Capítulo

5

Desenvolvimento de sistemas Web orientado a reuso com Python, Django e Bootstrap

Cynthia Pinheiro Santiago, Nécio de Lima Veras, Anderson Passos de Aragão, Daniel Albuquerque Carvalho, Luciana Alves Amaral

Abstract

The constant growth of the Internet has caused a significant demand for WEB applications. In this scenario, developers seek to create solutions that deliver the required functionality but that also offer usability, security, scalability, among others. Web development frameworks, providing extensive reuse of architectures and libraries, emerged in this context to reduce the development time and complexity inherent in such systems. In this chapter, we present a complete Web solution with Python and the Django and Bootstrap frameworks through a step-by-step, from the installation of the required software, through the integration of the components to the publication in production.

Resumo

O constante crescimento da Internet tem ocasionado uma demanda significativa por aplicações Web. Neste cenário, desenvolvedores buscam criar soluções que entreguem as funcionalidades requeridas mas que também ofereçam usabilidade, segurança, escalabilidade, entre outros. Os frameworks de desenvolvimento Web, proporcionando extenso reuso de arquiteturas e bibliotecas, surgiram nesse contexto com a intenção de reduzir o tempo de desenvolvimento e a complexidade inerentes a tais sistemas. Neste capítulo, apresentamos uma solução Web completa com Python e os frameworks Django e Bootstrap através de um passo-a-passo, desde a instalação dos softwares necessários, passando pela integração dos componentes até a publicação em produção.

5.1. Introdução

O crescimento global das empresas e suas constantes demandas por novas tecnologias fazem com que o desenvolvimento de software esteja em contínua mudança, especialmente quando o funcionamento do software é direcionado para o ambiente da Internet. Somado

a isso, o desenvolvimento de sistemas para a Internet geralmente envolve partes interessadas (*stakeholders*) mais heterogêneas em comparação com a construção de software tradicional. Ainda assim, os modelos de processo usados para o desenvolvimento desses produtos são emprestados das abordagens tradicionais [Manhas 2017]. Isso não significa que as empresas usem um processo de desenvolvimento ideal, em vez disso, existem várias desvantagens apresentadas por metodologias inadequadas e não versáteis, o que torna o processo de criação um desafio.

A Engenharia de Software (ES) é uma área da Ciência da Computação que abrange todos os aspectos do desenvolvimento profissional de software. Estes aspectos englobam desde a sua concepção até o encerramento [Sommerville 2016], passando pelos processos de planejamento, projeto, codificação e testes. Nos últimos anos, a ideia da oferta de "software como serviço" foi proposta, modificando a forma com que os usuários utilizam a internet e acessam seus dados (armazenados em nuvem). A partir disso, ocorreram mudanças significativas na maneira como o software é distribuído entre os usuários, impactando na forma como os sistemas são desenvolvidos, especialmente, os sistemas para a Web (*World Wide Web*), fazendo emergir tecnologias, infraestruturas e novas metodologias para o desenvolvimento Web.

O desenvolvimento de aplicativos Web está no auge devido ao avanço das tendências tecnológicas e à constante dependência da Internet. Em tempos de isolamento social causado pela pandemia do novo Coronavírus, essa dependência é muito mais perceptível para a maioria das pessoas. Como resultado das necessidades dos desenvolvedores, novas metodologias de desenvolvimento são propostas [Molina-Ríos and Pedreira-Souto 2020]. As equipes de produção do software não constroem um novo produto a partir de um projeto "em branco", elas o programam fazendo reúso de componentes e de outros programas (*frameworks*[1]). Neste sentido, a ES incorpora conceitos de reusabilidade de software durante o desenvolvimento de novos produtos, relacionados com a montagem de partes do sistema a partir de softwares preexistentes [Sommerville 2016]. No ambiente Web, essa abordagem tornou-se dominante para a construção de softwares tento em vista a existência de restrições para tempo e custo, além de exigências maiores sobre qualidade e segurança para os produtos de software.

Diante disso, várias tecnologias para o desenvolvimento de produtos Web vem surgindo ao longo dos anos, destacando elementos como produtividade e velocidade na construção dos produtos, segurança, performance do produto e, obviamente, reuso de componentes em diferentes projetos. Os produtos Web comumente são divididos em dois subprodutos tecnológicos: *front-end* e *back-end* [Ghimire 2020]. Tecnologias *back-end* estão ligadas à linguagens de programação, todavia, preocupam-se com a programação lógica dos sistemas Web e influenciam na forma como os dados são armazenados, acessados e servidos a partir dos "servidores". Por outro lado, tecnologias *front-end* concentram a parte estética e de exibição de conteúdos, também conhecida como desenvolvimento do lado do cliente. Neste capítulo abordaremos a linguagem Python apoiada pelo *framework* Django como principais tecnologias *back-end* e o *framework* Bootstrap como principal tecnologia *front-end*.

[1]Um framework é um conjunto de bibliotecas e ferramentas padronizadas para uma linguagem de programação específica [Ghimire 2020]

Nas seções posteriores serão detalhados alguns pontos acima citados, como Engenharia de software Orientada à Reúso, Desenvolvimento de Software para o ambiente da Web por meio das tecnologias Python, Django e Bootstrap. Além disso, será apresentado um projeto ilustrando o uso das tecnologias detalhadas.

5.2. Engenharia de Software Orientada a Reúso

O desenvolvimento de softwares, de uma forma geral, é tipicamente conhecido por ser um processo caro e lento. A pressão que acompanha todas as etapas de construção, bem como a busca por redução de custos e maior retorno do investimento, associados à redução do tempo disponível para o desenvolvimento podem fazer com que o resultado final e a qualidade do produto fiquem aquém do desejado. Neste sentido, a reutilização de software é uma solução viável, que pode ajudar a lidar com muitos destes desafios. Apesar da ideia de reúso de software ter sido proposta como estratégia de desenvolvimento no final dos anos 60, só a partir dos anos 2000 se tornou realidade. Este fato se deu como resposta às exigências cada vez mais frequentes dos *stakeholders* por menores custos de produção e de manutenção, entregas mais rápidas e softwares de maior qualidade.

Engenharia de software baseada em reúso é uma estratégia onde todo o processo de desenvolvimento de um sistema é orientado a maximizar o reúso de ativos de software já existentes, desde sistemas completos, assim como componentes, *frameworks*, classes, funções e até mesmo conceitos [Sommerville 2016]. A reutilização de software também pode ser definida como "o uso dos conhecimentos e artefatos de engenharia existentes para construir novos sistemas de software" [Frakes and Fox 1996]. Segundo [Sommerville 2016], esta abordagem apresenta algumas reconhecidas vantagens como: (i) os ativos de software reusados (anteriormente experimentados e testados em sistemas em funcionamento) tendem a ser mais confiáveis do que os que são desenvolvidos a partir de um "projeto em branco"; (ii) como o custo de um ativo de software existente já é conhecido e consolidado, isso reduz a margem de erro na estimativa dos custos do projeto; (iii) em vez de repetir o mesmo trabalho em vários sistemas, especialistas desenvolvem ativos de software passíveis de reúso que encapsulem seu conhecimento e que sejam passíveis de utilização inclusive por outros desenvolvedores; (iv) alguns padrões, como os de interface de usuário, podem ser implementados como um conjunto de componentes reusáveis, resultando em menos erros por parte dos usuários uma vez que a interface já lhes é familiar; (v) o reúso de ativos de software em geral acelera a produção do sistema, já que reduz seu tempo de desenvolvimento e validação.

Por outro lado, também são conhecidos obstáculos com relação à adoção do reúso de software como: (i) resistência à modificação de mentalidade dos próprios desenvolvedores quanto ao reuso, (ii) alta curva de aprendizagem de alguns ativos de software a serem reusados, (iii) a dificuldade em manter um repositório de componentes reutilizáveis no que se refere à classificação, catalogação e pesquisa de componentes (às vezes, pode ser mais custoso encontrar um componente que reutilizá-lo) além de outras barreiras detectadas, analisadas e apresentadas em [Almeida 2019].

Ao longo do tempo, muitas técnicas foram desenvolvidas para oferecer suporte ao reúso de software em suas mais distintas granularidades, desde sistemas completos até funções. Neste aspecto, dentre as abordagens de reúso de software disponíveis está a que

faz uso de *frameworks* de aplicação que, segundo [Schmidt et al. 2004], podem ser definidos como um conjunto integrado de artefatos de software (tais como classes, objetos e componentes) modulares e/ou extensíveis que colaboram para fornecer uma arquitetura reusável. Estendendo esse conceito, um *framework* de aplicação Web pode ser entendido como uma coleção de pacotes e módulos compostos por código padronizado e pré-escrito que suporta o desenvolvimento de aplicações Web, tornando o desenvolvimento mais rápido e fácil, e fazendo com que os programas que o utilizem tornem-se mais confiáveis e escaláveis. Tais *frameworks* já possuem componentes internos que "configuram" o projeto, para que o desenvolvedor tenha menos trabalho repetitivo.

