SISTEMA DE GESTÃO DE FREQUÊNCIA DE RESIDENTES UFN

Tiago Resch Moreira[1], **Ana Paula Canal**[1], **Alexandre Zamberlan**[1]

[1]Sistemas de Informações
Universidade Franciscana (UFN), Santa Maria, RS, Brasil

{tiagoresch;apc;alexz}@ufn.edu.br

Abstract. *In the context of Residency programs, there are flows and processes that need attention. One of the flows is the control of the frequency of residents in their shifts (or scales) in care units. In this way, coordinators and preceptors could view the attendance history of residents in care units, generating reports. Thus, this work designed, implemented and installed a Web and Online System for Frequency Control for Residents at UFN, by Python-Django technology, Bootstrap framework, Linux server, Gunicorn and Nginx services, under the best practices of SCRUM methodology and Kanban technique. The results were significant, as the system is in operation at the UFN teaching hospitals.*

Resumo. *No universo de programas de Residência (Multiprofissionais), há inúmeros fluxos que precisam de atenção. Um exemplo é o controle da frequência dos residentes em seus plantões. Dessa forma, coordenadores e preceptores poderiam acompanhar os históricos de frequência de residentes, gerando relatórios diversos. Portanto, este trabalho projetou, implementou e implantou um Sistema Web e Online para Controle de Frequência de Residentes da UFN, via tecnologia Python-Django, framework Bootstrap, servidor Linux, serviços Gunicorn e Nginx, sob as boas práticas da metodologia SCRUM e da técnica Kanban. Os resultados foram significativos, uma vez que o sistema encontra-se em operação nos hospitais escola da UFN.*

1. Introdução

Os Sistemas de Informações ou os Sistemas de Conhecimentos estão presentes na sociedade e já se tornam indispensáveis nas mais diversas tarefas do dia-a-dia. Esses sistemas, há muito, fazem mais do que armazenar dados para futuros relatórios, eles promovem a integração e a contextualização da informação, permitindo a localização e a manipulação de padrões presentes nas bases de dados.

Um exemplo, é a gestão de residentes em seus programas de residência e em suas escalas em hospitais escolas. A gestão automatizada e informatizada desses processos permite aos coordenadores agilizar suas tarefas, receber e lidar com informações mais precisas, descobrir padrões dos perfis de residentes (sexo, idade, religião, entre outros), perfis comportamentais (atrasos, comprometimentos, entre outros).

Dessa forma, nesse contexto de programas de residência, um dos fluxos importantes é o controle da frequência dos residentes em suas escalas (plantões ou estágios) em unidades de atendimento. Com isso, gestores (coordenadores ou preceptores) podem monitorar históricos de frequência de residentes em unidades e, futuramente, tomar decisões,

sobre quais unidades precisam mais ou menos de residentes, quais foram os preceptores que mais atenderam residentes, entre outros.

Portanto, o objetivo deste trabalho foi projetar, implementar e implantar um Sistema Web e Online para Gestão e Controle de Frequência para Residentes. E como objetivos específicos: i) pesquisar e compilar sobre gestão de recursos humanos em geral; ii) pesquisar e compilar sobre controle de horas e frequência ponto; iii) identificar as categorias de sistemas de informação, para classificar adequadamente a proposta; iv) pesquisar e aplicar as boas práticas de desenvolvimento de software no sistema proposto; v) investigar e testar os *frameworks* de desenvolvimento Web no universo Python; vi) identificar nos trabalhos relacionados boas práticas e soluções utilizadas em contextos similares a dessa pesquisa; vii) usar o mapeamento objeto-relacional na construção do sistema.

2. Revisão bibliográfica

Esta seção busca apresentar conceitos, definições e exemplos que contextualizem a pesquisa e a proposta de sistema.

