a partir da utilização dessas formas (ASTAH, 2018).

O software foi criado pela empresa japonesa Change Vision, e, se tornou um dos softwares mais utilizados em seu segmento. Para a engenharia de software, o uso dessa ferramenta é de fundamental importância na concepção e modelagem das ideias que surgem para a construção de um novo programa ou banco de dados. (ASTAH, 2018)

O Astah se apresenta em versões com licenças pagas e gratuitas. A opção gratuita é mais do que suficiente para o aprendizado acadêmico na modelagem e concepção de programas. Alunos do âmbito acadêmico, podem ainda, adquirir a versão Professional com todos os recursos incluídos de forma totalmente gratuita, para isso basta preencher um cadastro no site da desenvolvedora e informar um e-mail acadêmico para a validação de vínculo com instituição de ensino. (ASTAH, 2018)

A interface é relativamente simples, e dependendo da escolha de modelagem do usuário, a barra de ferramentas deverá apresentar opções diferentes. Mas de uma forma geral é de fácil utilização, bem como ágil em sua proposta. (ASTAH, 2018)

3 MODELAGEM E IMPLEMENTAÇÃO

Ao definir o escopo do projeto levou-se em consideração que a ideia desta pesquisa é comparar qual a carga de trabalho mínima utilizada para construir uma aplicação WEB nos frameworks Django e Flask. Para alcançar esse objetivo foi necessário criar um sistema de gerenciamento de produtos para um e-commerce em ambas as tecnologias com as mesmas características funcionais.

3.1 REQUISITOS

No protótipo final os sistemas em Django e Flask devem atender, no mínimo, aos seguintes requisitos:

- Quando o usuário digitar o endereço do site ele deverá ser encaminhado para página inicial onde poderá ver os produtos que estão disponíveis e informações básicas sobre estes produtos.
- A página de listagem deve conter todos os produtos disponíveis no banco e um link para as ações de cadastro de novo produto, atualização do produto e exclusão de produtos.;
- A página de cadastro de novo produto deve conter os campos relacionado aos atributos editáveis no banco de dados;
- Os dados incluídos devem ser armazenados em um banco de dados baseado em SQL;
- A página de atualização deve conter um formulário que retorne o produto corresponde ao que foi mostrado na tabela de produtos através da recuperação via id do banco de dados escolhido;
- A interface deve ser realizada com recursos da própria ferramenta ou suas bibliotecas.

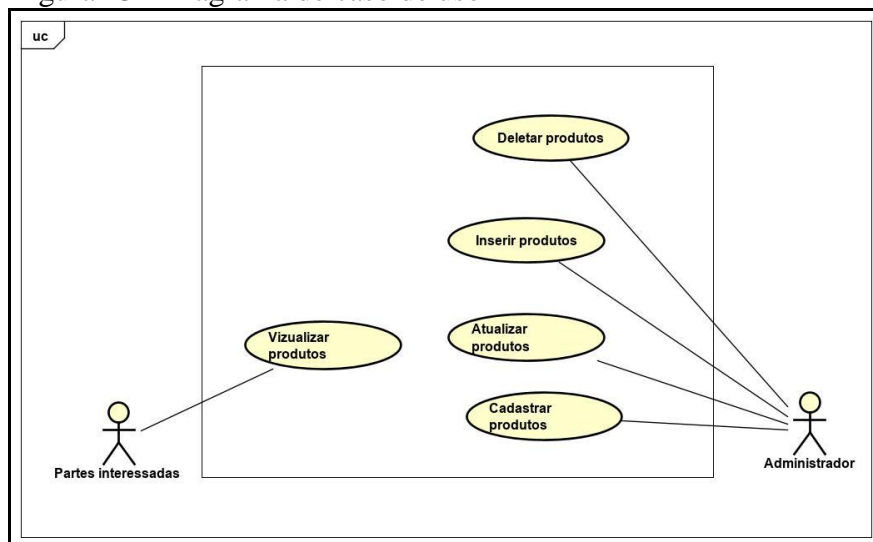
- Os recursos externo necessários para fazer o design da interface devem ser retirados do Bootstrap;

3.2 MODELAGEM DO SISTEMA

Os modelos foram feitos no início da pesquisa apenas para entender o ponto de partida das aplicações e qual o produto final mínimo. A modelagem foi feita dessa forma, pois a pretensão foi executar o que é exigido dentro das disciplinas e evoluir o protótipo conforme os recursos que o Flask e o Django oferecem.

Ou seja, o estado apresentado na modelagem é comum aos dois protótipos e ambos tinham que conter, no mínimo, as características modeladas nos diagramas abaixo. Mas a evolução de cada protótipo ocorreu de forma diferente (para atender os prazos estabelecidos) e para que fosse possível medir a carga de trabalho mínima para desenvolver uma aplicação funcional em cada ferramenta. A figura 13 mostra o diagrama de caso de uso do protótipo:

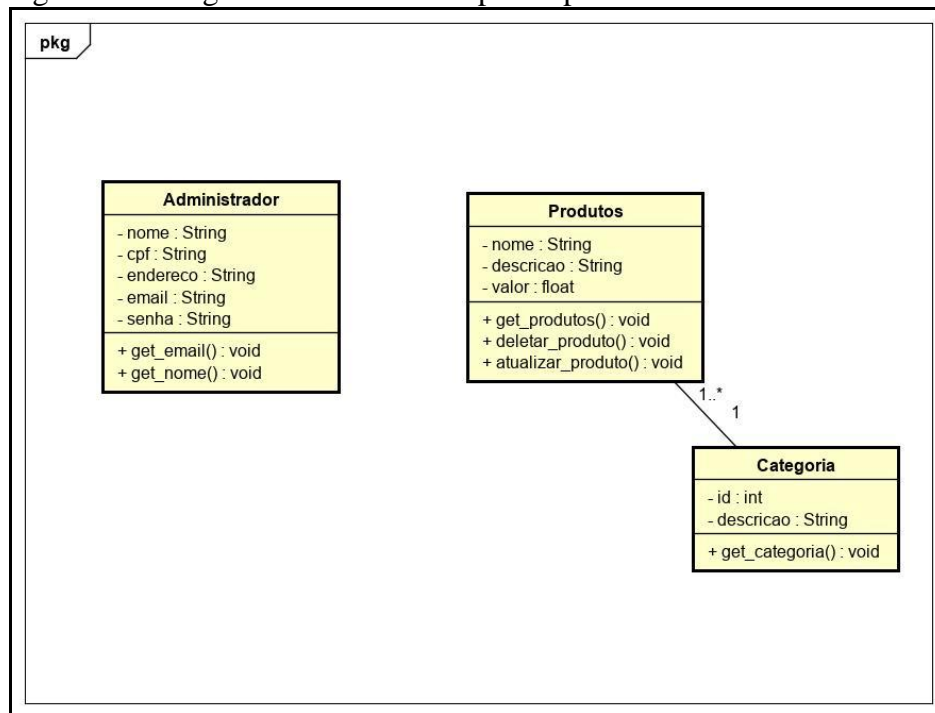
Figura 13 - Diagrama de caso de uso



Fonte: Os autores (2018)

A figura 14 mostra o modelo conceitual do diagrama de classes utilizado para a construção das aplicações:

Figura 14 - Diagrama de classes dos protótipos



Fonte: Os autores (2018)

Esse diagrama da figura 14 mostra, de forma estática, da estrutura das principais classes utilizadas nas aplicações. Não foram incluídas as classes automáticas geradas pelo framework, pois elas não foram definidas nesta pesquisa, mas foram incluídas conforme a necessidade e não se aplicam as duas ferramentas (CBVs do Django).

3.3 IMPLEMENTAÇÃO DJANGO

Para implementar o projeto em Django foi instalada a versão 3.6 do Python no Windows e foi criada uma máquina virtual para isolar as dependências específicas desta pesquisa de outros projetos realizados com o framework. Esse ambiente virtual possibilitou o desenvolvimento do projeto em versões diferentes do Django, mas o projeto final foi desenvolvido com a versão 2 do Django que foi instalada através do gerenciador de pacotes do Python.