Para este tipo de desenvolvimento, os *frameworks* tornaram-se uma parte essencial, já que os padrões Web estão sempre se sofisticando e, consequentemente, a complexidade da tecnologia necessária para atendê-los está cada vez maior, tornando praticamente proibitivos o custo e o tempo necessário para o desenvolvimento de uma aplicação a partir de um "projeto em branco". Os *frameworks* de aplicações Web geralmente têm sua arquitetura baseada no padrão MVC (*Model-View-Controller* ou Modelo-Visão-Controlador, em português) [Gamma et al. 1995] ou em suas variações. Isto permite separar as representações internas dos dados do modo como estes dados são apresentadas para os usuários. Além disso, os *frameworks* Web oferecem suporte à apresentação dos dados de maneiras diferentes e interagem com cada uma dessas apresentações. Quando os dados são modificados por meio de uma das apresentações, o modelo de sistema é alterado e os controladores associados a cada visão atualizam sua apresentação.

A interação do usuário com uma aplicação Web que usa o padrão MVC (Figura 5.1) segue um ciclo natural: "o usuário executa uma requisição (*request*) e, em resposta, a aplicação consulta e/ou altera seu modelo de dados, fornecendo uma visualização atualizada com uma resposta (*response*) ao usuário. Isto é um ajuste muito conveniente para aplicações Web, onde há uma série de requisições e respostas HTTP" [Freeman 2017].

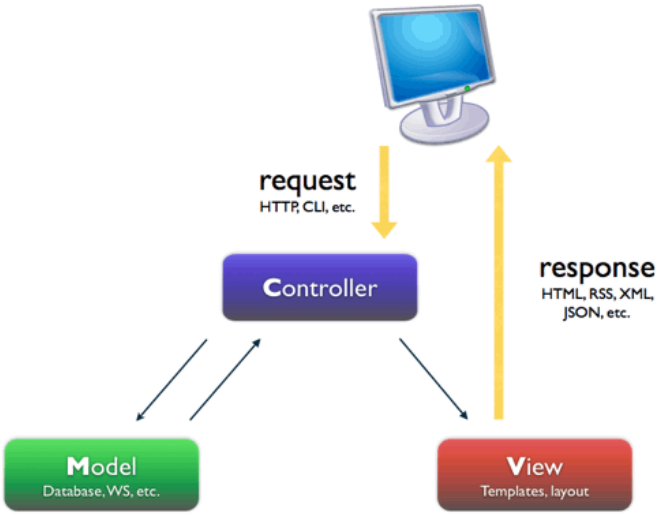

Figura 5.1. Padrão MVC, adaptado de [Dragos-Paul and Altar 2014]

Além disso, a maioria dos *frameworks* Web disponíveis oferece minimamente os

seguintes recursos: (i) suporte para a autenticação de usuário e controle de acesso; (ii) definição de *templates*[2] de páginas Web preenchidos dinamicamente; (iii) interface abstrata para manipulação de banco de dados; (iv) criação e gerenciamento de sessões do usuário; (v) suporte a tecnologias que permitem interação com o usuário (p.e. AJAX[3]) . Ou seja, eles dão apoio à resolução de problemas recorrentes utilizando uma abordagem genérica, permitindo ao desenvolvedor dedicar mais tempo à resolução dos problemas em si, e não em reescrever software que já existe. É interessante frisar que um mesmo sistema Web pode incorporar vários *frameworks*, de forma que cada um deles é projetado para apoiar o desenvolvimento de parte da aplicação.

5.3. Desenvolvimento Web

O termo "Desenvolvimento Web" pode ser definido como a construção, criação e manutenção de sites na Internet ou em uma intranet. Normalmente, o desenvolvimento na Web envolve um *front-end* (*client-side*), código que interage com o cliente através de um navegador, e um *back-end* (*server-side*), código executado no servidor Web, que encapsula a lógica de negócios da aplicação e a interação com o banco de dados. O padrão MVC, citado na Seção 5.2, favorece essa divisão: os desenvolvedores *front-end* concentram-se na camada de Visão (*View*) e os desenvolvedores *back-end* concentram-se na camada de Modelo (*Model*).

Frameworks podem ser utilizados tanto no *front-end* como no *back-end*. *Frameworks* e bibliotecas fazem parte da pilha de tecnologias Web: uma combinação de software, aplicativos, linguagens de programação e ferramentas que são construídas umas sobre as outras para criar um site. Uma pilha de tecnologia aplicável ao desenvolvimento Web pode ser visto na Figura 5.2. Uma biblioteca é uma coleção de ferramentas e recursos específicos que pode ser adicionada à aplicação Web a fim de obter funcionalidades. Diferentemente de um *framework*, uma biblioteca não oferece uma estrutura do ponto de vista arquitetural, mas implementa diferentes comportamentos e ações.

Nas subseções seguintes, apresentamos mais detalhes sobre cada uma das frentes do desenvolvimento de um sistema Web e mencionamos de que forma *frameworks* e bibliotecas podem ser utilizados neste sistema.

5.3.1. *Front-end*

Tudo o que é visto e "navegado" pelo usuário (identidade visual do site, menus, textos, imagens etc.) é criado e mantido pelo desenvolvedor *front-end*. Para isso, são escritos programas para vincular e estruturar os elementos visuais, adicionando interatividade com os usuários. Esses programas são executados através de um navegador (*browser*). No *front-end*, o desenvolvedor transforma a ideia de design da aplicação em realidade, concentrando-se no layout, design e interatividade da aplicação Web. Para tanto, linguagens como HTML (linguagem de marcação), CSS (linguagem de estilo) e JavaScript

[2]Um template é um documento apenas com a apresentação visual e instruções sobre onde e qual tipo de conteúdo deve entrar em cada parcela da apresentação.

[3]O AJAX (Asynchronous JavaScript and XML) é um conjunto de técnicas de desenvolvimento voltado para a Web que permite que aplicações trabalhem de modo assíncrono, processando qualquer requisição ao servidor em segundo plano.[Holdener 2008]

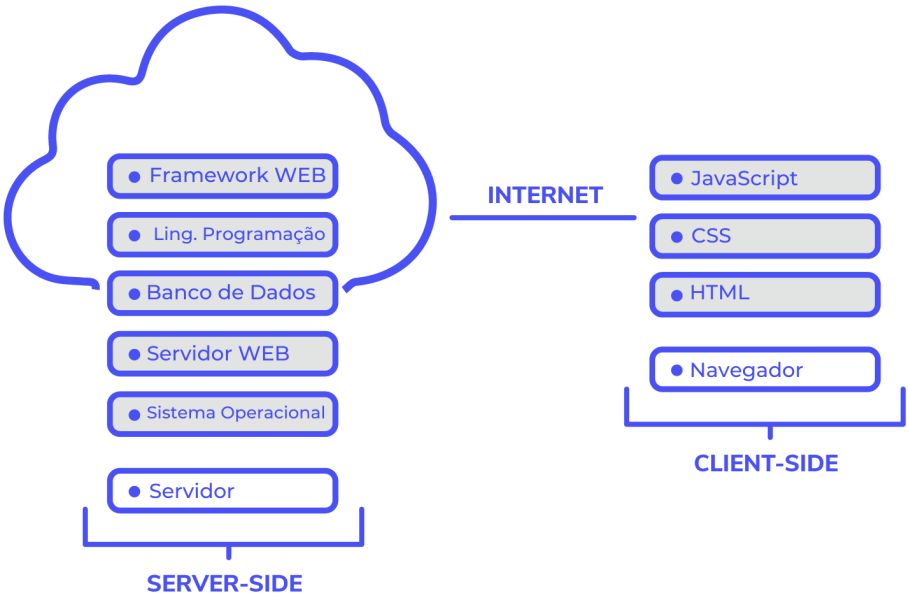

Figura 5.2. Pilha de tecnologias, adaptada de [Fawcett 2019]

(linguagem de script/programação) são utilizadas. A seguir, uma breve descrição de cada uma destas linguagens:

- HTML (*HyperText Markup Language*): é a linguagem de programação básica para desenvolvimento Web *front-end*. Fornece a estrutura para o conteúdo de um site e define palavras, títulos, parágrafos, imagens etc. O HTML consiste em um conjunto de *tags* pré-definidas, representando funções diferentes que "convertem" o conteúdo em um formato legível na tela.

- CSS (*Cascading Style Sheets*): é uma "folha de estilos" que descreve como os elementos HTML aparecerão em uma página da Web. Usa-se o CSS para controlar a apresentação, o estilo e a formatação dos elementos HTML em um site, configurando cores, bordas, imagens de fundo etc. Os arquivos CSS declaram um conjunto de regras, que definem propriedades e seus respectivos valores para os elementos HTML.

- JavaScript: refere-se ao controle do comportamento de uma página Web. É uma das linguagens de programação mais populares e difundidas no mundo. O uso do JavaScript torna os sites interativos, manipulando as especificidades dos elementos HTML e CSS. Com JavaScript, um usuário pode clicar em um botão e disparar uma ação, fazer o site rolar para a parte inferior ou exibir fotos aleatórias, por exemplo.