2.1. Gestão de Recursos Humanos em cursos de Residência

A Comissão de Residência Multiprofissional em Saúde - COREMU, instituída pela Lei no 11.129, de 30 de junho de 2005, tem atribuições definidas pela Portaria Interministerial no 1.077, de 12 de novembro de 2009. É uma instância de caráter deliberativo e é responsável pelo planejamento, supervisão, fiscalização, controle, normatização e administração geral das atividades desenvolvidas nesta modalidade de ensino, estando subordinada à Comissão Nacional de Residência Multiprofissional em Saúde - CNRMS, do Ministério da Educação - MEC.

A COREMU é constituída por: um coordenador e um vice-coordenador da Residência Multiprofissional em Saúde ou em Área Profissional da Saúde; um coordenador de cada Programa de Residência Multiprofissional e em Área Profissional; um representante de tutores; um representante de preceptores; um representante dos residentes e um representante do gestor local do Sistema Único de Saúde. Independente do conselho de Residência, faz-se necessário a geração da escala dos residentes em instituições da área da Saúde e o acompanhamento das horas, efetivamente, realizadas nas escalas.

2.1.1. Controle de Horas

De acordo com Consolidação das Leis do Trabalho (CLT), os colaboradores ou funcionários de uma empresa cumprem sua jornada de trabalho conforme um sistema de marcação de horários durante o mês. Dentro desse formato, é controlado a quantidade de horas trabalhadas por dia, as pausas feitas durante a jornada, horas extras, atrasos e todas as informações relacionadas à jornada dos funcionários [Rezende 2018]. De uma forma básica, a jornada de trabalho é o período que o colaborador está a disposição do empregador e, em geral, as jornadas são de 8 horas diárias de trabalho. Também pode ser adotado trabalho por meio de escala e horários diferentes para atender esse limite. Em termos de hora extra, o colaborador pode fazer até 2 horas extras por dia, porém as horas excedentes devem ser pagas a ele com um acréscimo, que pode ser de 50% e ou 100% do valor da hora normal do funcionário.

Considerando a jornada, é necessário que existe um intervalo de intrajornada, que é uma pausa feita durante o expediente para almoço e ou descanso [Rezende 2018]. Dentro disso, existe a necessidade de um sistema para controle da jornada, que é considerado o sistema de controle de horas, onde que o empregado é obrigado a marcar seu horário de entrada e de saída, sendo também necessário em casos aonde a jornada se estenda eventualmente. Esse controle, pode ser feito por meio de Livro de Ponto, por meio de computadores usando um sistema de reconhecimento de digital (por exemplo, biometria) e ou por meio de equipamento especifico de controle de horas, que seja registrado no Ministério do Trabalho [Rezende 2018].

O ponto eletrônico, conhecido também como Relógio de Ponto, é um dispositivo eletrônico para que funcionários registram os seus horários de entrada, saída e até mesmo, as pausas para o almoço. Ele é uma das formas mais comuns de registrar o ponto. Contudo, essa não é a opção de marcação de ponto mais tecnológica. Até o presente momento, há sistemas de controle de ponto *on-line*, que permitem a marcação de ponto por aparelhos móveis, o que torna a tarefa de controlar a jornada muito mais fácil e eficiente.

2.2. Sistemas de Informação

O termo Sistemas de Informação (SI) está associado a diferentes definições, categorias e aplicações. Os principais autores da área, [O'brien and Marakas 2013] e [Laudon and Laudon 2014], defendem que SI deve coletar, processar, armazenar e apresentar informações diversas à qualquer nível (operacional, tático ou estratégico) de uma instituição. Todo esse conjunto deve tratar os processos de tomada de decisão e de controle da organização, seja pelo setor da Tecnologia da Informação, seja pelo proprietário ou responsável do negócio (*stakeholders*).