Por uma questão de organização foi criado um diretório para o projeto final onde a ambiente virtual foi adicionado através do comando `"Python -m venv maquina_virtual"`. Para ativar o ambiente virtual foi utilizado o comando `"activate"` que, no Windows, se encontra no

subdiretório *Scripts* . Uma vez que a máquina virtual tenha sido ativada tudo que for instalado só existirá em seu escopo. A figura 15 mostra a instalação da versão 1.11.5 do Django na máquina virtual criada:

Figura 15 – Instalação do Django na máquina virtual

```
C:\Users\thamirysbispo\Desktop\tcc_python\maquina_virtual> cd Scripts

C:\Users\thamirysbispo\Desktop\tcc_python\maquina_virtual\Scripts>activate
(maquina_virtual) C:\Users\thamirysbispo\Desktop\tcc_python\maquina_virtual\Scripts>pip install django==1.11.5
Collecting django==1.11.5
  Using cached https://files.pythonhosted.org/packages/18/2d/b477232dd619d81766064cd07ba5b35e956ff8a8c5c5d41754e0392b96e3/Django-1.11.5-py2.py3-none-any.whl
Collecting pytz (from django==1.11.5)
  Using cached https://files.pythonhosted.org/packages/61/28/1d3920e4d1d50b19bc5d24398a7cd85cc7b9a75a490570d5a30c57622d34/pytz-2018.9-py2.py3-none-any.whl
Installing collected packages: pytz, django
Successfully installed django-1.11.5 pytz-2018.9
You are using pip version 9.0.1, however version 19.0.2 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

(maquina_virtual) C:\Users\thamirysbispo\Desktop\tcc_python\maquina_virtual\Scripts>
```

Fonte: Os autores (2018)

Com esse comando já foi possível inserir na máquina virtual todos os recursos do Django (o core, middlewares, sistema de URLs, códigos de acesso a banco de dados...) e caso a máquina seja desativada não será possível acessar qualquer recurso do Django. É possível criar quantas máquinas virtuais forem necessárias e instalar diferentes recursos.

Para iniciar o desenvolvimento do projeto em Django foi utilizado o comando “*Django-admin startproject tcc*” que gera a estrutura básica para uma aplicação web e para criar uma aplicação dentro do projeto tcc foi executado o comando “*Django-admin startapp prototipo*”. Esse comando também gera, automaticamente, alguns arquivos de configuração, mas para é necessário criar uma estrutura de diretórios manualmente. Nessa etapa, foram criados os diretórios de templates e arquivos estáticos além da centralização do modelo do projeto no diretório “tcc”.

Para cumprir os requisitos estipulados para o sistema optou-se por utilizar as CBVs do Django. Sua utilização requer o uso de boas práticas para facilitar o entendimento do comportamento definido em cada método, tais como a nomenclatura que segue o padrão (no caso de templates) “nome da página que será renderizada + o nome da CBV”, a figura 16 mostra como ficou a configuração da CBV para o template que renderiza a página inicial do sistema:

Figura 16 - configuração da CBV para página inicial

```
class IndexTemplateView(TemplateView):
    template_name = "prototipo/index.html"
```

Fonte: Os autores (2018)

Essas classes foram herdadas de acordo a necessidade da aplicação, dessa forma, para renderizar a página inicial criou-se uma classe Index que herda as características da CBV “*TemplateView*” cuja responsabilidade é renderizar templates, conforme mostra a figura 16.

Já a listagem de produtos foi codificada através da CBV *ListView*. Ela puxa todos os dados do modelo que estiver linkado em sua configuração e deixa disponível para visualização na página lista_produtos.html. A figura 17 mostra o resultado no navegador:

Figura 17 – Lista de produtos em Django

Nome	Descrição	Valor	Ações
caderno	artigo de papelaria	25.00	Atualizar Excluir
Notebook	Informática	2500.00	Atualizar Excluir
Pasta	Papelaria	5.00	Atualizar Excluir
			Cadastrar Produto

Fonte: Os autores (2018)

Com essa configuração já foi possível passar para o template os dados dos produtos armazenados no banco de dados SQLite3 que já fica disponível no Django assim que o comando para criar uma aplicação é executado. Apesar de não ser um banco de dados robusto ele se integra muito bem com o Django e facilita os testes, pois a integração é feita de forma automática.

Para modelar a forma como os dados devem ser reconhecidos pelo banco criou-se uma classe modelo no diretório “*models*” do Django. Cada atributo dessa classe (nome, descrição e

valor) contém o tipo correspondente a classe *Model* (classe criada pelo Django) que é herdada pela classe *Produtos*, conforme mostra a figura 18.

Figura 18 - Modelo da classe *Produtos*

```
34 class Produtos(models.Model):
35     nome = models.CharField(
36         max_length=255,
37         null=False,
38         blank=False
39     )
40     descricao = models.CharField(
41         max_length=255,
42         null=False,
43         blank=False
44     )
45
46     valor = models.DecimalField(
47         max_digits=8,
48         decimal_places=2,
49         null=False,
50         blank=False
51     )
52
53     objetos = models.Manager()
```

Fonte: Os autores (2018)

Essas classes facilitam as operações realizadas no banco de dados. Toda alteração realizada no banco é facilmente replicada na aplicação com o comando migrate. O mais trabalhoso no Django foi configurar a base para cada comportamento, mas a partir do momento que cada configuração é realizada de forma adequada o framework automatiza diversas funções.

Exemplo disto é a atualização dos dados dos produtos que foi implementada a partir da CBV *UpdateView*, conforme mostra a figura 19. Para realizar a atualização não foi necessário executar nenhuma linha de SQL, nem configurar o comportamento dos métodos da classe *update*, pois eles são abstraídos pelo Django.

Figura 19 - View de atualização de dados

```

78 class ProdutosUpdateView(UpdateView):
79     template_name = "prototipo/atualiza_produtos.html"
80     model = Produtos
81     fields = '__all__'
82     context_object_name = 'produtos'
83     success_url = reverse_lazy("prototipo:lista_produtos")
84

```

Fonte: Os autores (2018)

No sistema desta pesquisa foi definido que todos os atributos podem ser atualizados, mas conforme a necessidade é possível restringir esta operação para atributos específicos através da variável `fields`. Essas configurações afetam os forms definidos para aplicação. As classes definidas para criação dos forms no Django herdam as características da classe `ModelForm` e especificam quais dados devem ser mostrados no formulário e como eles devem ser apresentados.

Todas as características implementadas na aplicação foram interligadas através das rotas definidas no arquivo `urls.py` do protótipo. A figura 20 mostra como as CBVs são implementadas nas rotas.

Figura 20 – Sistema de URLs no Django

```

26 # GET /produtos
27 path('produtos/', ProdutosListView.as_view(), name="lista_produtos"),
28
29 # GET /produtos/cadastrar
30 path('produtos/cadastrar', ProdutosCreateView.as_view(), name="cadastra_produtos"),
31
32 # GET/POST /produtos/{pk}
33 path('produtos/<pk>', ProdutosUpdateView.as_view(), name="atualiza_produtos"),
34
35 # GET/POST /produtos/excluir/{pk}
36 path('produtos/excluir/<pk>', ProdutosDeleteView.as_view(), name="deleta_produtos"),
37

```

Fonte: Os autores (2018)

A lógica para cadastrar e excluir produtos é similar ao que foi apresentado, pois também utilizou-se CBVs para atender estas requisições. No final do período estipulado para desenvolver o protótipo em Django todos os requisitos do projeto mínimo esperado foram cumpridos, mas não foi possível aplicar funções adicionais. O tempo necessário para entender