Na maioria dos casos, os *frameworks* para *front-end* são caixas de ferramentas ("*tool boxes*") que incluem diversas bibliotecas CSS e JavaScript. Entre os *frameworks* e

bibliotecas mais utilizados no *front-end* estão o Ember[4], React[5], Backbone[6], Vue[7], Angular[8] e Bootstrap[9].

O Bootstrap é o *framework front-end* utilizado no projeto ilustrativo deste capítulo. Desde 2013, ele se tornou um dos projetos mais populares na plataforma de compartilhamento de código GitHub[10]. Possui suporte mantido por uma comunidade ativa e um vasto ecossistema, incluindo modelos e extensões criados em torno dele. Lançado inicialmente pelo Twitter[11] para manter a consistência em seus projetos internos de Web design e desenvolvimento, o Bootstrap evoluiu e, desde o lançamento da versão 3, foi licenciado sob a licença MIT de código aberto [Bootstrap 2020a]. Mais detalhes sobre este *framework* na seção 5.5.

5.3.2. *Back-end*

O *back-end* consiste no servidor Web que hospeda o site, uma aplicação para executá-lo e um banco de dados para armazenar as informações. Como o nome sugere, o desenvolvedor *back-end* trabalha na parte de "trás" da aplicação, escrevendo programas que garantam que o servidor, a aplicação Web e o banco de dados funcionem bem juntos. Esse desenvolvedor precisa analisar quais são as necessidades de negócio dos *stakeholders* e fornecer soluções de programação eficientes e seguras. Para isso, são utilizadas uma variedade de linguagens do tipo "*server-side*", como PHP, Ruby, Java ou Python.

Além disso, dada a complexidade das tecnologias envolvidas no *back-end* e a quantidade de requisitos de qualidade que devem ser atendidos (p. e. desempenho, confiabilidade, segurança e disponibilidade do serviço), geralmente são utilizados *frameworks back-end* que auxiliam o desenvolvedor em muitos destes aspectos.

Uma das características comuns que a maioria dos *frameworks back-end* oferecem é uma abstração do banco de dados com a qual ele possa interagir, em lugar do próprio banco de dados (Figura 5.3). Isso é feito através de um Mapeamento Objeto-Relacional (do inglês, ORM: *Object-Relational Mapping*), definido como uma ferramenta que fornece uma metodologia e um mecanismo para sistemas orientados a objetos para armazenar dados de forma segura e por um longo período de tempo em um banco de dados, tendo controle transacional sobre eles, mas sendo expresso, se necessário, como objetos dentro da aplicação [O'Neil 2008].

O ORM libera o desenvolvedor da preocupação de conhecer a estrutura ou esquema do banco de dados. O acesso a dados é todo feito usando um modelo conceitual que reflete os objetos de negócios dos desenvolvedores, correlacionando dados em tabelas de um banco de dados relacional com classes em uma linguagem de programação. Assim, o desenvolvedor trabalha com conceitos da programação orientada a objetos para gerenciar os dados armazenados no banco [Dragos-Paul and Altar 2014].

[4]https://emberjs.com/
[5]https://reactjs.org/
[6]https://backbonejs.org/
[7]https://vuejs.org/
[8]https://angular.io/
[9]https://getbootstrap.com/
[10]https://github.com/
[11]https://twitter.com/

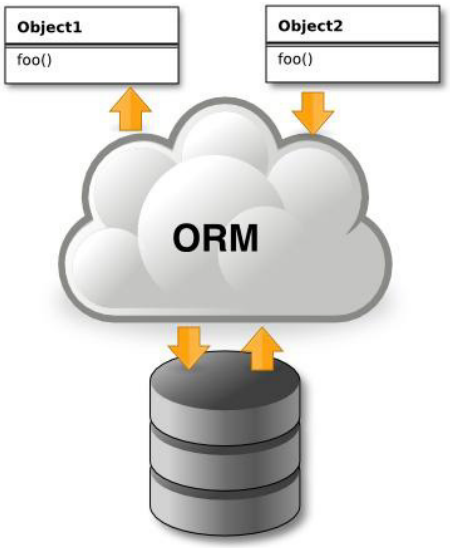

Figura 5.3. Mapeamento Objeto-Relacional (ORM) [Dragos-Paul and Altar 2014]

Um outro ponto em que os *frameworks back-end* auxiliam os desenvolvedores é em relação à segurança da aplicação. A vulnerabilidade de aplicações Web é a causa dos mais diversos tipos de ataques aos usuários. São muitos os pontos em que os desenvolvedores devem ficar atentos como: validação dos dados de entrada dos usuário, autenticação no sistema, gerenciamento de privilégios de acesso, acesso não autorizado à interface de gerenciamento, acesso a dados sensíveis no banco de dados, roubos de sessão, tipo de criptografia dos dados, manipulação indevida de parâmetros do sistema, ataques de negação de serviço, entre outros [Dragos-Paul and Altar 2014]. Por isso, muitos *frameworks back-end* disponibilizam aos desenvolvedores solução robustas em segurança já prontas que evitam em grande parte os inconvenientes mencionados.

Além disso, outra característica fornecida pela maioria dos *frameworks back-end* é o mapeamento de URLs (do inglês, *URL Mapping* ou *URL Routing*), que consiste em criar padrões de URL para a aplicação Web. O *framework* compara requisições HTTP com padrões pré-estabelecidos no sistema (*URL patterns*). Se forem compatíveis, estas requisições são encaminhadas para tratamento em sua função correspondente, que foi previamente mapeada à URL no *framework*. Por outro lado, após o tratamento das requisições, a resposta (*response*) correspondente é encaminhada para uma URL de saída, gerada depois que uma função específica é executada.

Atualmente, existem diversos *frameworks* para *back-end* disponíveis que implementam estas características. Alguns dos mais populares entre os desenvolvedores são: Spring[12], Express[13], Laravel[14], ASP.NET[15], Rails[16] e o Django[17].

[12]https://spring.io/
[13]https://expressjs.com/
[14]https://laravel.com/
[15]https://dotnet.microsoft.com/apps/aspnet
[16]https://rubyonrails.org/
[17]https://www.djangoproject.com/

O Django, *framework back-end* utilizado no projeto ilustrativo deste capítulo, é escrito em Python e vem com o conceito de "baterias incluídas", ou seja, ele dá suporte para a maior parte das necessidades de um desenvolvimento *server-side*, sendo altamente personalizável, escalável, possuindo uma extensa documentação e uma grande comunidade de desenvolvedores. Tornou-se um projeto de código aberto e foi publicado sob a licença BSD em 2005 [Foundation 2020a]. Mais detalhes sobre este *framework* na seção 5.4.

5.4. Python e Django

Python é uma linguagem de alto nível, interpretada, de tipagem forte e dinâmica, desenvolvida na década de 80 por Guido van Rossum no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação (CWI), na Holanda. O propósito geral do seu desenvolvimento foi facilitar a programação sem que o programador se preocupasse com o hardware. Tornou-se uma linguagem simples e muito poderosa, utilizada principalmente em áreas como *Data Science*, *Data Analytics*, Inteligência Artificial, *Deep Learning*, Desenvolvimento Web e aplicações de forma geral. Além disso, possui diversas bibliotecas que permitem o reúso de funcionalidades no desenvolvimento de software. Até o presente momento, são 246.267 bibliotecas disponíveis [Foundation 2020b].

A linguagem vêm ganhando popularidade equiparando-se com linguagens tradicionais como C++ e Java. Segundo o RedMonk, uma das maiores empresas que coletam estatísticas para programadores, Python está na segunda colocação no ranking de linguagens com maior popularidade [Stephens 2020]. É uma linguagem multiplataforma, o que garante portabilidade e compatibilidade em diversos sistemas operacionais. Além disso, Python tem código aberto, o que significa que qualquer pessoa pode ver e alterar sua codificação para fazer suas próprias bibliotecas. Suporta também interação com outras linguagens, permitindo que desenvolvedores possam utilizar ferramentas e/ou funcionalidades que não se encontram nativamente em Python, algumas escritas em C/C++, Java, C#, .NET, PHP, entre outras.

Python se propõe a trazer mais comodidade na hora de programar, por isso, diferentemente das demais linguagens, utiliza poucas palavras, o que implica em menos código. Quando começamos a programar em Python, aprendemos que a indentação é primordial, temos que ter um código organizado e limpo para funcionar, assim temos menos erros. Esta linguagem faz uso de palavras reservadas curtas da língua inglesa, facilitando assim, o entendimento e o aprendizado, além de deter de um grande poder de processamento (flexibilidade e simplicidade), fator que pode ser utilizado tanto por aqueles que estão entrando no mundo da programação, como também para os mais experientes.