Em relação às categorias, há de Sistemas de Informação Gerencial (SIG) e Sistemas de Apoio à Decisão (SAD). Ambas são complementares e estão focadas em tomadas de decisão. A SIG fornece condições de coletar, tratar, armazenar e entregar informações (relatórios), enquanto o SAD ajuda na solução de problemas, apontando tendências (predições) e/ou ocorrências percebidas nos relatórios e/ou documentos entregues pelo SIG [O'brien and Marakas 2013] e [Laudon and Laudon 2014].

O uso da Internet para execução de sistemas (*online*) distribuídos geograficamente é um fato nas diferentes áreas. Dessa forma, o conceito de *Cloud Computing* (Computação em Nuvem), que se refere ao processamento computacional e armazenamento de dados que são executados fora da infraestrutura local [dos Santos et al. 2021], é uma realidade de solução computacional. Assim, por meio de uma conexão com a Internet é possível utilizar serviços de processamento, banco de dados, etc. de modo seguro. Com isso, todo o processamento e armazenamento ocorre no lado do servidor, permitindo manutenção e garantias de segurança do sistema de informação em um ambiente centralizado, seguro e controlado [dos Santos et al. 2021].

2.3. Boas Práticas de Desenvolvimento e *Frameworks*

Nesta seção buscou-se apresentar conceitos exemplos de boas práticas dos processos de projeto e desenvolvimento de sistemas, principalmente via a metodologia SCRUM e a técnica Kanban. Além disso, foram apresentadas noções de *frameworks*.

Na metodologia SCRUM é possível dividi-la em papéis (*scrum master*, *product owner*, *dev team*), eventos (planejamento, revisão e retrospectiva de *sprint*) e artefatos (*product backlog*, *sprint backlog* e incremento/entrega) [Wykowski and Wykowska 2019].

Conforme em [Sutherland 2016], é consenso que a metodologia SCRUM tem por prioridade o desenvolvimento de um sistema (produto ou serviço) de modo menos complexo, mais eficiente e focado em resultados. Os encontros (*sprints*) são semanais ou quinzenais e sempre há apresentações das funcionalidades do sistema (*product backlog*) em forma de protótipo. Isto é, o interessado ou o responsável pelo sistema acompanha e testa toda funcionalidade implementada até aquele instante.

Dentro do contexto deste trabalho, há os papéis do coordenador da Comissão de Residências Multidisciplinar (COREMU), alunos da área da Computação da Universidade Franciscana e professor do Laboratório de Práticas da área da Computação. Dessa forma, os idealizadores deste sistema (coordenador da COREMU e o professor do Laboratório de Práticas da Computação) são os responsáveis pelos processos de modelagem, desenvolvimento, testes e implantação do sistema. O delineamento, as definições de aspectos funcionais (planejamento, execução, revisão e retrospectiva do *sprint*) do sistema, os layouts e a dinâmica de funcionamento geral ficaram sob responsabilidade do coordenador da COREMU. Os alunos da área da Computação fazem parte do time de desenvolvimento (*dev team*) do sistema. O professor do Laboratório de Práticas da Computação comportou-se, também como *scrum master*, que deve gerenciar o desenvolvimento como um todo (prazos, metas, responsáveis, controle de versões do sistema, reuniões, etc). E a COREMU e a Universidade Franciscana são consideradas as donas do produto (*product owner*).

Em geral, a metodologia SCRUM fornece sugestões ou boas práticas no projeto e/ou desenvolvimento de sistemas, que podem ser customizadas ou, até mesmo, suprimidas. Porém, o uso da técnica Kanban auxilia na gestão de tarefas, prazos, responsáveis, entre outros, impactando diretamente nos *sprints* e nos *product backlog*. Dentro das boas práticas de projeto e desenvolvimento de sistemas, há o de versionamento de código associado a *frameworks*. Um *framework* para desenvolvimento de software é uma abstração que fornece funcionalidades genéricas para reutilização de código para partes do sistema em desenvolvimento [Wykowski and Wykowska 2019]. As principais vantagens de usar *frameworks* são maior facilidade para a detecção de erros, a concentração na abstração de soluções do problema e a eficiência na resolução dos problemas e otimização de recursos.