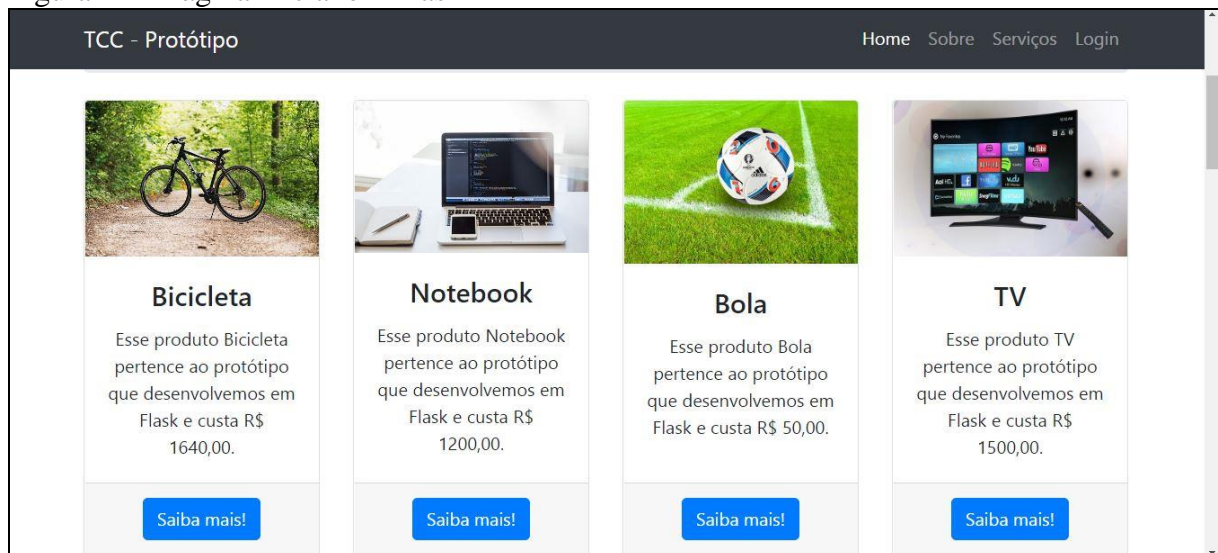
o funcionamento das configurações do framework consumiram grande parte do tempo determinado.

3.4 IMPLEMENTAÇÃO EM FLASK

A primeira requisição feita ao sistema é a de acesso a página inicial. Essa requisição é reconhecida no protótipo por meio de uma diretiva “`@web.route()`” que indica o nome da aplicação criada. Cada diretiva criada recebeu como parâmetro o nome da página que será retornada ou retorna um redirect para próxima diretiva que deve ser acionada. Na prática esse processo foi muito simples, pois as definições de funções foram feitas exatamente como estudado em Python e não houve, nessa etapa, nenhuma abstração explícita do Flask que dificulta-se o entendimento dos processos que estavam ocorrendo durante a requisição.

Uma vez que a requisição foi atendida o usuário consegue visualizar a página apresentada na figura 21:

Figura 21 – Página inicial em Flask



Fonte: Os autores (2018)

Os dados apresentados na página inicial são recuperados do MySQLdb. Nessa etapa foram aplicados vários recursos da linguagem Python. Na conexão com o banco de

dados, por exemplo, aplicaram-se tuplas, dicionários, listas e outros elementos próprios da linguagem.

A criação do banco de dados no Flask foi a etapa do desenvolvimento em que foi necessário recorrer diversas vezes aos fóruns, pois o formato em que os dados eram gerados não eram compatíveis com os dados esperados pelo banco, por conseguinte os campos eram exibidos na tela, mas os dados não.

Quando os problemas com os formatos foram resolvidos ainda foi necessário resolver os problemas com repetição de código. Para isso, as queries SQL foram centralizadas em uma variável e sempre que era necessário utilizar a query em uma função do Python bastava incluir a variável e não a query inteira. Essa decisão facilitava, inclusive, o entendimento do código, conforme mostra a figura 22:

Figura 22 - função salvar produto

```
15 def salvar(self, produto):
16     cursor = self.__db.connection.cursor()
17
18     if (produto.id):
19         cursor.execute(SQL_ATUALIZA_PRODUTO, (produto.nome, produto.categoria, produto.valor, produto.id))
20     else:
21         cursor.execute(SQL_CRIA_PRODUTO, (produto.nome, produto.categoria, produto.valor))
22         produto.id = cursor.lastrowid
23     self.__db.connection.commit()
24     return produto
```

Fonte: Os autores (2018)

A implementação do banco de dados também possibilitou a inserção da página de login cujo layout foi adaptado a partir do template da página inicial (com poucas alterações), assim como em todas as páginas do sistema, através do recurso de reuso (herança de templates) conforme mostra a figura 23:

Figura 23 - Tela de login

A imagem mostra a interface de login de um sistema web. No topo, há uma barra de navegação escura com o texto "TCC - Protótipo" à esquerda e links "Home", "Sobre", "Serviços" e "Login" à direita. O conteúdo principal é uma caixa cinza clara com o título "Login" em negrito. Abaixo do título, há dois campos de entrada brancos, um para "Email" e outro para "Password". Abaixo desses campos, há uma opção "Lembrar" com uma caixa de seleção vazia. Um botão azul com o texto "Entrar" está posicionado abaixo da opção. No rodapé, uma barra escura contém o texto "© UFRA 2018".

Fonte: Os autores (2018)

Para ter acesso a alguns recursos do sistema (cadastro, atualização ou exclusão de produtos) é pré-requisito que o usuário esteja logado. Os dados de autenticação inseridos no formulário são enviados através do método POST. Para fins de usabilidade o usuário recebe uma mensagem padrão para visualizar se está logado ou não. Essa mensagem foi inserida através da função *flash()* (importada no pacote Flask).

Quando a requisição para *login* do usuário é atendida e validada aplica-se o conceito de capítulo no Flask. Como o status pode ser logado ou não logado o Flask trata dois ciclos de requisição. No primeiro ciclo o usuário é redirecionado para a página inicial (se realizou o acesso direto a página de *login*) ou a página que tentou acessar antes de autenticar e no segundo ciclo ele volta para página de *login*.

Para que os dados do usuário sejam mantidos nesse processo eles precisam ser armazenados em uma capítulo. A capítulo inicia com a criação do objeto *session*. Os dados da capítulo são criptografados e para que sejam utilizados é necessário criar uma chave secreta no Flask. A figura 24 mostra a rota para autenticação.

Figura 24 - rota para autenticar o usuário

```

38
39 @app_web.route('/autentica_usuario', methods=['POST',])
40 def autentica_usuario():
41     usuario = usuario_dao.buscar_por_id(request.form['usuario'])
42     if usuario:
43         if usuario.senha == request.form['senha']:
44             session['usuario_logado'] = usuario.id
45             flash(usuario.nome + ' logou com sucesso!')
46             proxima_pagina = request.form['proxima']
47             return redirect(proxima_pagina)
48         else:
49             flash('Login ou senha incorretos')
50             return redirect(url_for('login'))
51

```

Fonte: Os autores (2018)

Ao logar no sistema o administrador tem acesso a tabela de produtos e as ações de cadastro de novo produto, atualização das informações sobre o produto e a opção de excluir o produto, conforme mostra a figura 25. O processo de logout funciona com a mesma estrutura da sessão e para efetivá-lo basta limpar os dados do usuário logado (passar o valor *None* para capítulo)

Figura 25 - Listagem de produtos em Flask

TCC - Protótipo				Home	Sobre	Serviços	Login
Tapete	Casa	200	Editar Deletar				
Amaciante	Limpeza	2	Editar Deletar				
Vassoura	Limpeza	10	Editar Deletar				
Mapa	Utilitários	7	Editar Deletar				
Globo	Utilitários	20	Editar Deletar				

[Novo Produto](#)

Fonte: Os autores (2018)

Para mostrar as informações listadas conforme armazenadas no banco de dados utilizou-se o sistema de diretivas do Flask, no template, com esta configuração: “`{{nome do atributo}}`”. O sistema de diretivas é muito simples de implementar e permite, por exemplo,

replicar características da própria interface, como os blocos de conteúdo do bootstrap, conforme mostra a figura 26:

Figura 26 – Template em laço na página inicial

```

1  {% extends "template.html" %}
2  {% block conteudo %}
3
4
5  <header class="jumbotron my-4">
6    <h1 class="display-3">Protótipo TCC</h1>
7    <p class="lead">Desenvolvido em Flask</p>
8    <a href="#" class="btn btn-primary btn-lg">Call to action!</a>
9  </header>
10
11
12  <div class="row text-center">
13
14    {% for produto in produtos %}
15      <div class="col-lg-3 col-md-6 mb-4">
16        <div class="card">
17          
18          <div class="card-body">
19            <h4 class="card-title">{{ produto.nome }}</h4>
20            <p class="card-text">Esse produto {{ produto.nome }} pertence ao protótipo que desenvolvemos em Flask e custa R$ {{ produto.valor }}
21          </div>
22          <div class="card-footer">
23            <a href="#" class="btn btn-primary">Saiba mais!</a>
24          </div>
25        </div>
26      </div>
27
28    {% endfor %}
29
30  </div>
31  <!-- /.row -->
32
33  {% endblock %}

```

Fonte: Os autores (2018)

As informações sobre os produtos foram organizadas dentro da classe produtos que também foi modelada no banco de dados (assim como a classe Usuario). Ela ficou acessível em todo projeto (através de herança). Mas para passar as informações do usuário (descritas na classe) através dos formulários foi necessário importar o *helper request* do Flask e a partir dele recuperar os dados de requisição nas funções implementadas nas rotas.

As informações do formulário de cadastro de novos produtos, representado na figura 27, foram recuperadas, por exemplo, através dos seguintes *requests*: “*request.form['nome']*”, *request.form['categoria']* e *request.form['valor']*”.

Figura 27 – Tela de cadastro de produtos

Fonte: Os autores (2018)

Esse recurso também foi utilizado para atualizar informações sobre o produto. Para recuperar os dados do banco e passar para o formulário de edição cada produto foi individualizado através de um *id* que representa um objeto produto (instanciado). O *id* também foi passado para rota, dessa forma quando o usuário clica em editar são mostrados na tela de edição de produtos, os dados relativos ao *id* de um produto, conforme mostra a figura 28:

Figura 28 - rota para atualizar produtos

Fonte: Os autores (2018)

A figura 29 mostra como *id* do produto é capturado na requisição. Mostra, também, como o Flask redireciona a rota atualizar para página inicial quando o processamento termina

(através do “`redirect (url_for('index'))`”). O recurso `redirect` precisa ser importado para aplicação no Flask.

Figura 29 – rota para atualizar produtos

```
88 @app_web.route('/atualizar', methods=['POST',])
89 def atualizar():
90
91     nome = request.form['nome']
92     categoria = request.form['categoria']
93     valor = request.form['valor']
94     produto = Produto(nome, categoria, valor, id=request.form['id'])
95     produto_dao.salvar(produto)
96     return redirect(url_for('index'))
97
```

Fonte: Os autores (2018)

O *id* também foi utilizado para excluir produtos na aplicação. As ações aplicadas no navegador foram replicadas da forma esperada no banco de dados, com isso o CRUD da aplicação foi implementado com sucesso e os requisitos foram atendidos.

Dentro do tempo estipulado para pesquisa ainda foi possível implementar funções adicionais, tais como o preview das imagens associadas a descrição de produtos. Para fazer isso também foi necessário implementar o recurso de arquivos no Flask (essa função não estava no escopo). A sessão também foi criada como função adicional, pois a princípio não era um requisito do protótipo.

4 CONCLUSÕES

Ao analisar a quantidade de aplicações WEB disponíveis em diversos tipos de dispositivos eletrônicos é comum associar a valorização do software a evolução do hardware. De fato, o hardware tem um peso importante nesse cenário, mas ele é apenas parte do processo. Prova disto é a série de ações precisam ser projetadas no software para enviar e receber informações através de um navegador.

Visto que esta pesquisa está relacionada com a prática do aluno na Universidade associada às atividades que serão conferidas a ele, como profissional, no mercado. Através desta pesquisa concluiu-se que Python é uma linguagem adequada para evoluir nas disciplinas durante o curso, pois ela possui todas as características que são abordadas do início ao fim da graduação.

Para alcançar os objetivos das disciplinas sem comprometer o desenvolvimento do aluno é importante que as ferramentas associadas à linguagem ofereçam suporte ao cumprimento dos requisitos de prazo, qualidade e desempenho esperados. O objetivo da pesquisa conforme será mostrado na subcapítulo seguinte foi alcançado, pois foi possível determinar qual a ferramenta que melhor se adequa a realidade dos alunos do curso de sistemas da UFRA.

4.1 SOBRE O USO DOS FRAMEWORKS

Na primeira etapa de desenvolvimento dos projetos em Django e Flask a intenção foi conhecer a estrutura geral de cada um e, sobretudo, identificar como o Python foi utilizado para construir as estruturas de dados que funcionam nos módulos. Foi perceptível que mesmo se tratando de frameworks ou microframeworks o domínio sobre a linguagem de programação base é muito importante, porque facilita o entendimento de determinados comportamentos (erros, respostas do software a determinada ação...).

Nesse sentido, uma das características identificadas em Flask como um ponto forte foi que em todas as etapas de programação ele não se distanciou expressivamente da

programação explícita em Python. Dentro da perspectiva do ensino e aprendizagem esse aspecto é muito importante para os alunos, pois evita a sobrecarga de informações uma vez que em Django, por exemplo, algumas estruturas utilizadas para facilitar o desenvolvimento fogem do controle do desenvolvedor, pois estão muito encapsuladas.

Apesar da complexidade gerada pelas classes prontas de Django não serem uma imposição do framework, uma vez que ele tem suporte para outras funções, é muito comum a utilização delas pela comunidade e pelos documentos que mostram exemplos de aplicação em projetos em seu formato.

Destaque-se no Flask a possibilidade de codificar de forma minimalista, como a própria documentação indica, sem a necessidade de escrever muito. Entretanto é necessário ressaltar que a “base de conhecimento do Flask” não é tão robusta quanto a de um framework como o Django que possui muitas soluções incorporadas em sua base.

Já a reutilização de código é evidente na extensão de formato (design) dos templates em ambas as linguagens. O sistema de rotas do Flask é muito mais intuitivo. Em contrapartida o número de bibliotecas disponíveis para Django é muito maior, portanto se a intenção do programador for reutilizar o maior número de recursos possíveis para sua aplicação ele encontrará menos percalços ao desenvolver utilizando o Django.

Mas, é importante notar que os requisitos definidos para os projetos da pesquisa se aproximam mais da realidade da academia do que do mercado, no sentido de que o reuso desmedido pode tornar a comunicação com sistema um pouco confusa, pois muitas ações ocorrem implicitamente em códigos prontos e isso pode ser um problema para quem está iniciando.

A integração com banco de dados foi a parte mais complexa em Flask, pois exigiu experiência mais avançada com o sistema de gerenciamento de banco de dados. No entanto, quanto a velocidade de desenvolvimento destaca-se que é possível ter um retorno muito mais rápido em Flask, pois é muito simples associar o sistema de rotas aos templates

Em relação à comunidade Django tem um número maior de aplicações que rodam em sua estrutura. Foi muito mais rápido encontrar soluções nos fóruns para problemas que surgiram durante a pesquisa. Isso impacta, também, no processo de escalabilidade, pois à

medida que o sistema crescia era mais difícil manter o sistema em Flask organizado, pois ele oferece liberdade ao desenvolvedor para aplicar seu ambiente da forma que achar mais conveniente, por conseguinte, não existe um padrão específico definido para ele.

Já o Django, com sua estrutura MVT foi preparado desde o início para comportar novos componentes no sistema. O tempo que gasto na configuração do projeto é recompensado com a possibilidade de gerenciar de forma mais clara o desenvolvimento, os testes e a manutenção do código.