Por estes diferenciais, diversas empresas optam por codificar uma parte ou mesmo a totalidade de seus produtos em Python. Alguns delas são: a Netflix, maior stream de vídeos, que utiliza Python para aumentar a segurança de seus dados, analisar alertas e gerar relatórios de dados; o Google, que o utiliza em projetos que demandam entregas rápidas e facilidade de implantação como o Google App Engine, uma plataforma de computação em nuvem para desenvolver e hospedar aplicativos em data centers gerenciados pelo Google; o Instagram, maior rede social de compartilhamento de fotos, que migrou suas aplicações para o Python; o Spotify, mais famosa stream de músicas, que usa o Python

para analisar dados e no back-end do aplicativo [Demchenko 2019].

Por outro lado e seguindo a popularidade desta linguagem surgiu o Django, um *framework* para criação de aplicações Web escrito em Python, criado em 2005 por um grupo de programadores do Lawrence Journal-World com a intenção de tornar mais rápido o desenvolvimento de aplicações Web. Este *framework* tornou-se conhecido por fornecer soluções para grande parte dos problemas tradicionais em desenvolvimentos Web, possuindo dezenas de tarefas comuns já prontas para serem reutilizadas, como por exemplo autenticação de usuário, administração de conteúdo, mapas de site, entre outras.

Django também ajuda a evitar erros de segurança comuns e proporciona escalabilidade aos sistemas, ou seja, consegue oferecer capacidade de expansão de um sistema sem perda do seu desempenho, como é o que acontece por exemplo com Mozilla Firefox, Pinterest e Instagram. Muitas empresas escolhem o Django por sua extrema versatilidade, sendo ele utilizado para criação de sistemas que vão desde gerenciamento de conteúdo até plataformas científicas.

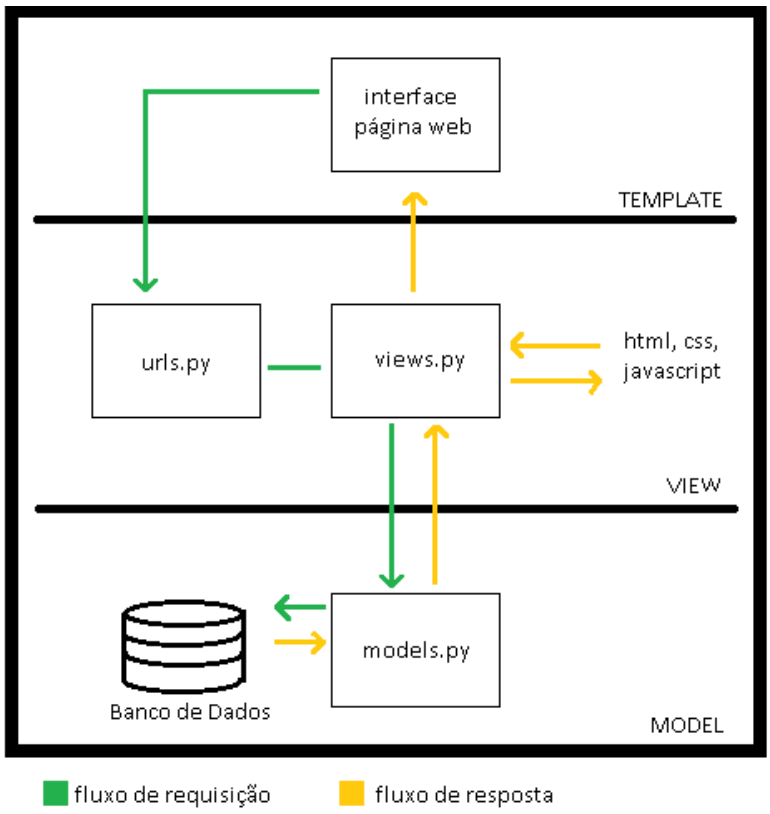

Figura 5.4. Fluxo MTV do Django

Diferente de outros *frameworks* Web que utilizam o MVC, descrito na seção 5.2, o Django utiliza a estrutura MTV (Model-Template-View), representado na Figura 5.4, onde o *framework* gerencia a maior parte da comunicação entre requisições HTTP. Quando é criado um projeto em Django temos um diretório inicial de pastas e arquivos, onde estão os arquivos models.py, views.py e urls.py, responsáveis pelo fluxo MTV. Maiores detalhes sobre a instalação do Django podem ser vistos em [Foundation 2020a]. A

seguir, explicaremos as camadas MTV e o funcionamento do fluxo entre elas:

- *Model*: Django já vem com uma solução para mapeamento objeto-relacional (ORM, do inglês *Object-Relational Mapping*) no qual o esquema do banco de dados é descrito em código Python [Foundation 2020a]. Neste caso, abstrações em Python podem ser usadas para criar consultas complexas sem que seja necessário realizar ações diretas no banco.

- *Template*: Esta é a camada de apresentação, onde as informações são visualizadas pelos usuários. Um *template* consiste de partes estáticas do arquivo HTML de saída e de partes com uma sintaxe especial que descrevem como o conteúdo dinâmico será apresentado. O Django tem um caminho de pesquisa para *templates*, o qual permite minimizar a redundância entre eles. Geralmente, uma *view* recupera dados de acordo com os parâmetros de pesquisa, carrega um *template* e o renderiza com os dados recuperados [Foundation 2020a].

- *View*: As *views* recebem a informação e o tipo da requisição ("POST" ou "GET") do lado do cliente e, em seguida, formatam os dados para que sejam armazenados no banco através dos *models* da camada *Model*. As *views* também se comunicam via *models* com o banco para recuperar dados que são transferidos posteriormente aos *templates*, para a visualização do usuário. Cada *view* é responsável por fazer uma entre duas coisas: devolver um objeto *HTTPResponse* contendo o conteúdo para a página requisitada ou levantar uma exceção como *Http404* [Foundation 2020a].

Em um site convencional, uma aplicação Web aguarda requisições HTTP do navegador. Quando uma requisição é recebida, a aplicação calcula o que é necessário fazer com base na URL e nas informações dos dados enviados via POST ou GET. Dependendo do que for necessário, a aplicação poderá ler ou gravar dados no banco ou executar outras tarefas necessárias para satisfazer a requisição. A aplicação retornará uma resposta, gerando dinamicamente uma página HTML para o navegador exibir, inserindo os dados recuperados em espaços reservados em um *template* HTML. Aplicações Web feitas com Django geralmente agrupam o código que manipula cada uma dessas etapas em arquivos separados: urls.py, views.py e models.py.

O arquivo urls.py é encarregado de armazenar o mapeamento de URLs (*urlpatterns*), redirecionando requisições HTTP para a função (também chamada *view*) apropriada, com base na URL da solicitação. As *views* são o coração da aplicação Web, recebendo requisições HTTP de clientes Web e retornando respostas HTTP.

No arquivo urls.py existe uma lista de mapeamentos do tipo URL/*view* correspondente, como pode ser visto no exemplo apresentado na Figura 5.5. O objeto *urlpatterns* é uma lista de métodos *path()*. O primeiro parâmetro do método *path()* é a URL que corresponderá a uma *view*. Opcionalmente, usa-se sinais de menor e maior (<, >) para definir partes de uma URL que serão capturadas e passadas para a *view* correspondente como argumentos nomeados. O segundo parâmetro do método *path()* é a view correspondente à URL informada e, por fim, o terceiro parâmetro é o nome para chamada no template.

Uma vez que o mapeamento em urls.py é encontrado pelo Django, a *view* correspondente à URL é chamada. As *views* geralmente são armazenadas em um arquivo

```
urlpatterns = [
    path('caminho/', views.equipment_list,name='equipment_list'),
    path('caminho/<int:id>/', views.equipment_detail, name='equipment_detail'),
]
```

Figura 5.5. Exemplo de arquivo urls.py

chamado views.py, como ilustra a Figura 5.6. Por meio deste arquivo são disparadas as respostas às requisições recebidas: podem ser acessados o banco de dados através da camada *Model* ou bibliotecas, caso seja necessário, e é enviado o retorno para renderização na camada *Template*. Todas as funções *view* recebem um objeto HttpRequest como parâmetro e retornam um objeto HttpResponse. Juntamente com o HttpRequest pode haver um ou mais argumentos para a *view*, que são nomeados na URL correspondente e recebidos como parâmetros pela função. Ao final, como mostra a Figura 5.6, a *view* retorna a função de renderização *render()* que envia ao template os dados necessários, juntamente com o *context*, que é a variável que contém dados para visualização, renderizando assim a página para visualização do usuário.

```
from django.shortcuts import render

def EntryList(request):
    context = {'objects':Entry.objects.all()}
    return render(request,'entry_list.html',context)
```