2.3.1. *Django* e *Bootstrap*

Django é um *framework* que é usado para criação de aplicações Web escrito em *Python*, com o intuito de agilizar a construção dos programas e garantir qualidade e segurança do sistema, uma vez que possui inúmeras bibliotecas ou pacotes com funcionalidades diversas ao programador [Framework Django 2020]. *Django* também ajuda na escalabilidade dos sistemas, ou seja, consegue oferecer capacidade de expansão sem perda de desempenho, como é o que acontece por exemplo com Mozilla Firefox, Pinterest e Instagram [Santiago et al. 2020]. Diferente de outros *frameworks* Web que utilizam o MVC (*Model-View-Controller*), o Django utiliza a estrutura MTV (*Model-Template-View*), onde o *fra-*

mework gerencia a maior parte da comunicação entre requisições HTTP. Quando é criado um projeto em *Django*, há um diretório inicial de pastas e arquivos, onde estão os arquivos models.py, views.py e urls.py, responsáveis pelo fluxo MTV [Framework Django 2020].

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 ambiente de desenvolvimento ou editor e com qualquer tecnologia ou linguagem *back-end* como ASP.NET, PHP, Ruby on Rails ou Python [Santiago et al. 2020]. 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) do *Bootstrap*, faz uso de vários tipos de *containers*, linhas e colunas para organizar e alinhar visualmente o conteúdo. *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* [Santiago et al. 2020].

2.3.2. Controle de Versão

Um sistema de controle de versões de código (ou versionamento) é um software que tem como objetivo gerenciar as diferentes versões de arquivos (programas ou não) em um sistema. O versionamento deve manter histórico de alteração dos arquivos fontes e também da documentação produzidos e/ou alterados, controlando conflitos, produtividade, status das versões. As vantagens do uso de versionamento, por exemplo, são qualidade no trabalho em equipe (várias pessoas trabalhando sobre um mesmo sistema sem problemas de conflitos), resgate de versões, ramificação do projeto (divisão do projeto em várias linhas/ramos de desenvolvimento, que podem ser trabalhadas paralelamente, sem que uma interfira na outra), segurança (controle de invasão e criptografia) e backup e confiança (armazenamento em nuvem de arquivos, por exemplo) [Molinari 2007]. Neste trabalho, está sendo utilizado o sistema de controle de versões do *GitHub*, o que possibilitou o desenvolvimento e versionamento do trabalho de uma forma efetiva e prática.

2.3.3. Trabalhos relacionados

Os trabalhos relacionados discutidos nesta seção são de [Cezário and Zamberlan 2021] e [Zamberlan et al. 2021], ambos utilizaram arquitetura cliente-servidor (monolítica),

frameworks Bootstrap e *Django*, linguagem *Python*, servidor Linux Ubuntu, serviços NGINX e GUNICORN. Os trabalhos foram escolhidos como relacionados, pois usaram tecnologias que estão sendo utilizadas nesta pesquisa. Além disso, os pontos positivos (ou boas práticas de desenvolvimento) foram incorporadas neste sistema proposto, ou seja, funcionalidades previamente implementadas e testadas estão sendo reutilizadas neste sistema.

No trabalho de [Cezário and Zamberlan 2021], foi projetado e implementado um sistema Web para gerenciar uma base de dados referentes a pacientes diabéticos. O sistema Web garantiu os processos de *Create, Retrieve, Update, Delete* (CRUD) de pacientes, nutricionistas, educadores físicos, médicos, atividades físicas, alimentos, medicamentos, refeições realizadas, atividades executadas, entre outros. Trabalhou-se com dados de monitoramento referentes a data, quantidade de insulina utilizada, quantidade de calorias e carboidratos ingeridos, qualidade do sono (escala *Likert*) e esforço subjetivo de atividades físicas (escala *Borg*). A pesquisa também teve foco no estudo e na avaliação de métodos de mineração de dados adequados para o contexto de monitoramento de dados de pacientes diabéticos. A pesquisa foi baseada em revisão bibliográfica e estudo de caso. Os resultados foram considerados significativos, uma vez que 3 algoritmos ((*RandomForest*, *Expectation Maximisation* e *Bagging*)) tiveram respostas conclusivas do estudo.