No entanto, é necessário reforçar que se, tanto o Django quanto o Flask, forem usados de forma inconsistente, além de não ter todo seu potencial explorado, podem não atingir os objetivos para os quais foram aplicados. Estas ferramentas acumulam uma grande quantidade de dados que se desenvolvem periodicamente bem como a linguagem em que são baseados.

A base utilizada em ambos é muito similar, mas à medida que se avançava as diferenças entre ambos ficavam mais evidentes, mas a proposta era de fato manter um tempo limite para adequar os resultados ao objetivo geral e no parâmetro prazo o Flask permitiu mais produtividade em menos tempo para atingir os recursos iniciais.

Esse resultado influenciou, também, no parâmetro desempenho, pois foram desenvolvidas funções adicionais no Flask, inclusive a integração com jquery nas imagens do formulário de produtos, ou seja, o número de tarefas desenvolvidas foi maior que no Django que só permitiu (dentro do prazo estabelecido) o cumprimento dos requisitos mínimos ilustrados na modelagem.

Em resumo, para o contexto da Universidade o Flask se adapta melhor aos requisitos de prazo e desempenho. Além disso, ele facilita a aprendizagem, pois permite reconhecer e integrar os conceitos estudados em Python que é sua linguagem base. Portanto, é uma opção melhor para se trabalhar em disciplinas de caráter introdutório e prático, pois deixa bem clara a relação entre linguagem de programação, framework e sistemas Web. Além disso, sua utilização facilitaria bastante a entrada em frameworks mais robustos como o Django, proporcionando aos alunos condições de competitividade no mercado de trabalho.

O Django apesar de poder ser usado em aplicações mais simples possui uma série de recursos que podem ser melhor aproveitados em projetos de grande porte, principalmente nos

projetos cujo foco é a qualidade e não, especificamente, a aprendizagem conforme mostra a análise apresentada nos quadros.

Quadro 3 - Análise da qualidade aplicada ao Flask

ATRIBUTOS DE QUALIDADE	CRITÉRIO AVALIADO	PESO
FUNCIONALIDADE	Adequação	2
	Acurácia	3
	Interoperabilidade	0
	Segurança de acesso	1
	Conformidade	0
CONFIABILIDADE	Maturidade	1
	Tolerância a falhas	1
	Recuperabilidade	3
	Conformidade	0
USABILIDADE	Inteligibilidade	1
	Apreensibilidade	1
	Operacionalidade	1
	Atratividade	0
	Conformidade	0
EFICIÊNCIA	Comportamento em relação ao tempo	3
	Utilização de recursos	3
	Conformidade	0
MANUTENIBILIDADE	Analísabilidade	3
	Modificabilidade	1
	Estabilidade	3
	Testabilidade	0
	Conformidade	0
	Adaptabilidade	3

PORTABILIDADE	Capacidade para ser instalado	3
	Coexistência	3
	Capacidade para substituir	3
	Conformidade	0

Fonte: Os autores (2018). Baseado nos critérios da NBR ISO/IEC 9126-1:2003

O peso 2 em adequação foi dado, pois no sistema desenvolvido em Flask o conjunto de funções utilizadas para concretizar as tarefas foram implementados “na mão” foi um processo trabalhoso, mas em compensação o sistema de rotas é muito simples então foi possível integrar rapidamente os templates criados com as funções desenvolvidas.

O peso 3 em acurácia foi dado, pois todos os resultados obtidos em cada operação implementada ocorreram da forma que era esperado. Já para segurança de acesso foi dado o peso 1, pois por padrão o Flask não oferece o tratamento de segurança em seus módulos de acesso ao sistema. O mesmo peso também foi dado para maturidade e tratamento de falhas, pois o tratamento de erros do Flask não é padronizado, para tratar melhor esse requisito é necessário ter um conhecimento mais aprofundado em Python, pois precisa ser feito “na mão”.

Apesar disso, sempre que houve alguma falha durante os processamentos de requisições testados no Flask não houve perda de dados e o sistema voltou rapidamente ao seu estado operável quando as falhas foram reparadas, por isso o peso dado a recuperabilidade foi dado o peso 3.

As funções nativas do Flask não agregaram valor algum aos módulos do Bootstrap, por isso os atributos relacionados a usabilidade receberam peso 1. O tempo de resposta para cada função foi adequado e apropriado em relação a quantidade de recursos utilizados, por isso o peso dado foi 3 para comportamento em relação ao tempo e para utilização de recursos.

Embora o Flask não possua um módulo de tratamento de falhas padronizado para implementação foi fácil diagnosticar a causa das falhas, pois a maioria estava relacionada ao sistema de rotas e ele de fácil compreensão (intuitivo), por isso o peso dado para

analisabilidade foi 3. No entanto, ficou claro que a medida que o sistema crescia ficava mais difícil gerenciar as modificações no sistema para comportar tarefas mais complexas (como a integração com o banco de dados) sem alterar a lógica dos módulos que já haviam sido criados, por isso o peso dado a modificabilidade foi 1. Apesar disso, o software, no contexto aplicado, se manteve estável, por isso o peso dado a estabilidade foi 3.

No ambiente de desenvolvimento todas as características de portabilidade (adaptabilidade, capacidade para ser instalado, coexistência e capacidade para substituir) foram satisfeitas principalmente porque o Flask é muito flexível, portanto o peso atribuído para elas foi 3.

Quadro 4 – Análise dos atributos de qualidade em Django

ATRIBUTOS DE QUALIDADE	CRITÉRIO AVALIADO	PESO
FUNCIONALIDADE	Adequação	3
	Acurácia	3
	Interoperabilidade	0
	Segurança de acesso	3
	Conformidade	0
CONFIABILIDADE	Maturidade	3
	Tolerância a falhas	3
	Recuperabilidade	1
	Conformidade	0
USABILIDADE	Inteligibilidade	3
	Apreensibilidade	3
	Operacionalidade	3
	Atratividade	0
	Conformidade	0
EFICIÊNCIA	Comportamento em relação ao tempo	3
	Utilização de recursos	3
	Conformidade	0

MANUTENIBILIDADE	Analisabilidade	1
	Modificabilidade	3
	Estabilidade	3
	Testabilidade	0
	Conformidade	0
PORTABILIDADE	Adaptabilidade	2
	Capacidade para ser instalado	2
	Coexistência	2
	Capacidade para substituir	2
	Conformidade	0

Fonte: Os autores (2018)

No Django, o peso 3 em adequação foi atribuído, pois no sistema desenvolvido as CBVs e funções utilizadas para concretizar as tarefas foram implementadas de forma automática. O sistema de rotas não é tão intuitivo quanto no Flask, mas funcionaram corretamente.

Também foi atribuído o peso 3 em acurácia, pois todos os resultados obtidos em cada operação implementada ocorreram da forma que era esperado. Para segurança de acesso também foi atribuído o peso 3, pois por padrão o Django oferece o tratamento de segurança em seus módulos. O mesmo peso também foi dado para maturidade e tratamento de falhas, pois o tratamento de erros é descrito na documentação das CBVs e também são incorporados de forma automática ao código.

Mas, diferente do que ocorreu com Flask, no Django sempre que houve alguma falha durante os processamentos de requisições foi necessário verificar a documentação e refazer alguns testes em versões diferentes e o sistema não voltou rapidamente ao seu estado operável quando as falhas foram reparadas, por isso o peso dado a recuperabilidade foi 1.

Todos os módulos inseridos no template do Django com bootstrap foram tratados por funções nativas do Django, por isso foi dado 3 em todos os atributos de usabilidade. O tempo de resposta para cada função foi adequado e apropriado em relação a quantidade de recursos

utilizados, por isso o peso dado foi 3 para comportamento em relação ao tempo e para utilização de recursos.