Figura 5.6. Exemplo de arquivo views.py

As aplicações feitas com Django armazenam, gerenciam e consultam dados por meio de objetos do tipo *models*, como mostrado no exemplo da Figura 5.7. Os *models* definem a estrutura dos dados armazenados, incluindo o tipo do dado, seu tamanho máximo, valores padrão, opções de lista de seleção, texto de ajuda para documentação, texto de etiqueta (*label*) para formulários etc. A definição do *model* feita no arquivo models.py é independente do banco de dados escolhido para a aplicação Web. O *framework* integra-se com diversos bancos de dados do mercado mas, por padrão, vem configurado com o SQLite. No entanto, é muito simples configurar no Django a troca de um banco para outro. Uma vez definido o banco, não é preciso escrever código para comunicar-se diretamente ele. É preciso apenas escrever a estrutura dos *models* e, em seguida, o Django, através de seu ORM, cuida de todo o trabalho de armazenamento e recuperação de dados. Se for necessário alterar a estrutura dos *models*, o desenvolvedor precisa apenas fazer as migrações, que são comandos do Django para que ele propague as alterações feitas para o esquema do banco de dados [Foundation 2020a].

```
class Entry(models.Model):
    name = models.CharField(max_length=40)
    date = models.DataTimeField()
    amount = models.IntegerField(default=0)

    def __str__(self):
        return self.name
```

Figura 5.7. Exemplo de arquivo models.py com a definição de *models*

Além de definir a estrutura dos dados, os objetos *models* oferecem um mecanismo simples para recuperação de dados no banco de dados. Por meio de uma *query* (consulta), retornam objetos nos quais serão realizadas as manipulações necessárias nas *views*. Existem vários tipos de consultas disponíveis, que podem ou não conter parâmetros para especificar os objetos a serem retornados, como podemos observar na Figura 5.8. No exemplo em questão, está sendo realizada uma consulta na tabela *Entry*, realizando um filtro por ano de publicação igual a 2005 e ordenando as consultas por ano de publicação (*pub_date*) e por título (*headline*).

```
Entry.objects.all()
Entry.objects.filter(pub_date__year=2005).order_by('-pub_date', 'headline')
Entry.objects.order_by('blog__name', 'headline')
```

Figura 5.8. Mecanismo do Django em *views.py* para recuperação de dados no banco

Uma vez que a requisição foi tratada na *view*, é retornado o resultado à camada *Template*, onde será feita a renderização de um arquivo de saída, tipicamente um arquivo HTML, contendo trechos de código em Python que manipulam os objetos retornados pelas *views* (Figura 5.9). No arquivo HTML, é possível utilizar também bibliotecas JavaScript, folhas de estilo (CSS), ou mesmo fazer uso de *frameworks front-end* como o Bootstrap, apresentado na seção 5.5.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="utf-8">
  <title>Entry List</title>
</head>
<body>
  <ul>
      {% for object in objects %}
        <li>{{ object.name }} {{ object.date }} {{ object.amount }}</li>
      {% endfor %}
    </ul>
</body>
</html>
```

Figura 5.9. Arquivo HTML com código Python para manipulação de objetos

Além disso, o Django possui diversos outros recursos que são essenciais e aceleram ainda mais o desenvolvimento de projetos Web. Alguns deles são: (i) formulários (ou *forms*), que coletam os dados do usuário e os convertem automaticamente para objetos Python; (ii) mecanismos seguros e robustos de autenticação e permissão, permitindo que os usuários do site criem contas e efetuem login ou logout com segurança; (iii) armazenamento em cache (*caching*), fazendo com que a renderização das páginas aconteça apenas quando necessário, evitando assim a lentidão na aplicação; (iv) disponibilização de uma interface padrão para administração do site, onde é possível criar, exibir e editar quaisquer modelos de dados, entre outros [Foundation 2020a].

5.5. Bootstrap

O Bootstrap é um *framework front-end*, que disponibiliza código fonte para a criação de interfaces Web. É considerado um “kit de ferramentas”, contendo componentes HTML,

CSS e JavaScript cujo uso proporciona rapidez e praticidade ao desenvolvimento. Este *framework* tem como principal objetivo auxiliar na criação de sites amigáveis e responsivos. Além disso, oferece uma grande variedade de *plug-ins*, possui temas compatíveis com vários outros *frameworks* e possui suporte para todos os navegadores. Também oferece extensibilidade usando JavaScript, com suporte interno para *plug-ins* jQuery e uma API JavaScript. O Bootstrap pode ser usado com qualquer IDE ou editor e com qualquer tecnologia ou linguagem back-end como ASP.NET, PHP, Ruby on Rails ou Python [Bacinger 2020].

O jQuery é uma biblioteca JavaScript rápida, compacta e rica em recursos. Utilizá-la simplifica em muito o processo de manipulação de eventos e o uso de animações, com uma API fácil de usar e compatível com vários navegadores [jQuery 2020]. Essa biblioteca é de código aberto e sua principal função é facilitar a utilização do JavaScript no desenvolvimento de sites.

O *framework* Bootstrap surgiu, em um primeiro momento, como uma tentativa de resolver o problema da falta de padronização visual nos sistemas Web desenvolvidos na equipe do Twitter, composta pelos engenheiros Jacob Thorton e Mark Otto em 2010. Na época de criação, a ideia era padronizar as interfaces gráficas do site, para evitar inconsistências [de Leone 2017]. Para otimizar o trabalho, resolveram desenvolver uma estrutura única para ser usada por todos os profissionais da equipe. A ferramenta foi lançada no segundo semestre de 2011 e já em 2012 tornou-se um dos projetos mais populares no GitHub.

Este *framework* facilita o trabalho de desenvolvimento pois dispensa a criação de diversos *scripts*, como acontece normalmente em muitos projetos. Com recursos como *tooltip* (dicas), *menu-dropdown*, *modal*, *carousel*, *slideshow* entre outros, garante ótimas experiências em seu uso e manipulação por parte dos desenvolvedores, com configurações bastante intuitivas [ISBRASIL 2017].

A responsividade proporcionada pelo Bootstrap permite que os usuários possam acessar os sites em todos os tipos de dispositivos de forma otimizada e sem que informações sejam perdidas durante a navegação. Por conta disso, os desenvolvedores não precisam se preocupar em fazer várias versões de um site para todos os tipos e tamanhos de telas disponíveis. O sistema de *grid* (ou grade, em português) do Bootstrap, faz uso de vários tipos de *containers*, linhas e colunas para organizar e alinhar visualmente o conteúdo [Bootstrap 2020b]. *Containers* são estruturas que contêm a informação e que ajudam a dispor o conteúdo no site. A informação contida nos *containers* é organizada em colunas, sendo 12 no total pelo padrão Bootstrap. O *layout*, o alinhamento e o tamanho das colunas da *grid* são customizáveis através de um conjunto de utilitários chamado *flexbox*. Para implementações mais sofisticadas, também é possível utilizar CSSs personalizados, além dos CSSs já disponibilizados pelo Bootstrap.

Este *framework* emprega vários estilos e configurações globais importantes, todos voltados para a normalização de estilos entre navegadores [Bootstrap 2020a]. Os componentes do Bootstrap pode ser incorporados ao projeto de duas formas. A primeira forma é através da inclusão do Bootstrap CDN (do inglês, *Content Delivery Network*). Nesse caso, todo o *framework* será carregado a partir de um repositório externo na internet, ficando a aplicação dependente da disponibilidade deste. O *link* deve ser inserido no cabeçalho do

documento HTML, na forma como é exibido na Figura 5.10.

```
<head>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
  integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk" crossorigin="anonymous">
</head>
```

Figura 5.10. Inclusão do Bootstrap CDN

As bibliotecas JavaScript do Bootstrap devem ser incluídas no projeto antes do fechamento do corpo do documento HTML, juntamente com o jQuery.js e o popper.js (Figura 5.11).

```
<body>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js" integrity="
  sha384-OgVRvuATP1z7JjHLkuOU7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI" crossorigin="anonymous"></script>
  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="
  sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>
  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="
  sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>
</body>
```

Figura 5.11. Inclusão de bibliotecas JavaScript

A segunda forma de se utilizar o Bootstrap é fazendo o download do código compilado já pronto para usar, com seus pacotes CSS e bibliotecas JavaScript compilados e minificados (Figura 5.12), sendo a partir de então acessados a partir de uma pasta local. Dessa forma, pode-se utilizar o Bootstrap sem necessitar conexão com a Internet, pois os arquivos estão sendo carregados localmente.

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap.min.css.map
│   ├── bootstrap-theme.css
│   ├── bootstrap-theme.css.map
│   ├── bootstrap-theme.min.css
│   └── bootstrap-theme.min.css.map
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphicons-halflings-regular.eot
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.ttf
    ├── glyphicons-halflings-regular.woff
    └── glyphicons-halflings-regular.woff2
```

Figura 5.12. Diretório Bootstrap

Navegadores diferentes definem regras distintas para renderização de CSS. Para solucionar essa divergência entre navegadores, surgiram as regras de "*reset*" do CSS que definem padrões de estilo com o objetivo de reduzir as inconsistências entre os navegadores em itens como altura padrão de linha, margens, tamanhos de fonte dos títulos e assim por diante. Para uma tipografia mais inclusiva e acessível, o Bootstrap usa uma pilha de

fontes nativas que seleciona a melhor *font-family* para cada sistema operacional e dispositivo, além de adotar o tamanho de fonte padrão (*font-size*) dos navegadores (tipicamente 16px), entre outras características [Bootstrap 2020c].