O trabalho de [Zamberlan et al. 2021] tratou de gestão informatizada de documentos, uma vez que a legislação brasileira têm facilitado as organizações em manipular documentos na sua forma eletrônica, em detrimento de sua forma fisica, liberando espaços e auxiliando em consultas diversas a conteúdos presentes nos documentos armazenados. O objetivo do trabalho foi implementar um sistema Web de gestão de documentos do tipo ata, para coordenações e secretarias de cursos de instituições de ensino. A pesquisa foi amparada por revisão bibliográfica e estudo de caso, e o sistema foi construído seguindo as boas praticas da metodologia SCRUM. O sistema também possui CRUD para usuários professores e secretárias, além de documentos do tipo ata. Uma das principais funcionalidades do sistema é a geração automatizada das atas (formato .pdf) e um mecanismo de pesquisa nos conteúdos das atas.

3. Materiais e métodos

Este trabalho é uma pesquisa interdisciplinar entre o curso de Sistemas de Informação, Laboratório de Práticas da Computação UFN, Programa de Residência da UFN e empresa parceira do Laboratório de Práticas (ER Sistemas).

Na Figura 1, é possível visualizar as relações entre os conceitos de Sistemas de Informação, a área da Computação, o sistema de controle de frequência e a aplicação na Residência UFN.

A solução proposta é baseada na metodologia SCRUM e na técnica Kanban. As ferramentas utilizadas são:

- Trello: ferramenta *online* para operacionalizar a técnica Kanban, que gerencia o cronograma de atividades deste trabalho;
- Astah: ferramenta para a diagramação de aspectos funcionais e estruturais da solução proposta;
- GitHub: Sistema *online* para controle de versionamento de código do sistema;

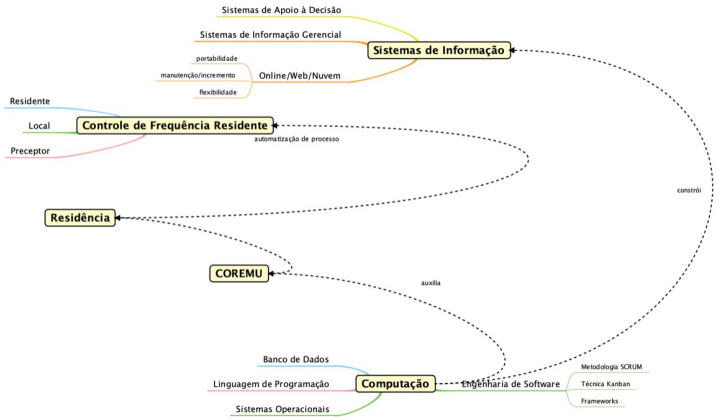


Figura 1. Mapa mental da relação Sistemas de Informação, área da Computação e Residência UFN.

- Linguagem Python 3.8, *framework* Django 3.1.5 e o ambiente de desenvolvimento Visual Studio Code;
- Pacotes Python-Django: django-bootstrap3, django-easy-pdf, gunicorn (Python WSGI HTTP Server), nginx (servidor HTTP, POP, IMAP e de proxy reverso), entre outros, para o desenvolvimento WEB do sistema; Bancos de dados SQLite (máquina local) e MySQL (máquina servidor), para o armazenamento dos dados do sistema.