Em Django, não foi tão simples diagnosticar a causa das falhas, pois a maioria estava relacionada ao funcionamento das CBVs e elas são implementadas de forma automática o que resultou em um gasto de tempo maior para entender como cada resposta funcionava, por isso o peso dado para analisabilidade foi 1. No entanto, ficou claro que a medida que o sistema crescia não era necessário aplicar soluções complexas para gerenciar as modificações no sistema. A integração com o banco de dados, por exemplo, foi mais simples do que em Flask e a lógica do SQL foi completamente encapsulada pelo Django, sem alterar a lógica dos módulos que já haviam sido criados, por isso o peso dado a modificabilidade foi 3. Além disso, o software, no contexto aplicado, se manteve estável, por isso o peso dado a estabilidade foi 3.

No ambiente de desenvolvimento todas as características de portabilidade (adaptabilidade, capacidade para ser instalado, coexistência e capacidade para substituir) foram satisfeitas, pois o processo foi facilitado pelo uso de máquinas virtuais. No entanto, ao transitar entre versões diferentes foi difícil atualizar alguns módulos, por isso o peso atribuído para elas foi 2.

O resultado da análise (39 – Flask / 46 - Django) mostra que no parâmetro qualidade o Django é superior ao Flask. Em geral, porque “obriga” o desenvolvedor a determinar características consideradas como boas práticas na documentação. Os detalhes de como a análise foi realizada foram descritos na documentação do sistema.

4.2 SOBRE AS FERRAMENTAS DE DESENVOLVIMENTO

As ferramentas disponíveis para interface se integram bem com o ambos os frameworks (Bootstrap versão 3 e 4). As ferramentas de desenvolvimento utilizadas para ambos também se integraram bem. O que dificultou o desenvolvimento no caso de Django foram as configurações relacionadas a versão dos *plugins* instalados no VS Code.

4.3 LIMITAÇÕES DO TRABALHO

O tempo utilizado para programação nos objetos de estudo e a língua () em que estavam a maioria dos documentos utilizados como base teórica não permitiram explorar funções mais avançadas. Existem muitas bibliotecas prontas para utilizar em Django, mas é comum encontrar bibliotecas que não são mantidas e, por conseguinte, desatualizadas. As versões das ferramentas utilizadas na pesquisa são equiparáveis, mas a dificuldade de conciliar os recursos do Django com a versão utilizada na pesquisa não permitiu incluir mais funções no sistema de gerenciamento de produtos desenvolvido nele.

O sistema de gerenciamento de produtos em sua primeira etapa de desenvolvimento, cumpre sua função e já possibilita a definição de um roteiro lógico que auxilie os alunos na escolha da ferramenta para utilizar em disciplinas práticas, mas os módulos implementados e as ações resultantes ainda podem ser testados em outras ferramentas. .

4.4 TRABALHOS FUTUROS

Percebe-se que a aplicação do conceito de desenvolvimento é muito mais abrangente do que se costuma presumir, pois não abarca somente o ambiente virtual, envolve também a prática no mundo real e neste, linhas de código, requisições, e dados como um todo são apenas símbolos, porque resultam de atividades de análise, projeto e gerência de sistemas de informação que exigem uma carga de trabalho dos desenvolvedores

Como o estudo foi voltado para análise das características relacionadas ao desenvolvimento em si, com base no conhecimento já adquirido nas disciplinas realizadas, não foi realizada uma pesquisa de campo com os alunos da faculdade (a aplicabilidade nesse caso dependeria da oferta das disciplinas). Em trabalhos futuros é possível aplicar as ferramentas estudadas em turmas específicas para avaliar o progresso dos alunos em cada ferramenta.

Partindo deste Trabalho de Conclusão de Curso, é possível avaliar futuramente os parâmetros estudados em um ambiente de produção específico, pois toda organização que

oferece algum tipo de serviço automatizado faz uso (direto ou indireto) deste tipo de tecnologia. Baseado nessa necessidade considera-se que as funcionalidades que serão criadas posteriormente tornarão o ambiente ainda mais completo e poderão ser analisados em produção com a equipe de desenvolvimento de uma empresa por exemplo.

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. 2003. **NBR ISO/IEC 9126-1**

Disponível em: <

https://aplicacoes.mds.gov.br/sagirms/simulacao/sum_executivo/pdf/fichatecnica_21.pdf>.

Acesso em: jun. 2018.

ASTAH. Astah - Software Design Tools for Agile teams with UML, ER Diagram ... 2018.

Disponível em: <<http://astah.net/>>. Acesso em: mar. 2018

BORGES, Luiz Eduardo. **Python para desenvolvedores**. 2. ed. Rio de Janeiro: edição do autor, 2010. Disponível em: <

https://ark4n.files.wordpress.com/2010/01/Python_para_desenvolvedores_2ed.pdf>. Acesso em: 23 nov. 2018.

BOTTLE. **Bottle**: Python Web Framework. Bottlepy.org, 2011. Disponível em:

<<https://bottlepy.org/docs/dev/>>. Acesso em: dez. 2019

CASS, Stephen; BULUSU, Parthasaradhi. **Interactive**: The Top Programming Languages 2018. Disponível em: <<https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018>>. Acesso em: 23 nov. 2018.

CORRÊA, Fernando; ANICHE, Maurício. **UML Introdução**: Modelagem de soluções.

Alura, 2018. Disponível em: <<https://www.alura.com.br/>>. Acesso em:

DJANGO SOFTWARE FOUNDATION. **Django documentation**: release 1.0. 2017.

Disponível em: < <https://docs.Djangoproject.com/pt-br/2.1/releases/1.0/>>. Acesso em: 23 nov. 2018

DJANGO SOFTWARE FOUNDATION. **Documentation**: escrevendo seu primeiro pp em

Django, parte 1. 2018. Disponível em: < <https://docs.Djangoproject.com/pt-br/2.1/intro/tutorial01/> >. Acesso em: 23 nov. 2018

DOWNEY, Allen; ELKNER, Jeff; MEYERS, Chris. **Aprenda computação com Python 3.0**. 2009. Disponível em: <

<https://chevitarese.files.wordpress.com/2009/09/aprendacomputaocomPython3k.pdf> >.

Acesso em: 23 nov. 2018

GAMMA, Erich et al. **Padrões de projeto**: soluções reutilizáveis de software orientado a objetos. Porto Alegre: bookman, 2007.

GUEDES, Gilleanes T. A. **UML 2**: uma abordagem prática. 2.ed. São Paulo: NOVATEC, 2011.

GUDWIN, Ricardo R. Engenharia de software: uma visão prática. 2. ed. DCA-FEEC-UNICAMP, 2015. Disponível em:

<<http://faculty.dca.fee.unicamp.br/gudwin/sites/faculty.dca.fee.unicamp.br/gudwin/files/ea975/ESUVP2.pdf>>. Acesso em:

LACERDA, Ivan Max Freire de. **Programador WEB**: um guia para programação e manipulação de banco de dados. Rio de Janeiro: SENAC, 2014.

MARQUES, Luan. **Desenvolvedor WEB com Flask**: Flask parte I. Disponível em:

<<https://www.alura.com.br/>>. Acesso em: dez. 2018.

MICROSOFT. **Visual Studio Code**: Code Editing. Redefined. 2018. Disponível em:

<<https://code.visualstudio.com/>>. Acesso em:

MOLINARI, Willian. **Desconstruindo a WEB**: as tecnologias por trás de uma requisição. São Paulo: Casa do código, 2016.

PIMENTEL, Fábio. **HTTP**: Entendendo a web por baixo dos panos. Disponível em:

<<https://www.alura.com.br/>>. Acesso em: dez. 2018.

PRESSMAN, Roger S. **Engenharia de software**: uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2011.

PUC-RIO. **Frameworks**: conceitos gerais. 2006. Disponível em: <https://www.maxwell.vrac.puc-rio.br/8623/8623_3.PDF>. Acesso em:

RAMOS, Vinícius. **Desenvolvimento WEB com Python e Django**. Python Academy, 2018. Disponível em: <<https://Pythonacademy.com.br/assets/ebooks/desenvolvimento-web-com-Python-e-Django/desenvolvimento-web-com-Python-e-Django.pdf>>. Acesso em: dez. 2018.