O Bootstrap oferece a possibilidade de se usar diversos componentes já prontos como botões personalizados, formulários, imagens e ícones, *navbars*, tabelas e modais. Com esse framework, a apresentação dos botões torna-se invariável de um navegador para outro, sendo também possível personaliza-los com diversas cores, formatos de bordas e tamanhos. Pode-se usar grupos de botões alinhados com funcionalidades das mais diversas.

Os formulários são elementos usados desde muito tempo no desenvolvimento Web. O Bootstrap estiliza elementos de entradas como <*input*>, <*textarea*> e <*select*> melhorando a visualização dos formulários. A exibição das imagens pode ser responsiva com o Bootstrap, além de ser possível deixa-las com as bordas arredondadas ou nos mais variados formatos, tamanhos e posições. É possível inclusive exibir imagens (ícones) dentro de botões, por exemplo. Com CSS é possível alterar cores, tamanhos e sombras dos mesmos.

Um outro elemento disponibilizado pelo Bootstrap são *navbars*, que são estruturas usadas pra criar menus de navegação de forma padronizada. Em telas maiores, a barra de navegação que contém o menu é exibida na horizontal, já em telas menores geralmente é transformada, de forma responsiva, em um menu suspenso.

Outro componente útil disponibilizado pelo Bootstrap para desenvolvimento de sites é o *modal*, uma janela *pop-up* que se abre sobre o conteúdo da página de acordo com uma ação prévia do usuário. O *framework* possui uma estrutura própria para os modais, com uma documentação específica.

Na seção seguinte, será exemplificado o uso de alguns destes componentes dentro do contexto de um projeto ilustrativo, que usa o Bootstrap como tecnologia *front-end*.

5.6. Projeto Ilustrativo

Para exemplificar o uso dos *frameworks* Django e Bootstrap descritos anteriormente apresentamos, a título de projeto ilustrativo, um sistema Web para controle de empréstimos de equipamentos (CONEQUI), desenvolvido pelo NUPREDS[18] (Núcleo de Práticas Profissionais em Engenharia e Desenvolvimento de Software), do Instituto Federal de Ceará, campus Tianguá. O principal objetivo do sistema é realizar o empréstimo e a devolução de equipamentos disponíveis na secretaria do campus para alunos, professores e funcionários previamente cadastrados, onde a autenticação no sistema e a confirmação das ações são feitas mediante a leitura da impressão digital do usuário.

O diagrama de classes da Figura 5.13 descreve a estrutura do sistema, com suas classes, atributos, operações e as relações entre os objetos. Para exemplificar os conceitos mostrados anteriormente, utilizaremos apenas as classes EquipmentType (Tipo de Equipamento) e Equipment (Equipamento) do diagrama. Ambas são utilizadas nos casos de uso Manter Tipo de Equipamento e Manter Equipamento, modelados cada um na

[18] https://nupreds.ifce.edu.br/

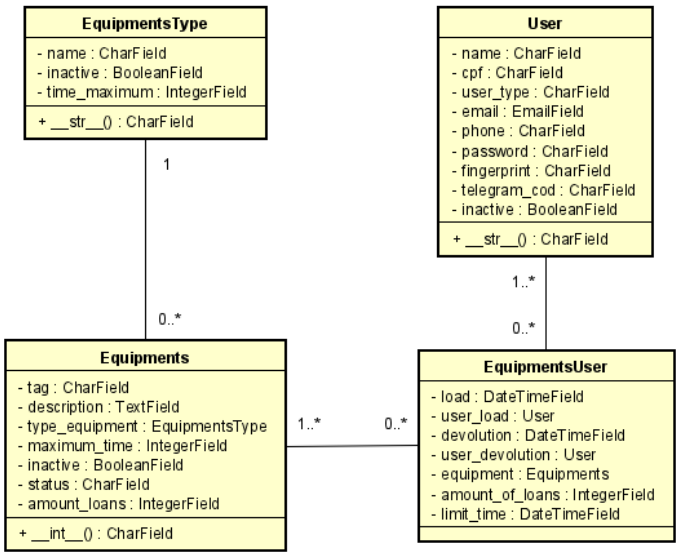

Figura 5.13. Diagrama de Classes do sistema CONEQUI

forma de um CRUD (*Create*, *Read*, *Update* e *Delete*), paradigma que contempla as quatro operações principais executadas em um banco de dados em uma única funcionalidade de sistema.

```
1   from django.db import models
2
3   class Equipment_type(models.Model):
4       name = models.CharField(max_length=20, unique=True)
5       inactive = models.BooleanField(default=False)
6       time_maximum = models.IntegerField(default=0)
7
8       def __str__(self):
9           return self.name
10
11  class Equipment(models.Model):
12      tag = models.CharField(max_length=10)
13      description = models.TextField()
14      type_equipment = models.ForeignKey('Equipment_type',on_delete=models.SET_NULL)
15      maximum_time = models.IntegerField(default=5)
16      inactive = models.BooleanField(default=False)
17      status =  models.CharField(max_length=9,null=False,default='Livre')
18      amount_of_loans = models.IntegerField(default=0)
19
20      def __int__(self):
21          return self.id
22
```

Figura 5.14. Arquivo models.py

A partir do diagrama de classes foram construídos os *models* do Django no arquivo models.py, exibidos na Figura 5.14. A classe Equipment_type (linha 3) e a classe Equipment (linha 11) possuem todos os atributos de suas respectivas representações no diagrama de classes. As classes associam-se através do atributo type_equipment da classe Equipment (linha 14), atributo este que é um objeto do tipo Equipment_type. Uma vez que são especificadas as classes do sistema e seus relacionamentos no arquivo models.py não é necessário preocupar-se com as tabelas no banco de dados, já que o próprio Django ocupa-se disso, de forma transparente para o desenvolvedor, através de seu ORM.

O mapeamento URL/*view* é descrito no arquivo urls.py, como pode ser observado na Figura 5.15. Nele, estão presentes todos os *paths* referentes à classe Equipment, onde

```python
from django.urls import path
from equipments import views

urlpatterns = [
    path('', views.equipment_list, name='equipment_list'),
    path('Ver/<int:pk>', views.equipment_view, name='equipment_view'),
    path('Novo', views.equipment_create, name='equipment_new'),
    path('Editar/<int:pk>', views.equipment_update, name='equipment_edit'),
    path('Inativar/<int:pk>', views.equipment_delete, name='equipment_delete'),
]

```

Figura 5.15. Arquivo urls.py

os quatro últimos correspondem às funções do CRUD Manter Equipamentos: Ver (*Read*), Novo (*Create*), Editar (*Update*) e Inativar (que corresponde ao *Delete*, uma vez que a remoção de equipamentos neste sistema é apenas lógica).

```python
from django.shortcuts import render, redirect, get_object_or_404
from equipments.models import *

def equipment_list(request, templete_name='equipments/equipment_list.html'):
    data = {}
    data['list_equipment'] = Equipment.objects.all().order_by('status','tag')
    return render(request, templete_name, data)

def equipment_view(request, pk, template_name='equipments/equipment_detail.html'):
    equipment = get_object_or_404(Equipment,pk=pk)
    return render(request, template_name, {'object':equipment})

def call_delete_template(request,pk,template_name='equipments/equipment_delete.html'):
    equipment = get_object_or_404(Equipment,pk=pk)
    return render(request, template_name, {'object':equipment})

def equipment_delete(request,pk):
    equipment = Equipment.objects.filter(id = pk).delete()
    return redirect('/')

```

Figura 5.16. Arquivo views.py

As *views*, fazem a recuperação de registros do banco através dos objetos definidos em models.py. Na Figura 5.16 estão algumas *views* do CRUD Manter Equipamentos. Por exemplo, a função equipment_list (linha 4) faz a busca de todos os objetos do tipo Equipamento e exibe esta listagem no template HTML (equipment_list.html) por meio da função *render*. A função equipment_view (linha 9), que corresponde à operação *Read* do CRUD Manter Equipamentos, recebe como parâmetro o identificador do objeto (pk) para recuperá-lo do banco e, sem seguida, o encaminha para ser renderizado no template HTML correspondente (equipment_detail.html). A função equipment_delete (Corresponde ao Delete do CRUD) (linha 17), recebe o identificador de um objeto do tipo Equipamento (pk) para realizar a busca e a inativação deste objeto no banco.