4. Resultados

Na Figura 2, é possível perceber os atores (Administrador, Coordenador, Tutor, Preceptor, Secretária, Residente) que usam ou interagem com o sistema, bem como as funcionalidades básicas, como por exemplo gerenciar usuários, gerenciar programas de residência, gerenciar unidades de atendimento, associar residentes a preceptores, liberar o ponto de residente, marcar ponto, entre outras.

Em relação à modelagem de aspectos estruturais do sistema, na Figura 3, apresenta-se os dois principais pacotes da proposta que são gestão básica e controle de frequência (foco do trabalho). O controle de frequência tem relação a um programa de residência e seus integrantes (basicamente, residentes e preceptores).

Na Figura 4, há o Diagrama de Entidade-Relacionamento que ilustra como os dados do sistema são armazenados em tabelas (entidades). As principais tabelas (entidades) são:

- Administrativo: dados de todos os usuários que não são residentes, como matricula, cpf, email, etc;
- Especialidade: armazena dados de especialidades que serão utilizadas para categorizar tanto um usuário Administrativo, quanto um setor de um hospital;
- Setor: dados dos diferentes setores ou alas de um hospital, por exemplo, ambulatório, UTI, etc;
- Hospital: dados de hospitais, prontos socorros e unidades de atendimento em uma cidade;
- Programa: dados dos programas de residência cadastrados, com nome, conselho de residência, duração do programa, etc;

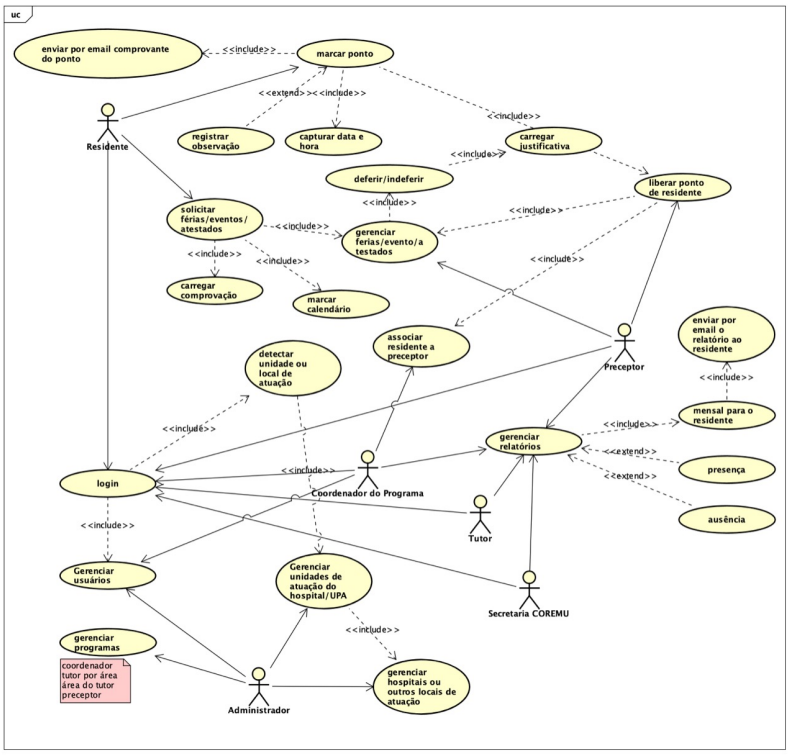


Figura 2. Diagrama de Casos de Uso com as principais funcionalidades do sistema.