ROCHA, Bruno César. **What the Flask? Pt-1 Introdução ao desenvolvimento web com Python**. Disponível em: <http://Pythonclub.com.br/what-the-Flask-pt-1-introducao-ao-desenvolvimento-web-com-Python.html#o_que_e_Flask>. Acesso em: nov. 2018.

RONACHER, Armin. **Documentação Flask**. Disponível em: <<https://Flask-ptbr.readthedocs.io/en/latest/foreword.html>>. Acesso em: nov. 2018.

SEABRA, Rodrigo Duarte; DRUMMOND, Isabela Neves; GOMES, Fernando Coelho. Análise comparativa de linguagens de programação a partir de problemas clássicos da computação. **Revista de Sistemas e Computação**, Salvador, v. 8, n. 1, p. 56-76, jan./jun. 2018. Disponível em: <<https://revistas.unifacs.br/index.php/rsc/article/view/5133>>. Acesso em: 25 nov. 2018

SEBESTA, Robert W. **Conceitos de linguagens de programação**. 9. ed. Porto Alegre: Bookman, 2011.

SOUSA, Álan Crístopher e. **Curso básico de Python 3**. Divinópolis, 2017. Disponível em: <<https://acristoffers.me/assets/Python3.pdf>>. Acesso em: nov. 2018.

STEPPAT, Nico. **Python 3 parte 1**: Introdução à nova versão da linguagem. Alura, 2018. Disponível em: <<https://www.alura.com.br/>>. Acesso em: dez. 2018.

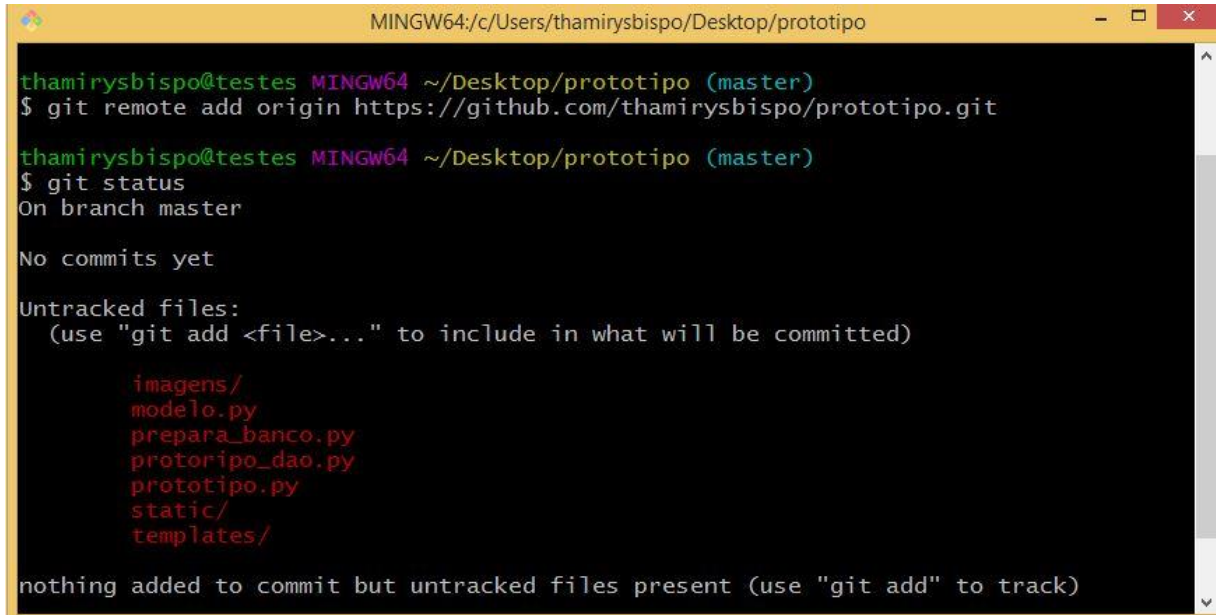
TORNADO. **Tornado WEB server**. 2018. Disponível em: <<https://www.tornadoweb.org/en/stable/>>. Acesso em:

WSGI. **What is WSGI?** WSGI.org, 2011. Disponível em: <<https://wsgi.readthedocs.io/en/latest/what.html>>. Acesso em: nov. 2018.

APÊNDICE A – COMMIT DOS ARQUIVOS DO PROTÓTIPO NO GITHUB

Os comandos necessários para armazenar os arquivos da pesquisa no GitHub estão representados nas imagens abaixo. O Git já estava instalado no computador.

Passo 1 – Abrir o GitBash na pasta do protótipo e executar o comando “git init”, adicionar o repositório de origem e executar o comando “git status” para verificar quais arquivos ainda não foram incluídos:



```

MINGW64:/c/Users/thamirysbispo/Desktop/prototipo
thamirysbispo@testes MINGW64 ~/Desktop/prototipo (master)
$ git remote add origin https://github.com/thamirysbispo/prototipo.git

thamirysbispo@testes MINGW64 ~/Desktop/prototipo (master)
$ git status
On branch master

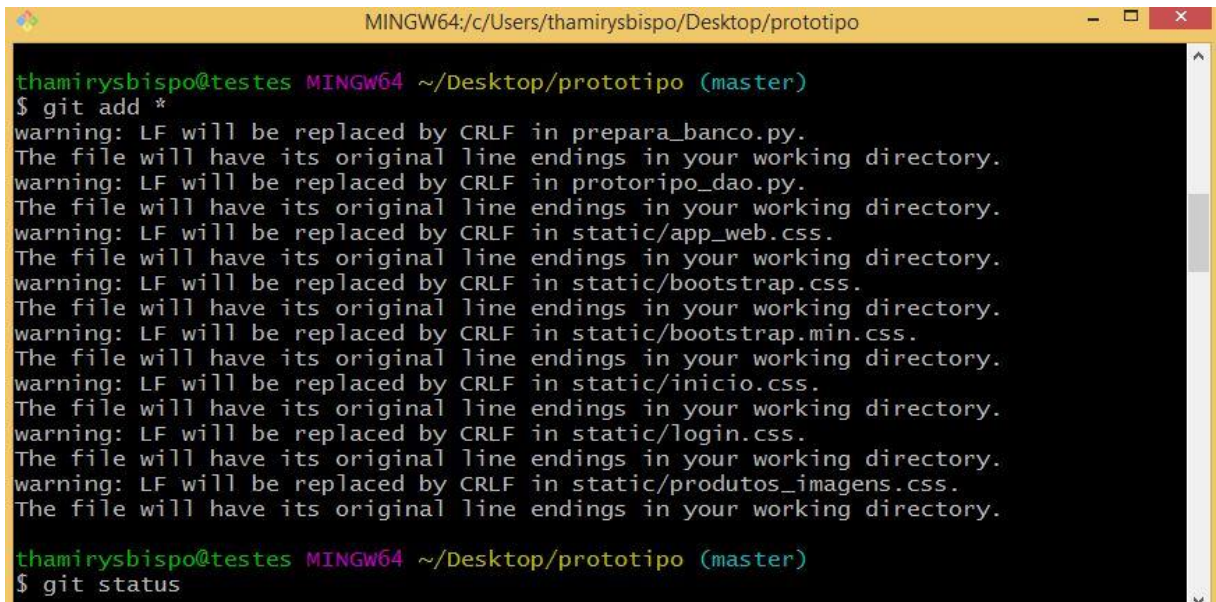
No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    imagens/
    modelo.py
    prepara_banco.py
    protoripo_dao.py
    prototipo.py
    static/
    templates/

nothing added to commit but untracked files present (use "git add" to track)
  
```

Passo 2 – Executar o comando “git add *” para adicionar os arquivos no git (nesse caso são todos):