Os templates HTML são renderizados através da função *render* da view. Na Figura 5.17, a página equipment_list.html é renderizada pela função equipment_list da views.py. Nos templates HTML, é possível manipular os objetos por meio de funções Python. Por exemplo, na linha 11, temos um laço para percorrer todos os objetos que foram recebidos pelo template, criando assim uma tabela com todos os objetos do tipo Equipamento cadastrados no sistema. Na linha 16, temos a chamada à URL equipment_view, seguida por um parâmetro, que é o número de identificação do equipamento. Isso vai permitir que o usuário visualize os dados de um equipamento específico. Outras ações possíveis nesta tela são a edição e inativação (remoção lógica) de um equipamento específico (linhas 17

```
<h1>Equipments</h1>
<table border="1">
<thead>
    <tr>
    <th>Etiqueta</th>
    <th>Descrição</th>
    <th>Ações</th>
    </tr>
</thead>
<tbody>
    {% for equipment in list_equipment %}
        <tr>
            <td>{{ equipment.tag }}</td>
            <td>{{ equipment.description }}</td>
            <td>
            <a href="{% url "equipment_view" equipment.id %}">Visualizar</a>
            <a href="{% url "equipment_edit" equipment.id %}">Editar</a>
            <a href="{% url "equipment_delete" equipment.id %}">Inativar/Ativar</a>
            </td>
        </tr>
    {% endfor %}
</tbody>
</table>
<a href="{% url "equipment_new" %}">New</a>
<a href="{% url "home" %}">voltar</a>
```

Figura 5.17. Template HTML para listagem de equipamentos

e 18). A Figura 5.18 exibe a tela resultante deste template.

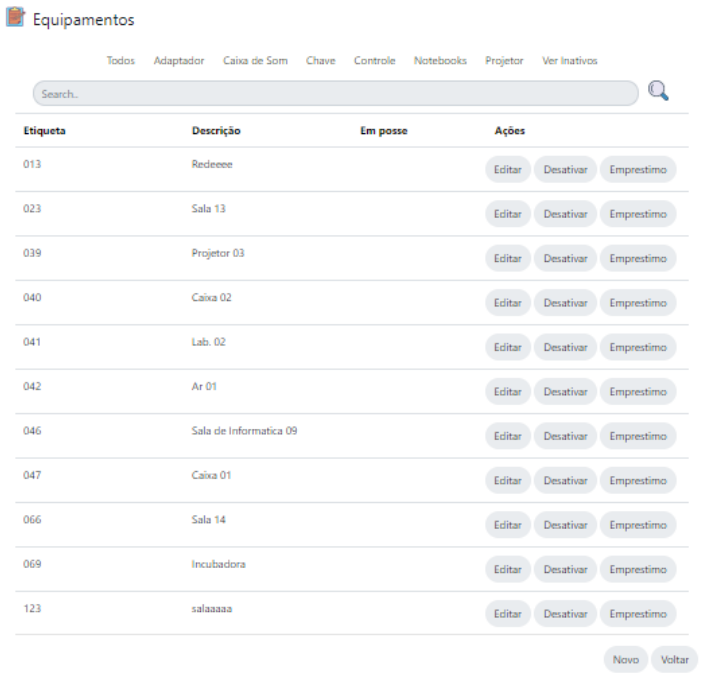

Figura 5.18. Renderização do template de listagem de equipamentos

Na tela da Figura 5.18, vários componentes Bootstrap foram utilizados. Para a barra de pesquisa, por exemplo, foi usado um formulário para que os usuários possam pesquisar equipamentos através de sua etiqueta e/ou descrição. Logo abaixo desta barra responsiva, é exibida uma tabela também responsiva contendo os equipamentos cadastrados que correspondem aos termos da busca.

A responsividade é proporcionada pelo mecanismo Bootstrap que permite que escolhamos quantas colunas da *grid* serão necessárias para exibir os elementos da página,

```
46  <div class="container-fluid">
47    <form class="form-padrao" method="POST" action="{% url 'search' type %}">{% csrf_token %}
48      <div class="input-group row">
49        <div class="col-lg-9 col-md-10 col-sm-12">
50          <input class="form-control mr-2 rounded-pill"  type="text" placeholder="Search.." name="search"
51          required="required" style="background: #e9ecef">
52        </div>
53        <div class="col-lg-1 col-md-3 col-sm-6">
54          <button class="border-0" type="submit" style="background: #FFFFFF;">
55            <span>
56              <img src="{% static "images/lupa.png" %}" style="height: 30px; width: 30px">
57            </span>
58          </button>
59        </div>
60      </div>
61    </form>
62  </div>
```

Figura 5.19. Código HTML da barra de pesquisa

considerando cada um dos tamanhos de dispositivos possíveis (sm para small, md para medium e lg para large), como pode ser observado nas linhas 49 e 53 da Figura 5.19. Ainda nesta Figura, também podemos observar que é possível customizar os botões, inclusive incluindo imagens (linhas 54 a 58).

```
60  <table class="table table-sm" style=" margin:auto; margin-top: 10px;" >
61    <thead>
62      <tr class = "d-flex">
63        <th class="col-lg-3 col-md-4 col-sm-12">
64          <a href="{% url "filter_list_equipment" 'Etiqueta' type %}" style="color: black;">Etiqueta</a>
65        </th>
66        <th class="col-lg-3 col-md-4 col-sm-12">
67          <a href="{% url "filter_list_equipment" 'Descricao' type %}" style="color: black;">Descrição</a>
68        </th>
69        <th class="col-lg-2 col-md-3 col-sm-9">
70          <a href="{% url "filter_list_equipment" 'EmPosse' type %}" style="color: black;">Em posse</a>
71        </th>
72        <th class="col-lg-4 col-md-6 col-sm-12">
73          <a href="" style="color: black;">Ações</a>
74        </th>
75        </tr>
76      </thead>
77    <tbody>
78    {% for equipment in list_equipment %}
79      <tr>
80        <td class="col-lg-3 col-md-4 col-sm-12">{{ equipment.tag }}</td>
81        <td class="col-lg-3 col-md-4 col-sm-12">{{ equipment.description }}</td>
82        {% load my_tag %}
83        <td class="col-lg-2 col-md-3 col-sm-9">{% NameUser equipment.id %}</td>
84        <td class="col-lg-4 col-md-6 col-sm-12">
85            <button class="btn rounded-pill" style="background: #e9ecef">
86                Editar
87            </button>
88            <button type="button" class="btn rounded-pill" data-toggle="modal" data-target="#ModalLabel{{equipment.id}}" style="margin-left: 5px;background: #e9ecef">
89            {% ActiveOrInactive equipment.id %}
90            </button>
91            <button class="btn rounded-pill" data-toggle="modal" data-target="#Modal{{equipment.id}}"style="margin-left: 5px; background: #4de7ff; color: white;">
92            {% else %}
93            <button type="submit" class="btn rounded-pill" data-toggle="modal" data-target="#Modal{{equipment.id}}"style="margin-left: 5px; background: #e9ecef">
94            {% endifequal %}
95            {% LoanOrDevolution equipment.id %}
96            </button>
97        </td>
98    </tbody>
99  </table>
```

Figura 5.20. Código HTML com tabela de listagem de equipamentos

Na Figura 5.20, observa-se o código da tabela HTML onde é exibida a listagem de equipamentos. A disposição dos campos da listagem nesta tabela é determinada pela identificação do tamanho do dispositivo (sm, md ou lg) e pela quantidade de colunas da *grid* necessárias para exibi-los na tela, através do atributo *class* da *tag* <*td*>.

Já na Figura 5.21, pode-se observar um exemplo do uso de modais no nosso sistema. Neste caso, o modal (pequena janela destacada) está sendo utilizado para solicitar ao usuário que informe sua impressão digital para concluir o empréstimo de um equipamento escolhido na tela anterior, que é a que está logo abaixo do modal (tela de listagem de equipamentos). Ao clicar com o botão “Emprestar” na tela de listagem de equipamentos, o modal é acionado e é solicitada a digital do usuário.

Conforme o código na Figura 5.22, para se exibir um modal é necessário um botão ou link que o acionará após ser clicado. *Data-toggle* é usado para ligar o elemento ao tipo

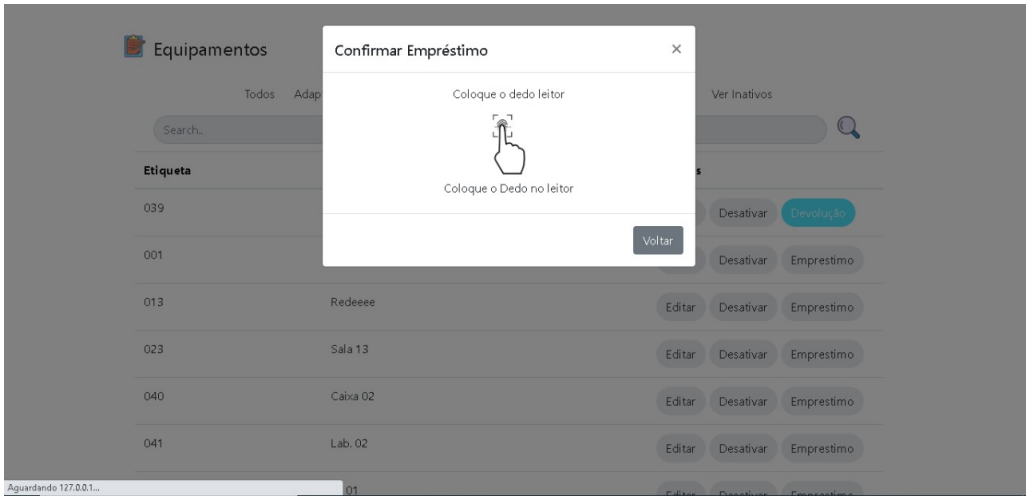

Figura 5.21. Modal para solicitação de impressão digital

```
<button type="submit" class="btn rounded-pill" data-toggle="modal" data-target="#Modal{{equipment.id}}"style=" background: #e9ecef">
    Emprestar
</button>
```

Figura 5.22. Código HTML do botão que aciona o modal

correspondente e é esse atributo que abre a janela modal. Já o *data-target* é usado junto ao modal para referência ao seu alvo, ou seja, funciona como uma forma de ID.

```
90  <div class="modal fade" id="Modal{{equipment.id}}" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel" aria-hidden="true">
91    <div class="modal-dialog">
92      <div class="modal-content">
93        <div class="modal-header">
94          <h5 class="modal-title" id="exampleModalLabel">Confirmar Empréstimo </h5>
95            <button type="button" class="close" data-dismiss="modal" aria-label="Close">
96              <span aria-hidden="true">&times;</span>
97            </button>
98        </div>
99        <div class="modal-body text-center">
100         {% for message in messages %}
101           <div class="alert alert-danger">
102             <strong>Error!</strong> {{ message }}
103           </div>
104           {% endfor %}
105           <p >Coloque o dedo leitor</p>
106           <span><img src={% static "images/digital.png" %} style="height: 80px; width: 80px;" ></span>
107           {% ifequal equipment.status 'Livre' %}
108           <form name="myForm{{equipment.id}}" id="myForm{{equipment.id}}" method="post" action="{% url "emprestar" equipment.pk %}">{% csrf_token %}
109           {% else %}
110           <form name="myForm{{equipment.id}}" id="myForm{{equipment.id}}" method="post" action="{% url "devolver" equipment.pk %}">{% csrf_token %}
111             {% endifequal %}
112             <button type="submit" class="btn border-0">Coloque o Dedo no leitor</button>
113           </form>
114       </div>
115         <div class="modal-footer">
116           <button type="button" class="btn btn-secondary" data-dismiss="modal">Voltar</button>
117         </div>
118     </div>
119   </div>
120 </div>
```

Figura 5.23. Código HTML do modal

A figura 5.23 corresponde ao código HTML da janela modal. A *div* principal (linha 90) deve ter um ID igual ao valor do atributo de destino de dados usado para acionar o modal (“Modalequipment.id”). A classe *modal* percebe o conteúdo da *div* como modal e coloca o foco nele. A classe *fade* adiciona um efeito de transição. O atributo role igual a *dialog* melhora a acessibilidade para pessoas que usam leitores de tela. A classe *modal-dialog* (linha 91) define a largura e a margem adequadas ao modal. A classe *modal-content* (linha 92) modifica o modal quanto à borda, cor de plano de fundo etc. Dentro

desta *div*, adiciona-se o cabeçalho, o corpo e o rodapé do modal. A classe *modal-header* na linha 93 é usada para definir o estilo do cabeçalho do modal. A classe *modal-title* na linha 94 é usada para dar titulo ao modal. Na linha 95, *data-dismiss* fecha o modal se o usuário clicar nele. A classe *close* estiliza o botão Fechar e a classe *modal-title* estiliza o cabeçalho com uma altura de linha adequada. A classe *modal-body* na linha 99 é usada para definir o estilo do corpo do modal. Nas linhas 100 a 114 está o conteúdo do modal, onde podemos adicionar qualquer marcação HTML como parágrafos, imagens, vídeos etc.

A documentação e a codificação completa deste projeto estão disponíveis para consulta em https://github.com/daniel-root/nupreds.

5.7. Considerações finais

A maneira como o mercado de software desenvolveu-se e a presença de sistemas computacionais na Web fez com que a indústria de software focasse cada vez mais na qualidade e eficiência de seus produtos, utilizando-se da Engenharia de Software como uma das maneiras de garantir qualidade e confiabilidade durante desenvolvimento de produtos de software. Contudo, além da qualidade e da agilidade, os desenvolvedores buscam velocidade e confiança na entrega dos produtos, por meio da reusabilidade de componentes de software durante o processo de construção de um sistema Web.

Diante da necessidade de um desenvolvimento de sistemas Web de forma mais dinâmica e eficiente, este capítulo apresentou uma solução Web completa desenvolvida com a linguagem Python e os *frameworks* Django e Bootstrap, uma vez que ambos viabilizam o uso do conceito de reúso de software. O crescimento e avanço da Internet e os custos cada vez mais baixos de serviços de computação com armazenamento em nuvem viabilizam uma explosão no crescimento do número de aplicações para o ambiente da Web e, apenas o uso de metodologias não garante a qualidade do produto final. Manter a cultura de entrega contínua de software é tão importante quanto o correto uso da Engenharia de Software durante a construção da aplicação para garantir que o dinamismo das exigências dos usuários sejam satisfatoriamente atendidas.

O capítulo abordou algumas das principais tecnologias para desenvolvimento Web vinculadas à linguagem Python e expos, por meio de uma aplicação prática, o uso e a integração da linguagem com os *frameworks* apresentados. O intuíto foi incentivar os novos desenvolvedores a perpetuarem o uso de melhores práticas de desenvolvimento de sistemas web, garantindo sempre eficiência na entrega e qualidade no produto final.

Referências

[Almeida 2019] Almeida, E. S. d. (2019). Software Reuse and Product Line Engineering. In Cha, S., Taylor, R. N., and Kang, K., editors, *Handbook of Software Engineering*, pages 321–348. Springer International Publishing, Cham.

[Bacinger 2020] Bacinger, T. (2020). What is bootstrap? a short bootstrap tutorial on the what, why, and how. [Online; acesso 20-Julho-2020].

[Bootstrap 2020a] Bootstrap (2020a). Documentação boostrap. [Online; acesso 18-Julho-2020].

[Bootstrap 2020b] Bootstrap (2020b). Sistema grid. [Online; acesso 25-Julho-2020].

[Bootstrap 2020c] Bootstrap (2020c). Tipografia. [Online; acesso 26-Julho-2020].

[de Leone 2017] de Leone, L. (2017). Bootstrap: o que é, porque usar e como começar com o framework. [Online; acesso 19-Julho-2020].

[Demchenko 2019] Demchenko, M. (2019). Six huge tech companies that use python: Does it fit your project? [Online; accessed 09-junho-2020].

[Dragos-Paul and Altar 2014] Dragos-Paul, P. and Altar, A. (2014). Designing an MVC Model for Rapid Web Application Development. *Procedia Engineering*, 69:1172–1179.

[Fawcett 2019] Fawcett, A. (2019). A Beginner's Guide to Web Development.

[Foundation 2020a] Foundation, D. S. (2020a). Documentação do django. [Online; accessed 09-junho-2020].

[Foundation 2020b] Foundation, P. S. (2020b). The python package. [Online; accessed 09-junho-2020].

[Frakes and Fox 1996] Frakes, W. and Fox, C. (1996). Quality improvement using a software reuse failure modes model. *IEEE Transactions on Software Engineering*, 22(4):274–279.

[Freeman 2017] Freeman, A. (2017). *Pro ASP.NET Core MVC 2*. Apress, London, U.K.

[Gamma et al. 1995] Gamma, E., Helm, R., Johnson, R. E., and Vlissides, J. (1995). *Design Patterns*. Prentice Hall.

[Ghimire 2020] Ghimire, D. (2020). Comparative study on Python web frameworks: Flask and Django.

[Holdener 2008] Holdener, A. (2008). *Ajax : the definitive guide*. O'Reilly, Farnham.

[ISBRASIL 2017] ISBRASIL (2017). O que é bootstrap? [Online; acesso 21-Julho-2020].

[jQuery 2020] jQuery, F. (2020). jquery write less, do more. [Online; acesso 25-Julho-2020].

[Manhas 2017] Manhas, J. (2017). Initial framework for website design and development. *International Journal of Information Technology*, 9(4):363–375.

[Molina-Ríos and Pedreira-Souto 2020] Molina-Ríos, J. and Pedreira-Souto, N. (2020). Comparison of development methodologies in web applications. *Information and Software Technology*, 119:106238.

[O'Neil 2008] O'Neil, E. J. (2008). Object/relational mapping 2008: hibernate and the entity data model (edm). In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1351–1356, Vancouver, Canada. Association for Computing Machinery.

[Schmidt et al. 2004] Schmidt, D. C., Gokhale, A., and Natarajan, B. (2004). Leveraging Application Frameworks. *Queue*, 2(5):66–75.

[Sommerville 2016] Sommerville, I. (2016). *Software engineering*. Pearson, Harlow Singapore.

[Stephens 2020] Stephens, R. (2020). Redmonk slackchat: January 2020 programming language rankings. *Procedia Engineering*.