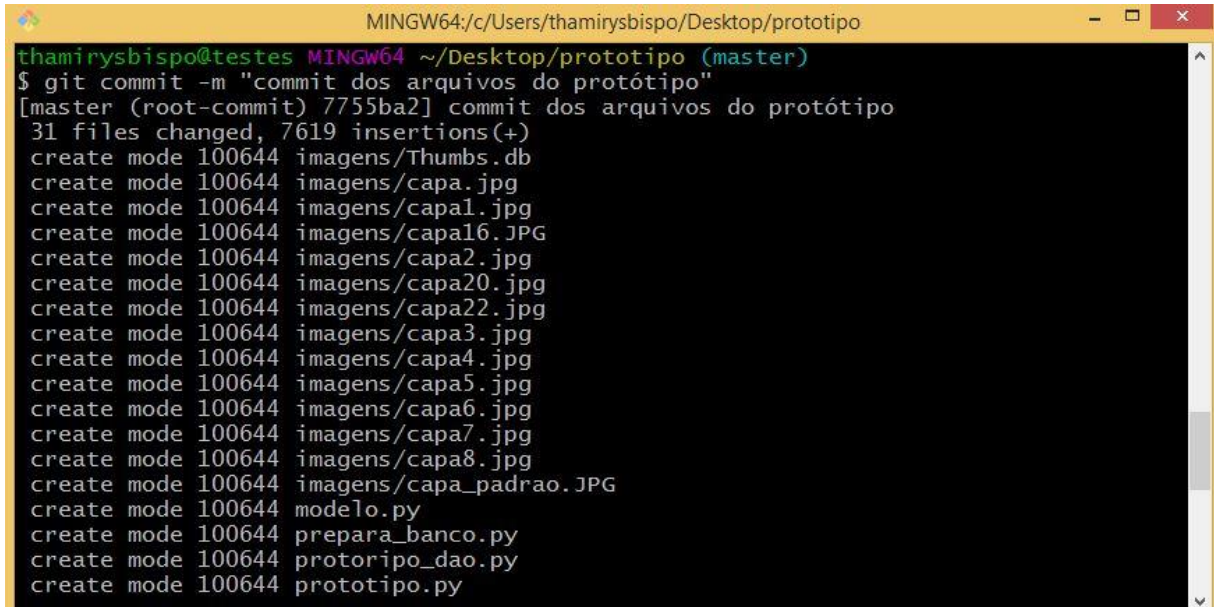


```

MINGW64:/c/Users/thamirysbispo/Desktop/prototipo
thamirysbispo@testes MINGW64 ~/Desktop/prototipo (master)
$ git add *
warning: LF will be replaced by CRLF in prepara_banco.py.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in protoripo_dao.py.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in static/app_web.css.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in static/bootstrap.css.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in static/bootstrap.min.css.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in static/inicio.css.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in static/login.css.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in static/produtos_imagens.css.
The file will have its original line endings in your working directory.

thamirysbispo@testes MINGW64 ~/Desktop/prototipo (master)
$ git status
  
```

Passo 3 – Executar o comando “git commit” para guardar os arquivos permanentemente:

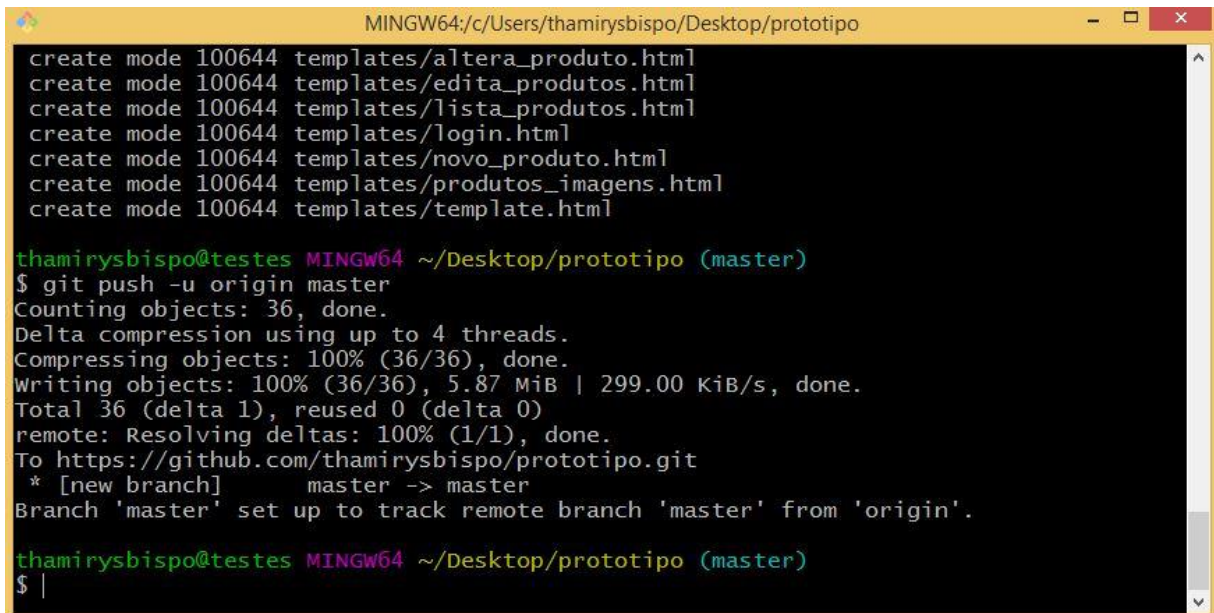


```

MINGW64:/c:/Users/thamirysbispo/Desktop/prototipo
thamirysbispo@testes MINGW64 ~/Desktop/prototipo (master)
$ git commit -m "commit dos arquivos do protótipo"
[master (root-commit) 7755ba2] commit dos arquivos do protótipo
31 files changed, 7619 insertions(+)
create mode 100644 imagens/Thumbs.db
create mode 100644 imagens/capa.jpg
create mode 100644 imagens/capa1.jpg
create mode 100644 imagens/capa16.JPG
create mode 100644 imagens/capa2.jpg
create mode 100644 imagens/capa20.jpg
create mode 100644 imagens/capa22.jpg
create mode 100644 imagens/capa3.jpg
create mode 100644 imagens/capa4.jpg
create mode 100644 imagens/capa5.jpg
create mode 100644 imagens/capa6.jpg
create mode 100644 imagens/capa7.jpg
create mode 100644 imagens/capa8.jpg
create mode 100644 imagens/capa_padrao.JPG
create mode 100644 modelo.py
create mode 100644 prepara_banco.py
create mode 100644 protoripo_dao.py
create mode 100644 prototipo.py

```

Passo 4 – Executar o comando “git push -u origin master” para enviar os arquivos ao GitHub:



```

MINGW64:/c:/Users/thamirysbispo/Desktop/prototipo
create mode 100644 templates/altera_produto.html
create mode 100644 templates/edita_produtos.html
create mode 100644 templates/lista_produtos.html
create mode 100644 templates/login.html
create mode 100644 templates/novo_produto.html
create mode 100644 templates/produtos_imagens.html
create mode 100644 templates/template.html

thamirysbispo@testes MINGW64 ~/Desktop/prototipo (master)
$ git push -u origin master
Counting objects: 36, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (36/36), done.
Writing objects: 100% (36/36), 5.87 MiB | 299.00 KiB/s, done.
Total 36 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/thamirysbispo/prototipo.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

thamirysbispo@testes MINGW64 ~/Desktop/prototipo (master)
$ |

```


Passo 5 – Verificar se os arquivos foram incluídos no repositório:








Protótipo desenvolvido em flask Edit

[Manage topics](#)

1 commit 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

 **ThamirysMSBispo** commit dos arquivos do protótipo Latest commit 7755ba2 9 minutes ago

 imagens	commit dos arquivos do protótipo	9 minutes ago
 static	commit dos arquivos do protótipo	9 minutes ago
 templates	commit dos arquivos do protótipo	9 minutes ago
 modelo.py	commit dos arquivos do protótipo	9 minutes ago
 prepara_banco.py	commit dos arquivos do protótipo	9 minutes ago
 protoripo_dao.py	commit dos arquivos do protótipo	9 minutes ago
 prototipo.py	commit dos arquivos do protótipo	9 minutes ago

Help people interested in this repository understand your project by adding a README.

Add a README