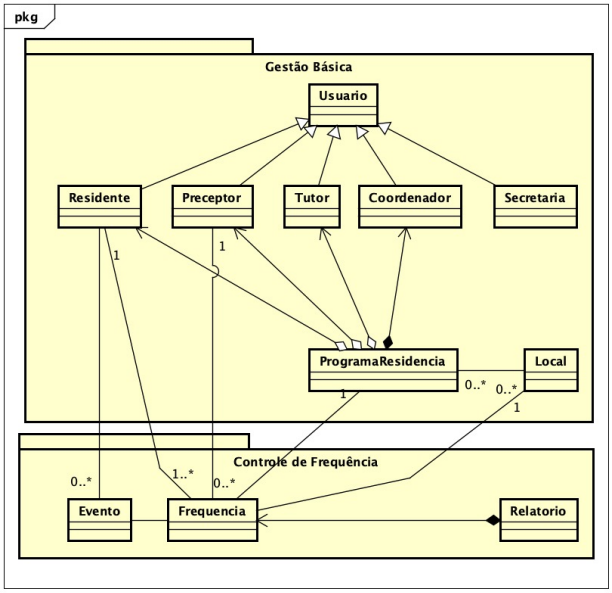


Figura 3. Diagrama de Domínio os principais elementos do sistema.

- Residente: armazena dados do residente cadastrado, com atributos matrícula, cpf, email, data de nascimento, etc;
- Frequência_Ponto: para armazenar o ponto (ou frequência) do residente quando associado a um preceptor e a um setor de um hospital. Essa tabela é o foco do sistema proposto;

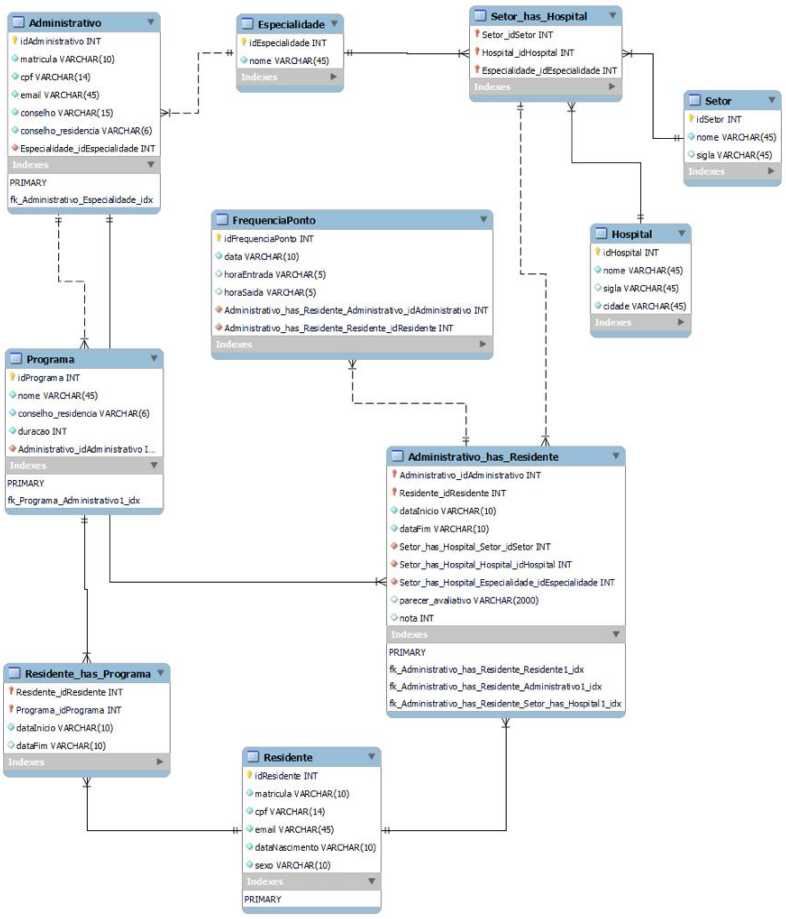


Figura 4. Diagrama de Tabelas.

- Administrativo_has_Residente: tabela de relacionamento entre um residente e um usuário administrativo, caracterizando a relação de residente com preceptor. Nessa tabela, são armazenados dados de início e fim de uma escala entre residente e preceptor, qual o setor, especialidade e hospital que os dois devem trabalhar, parecer avaliativo do preceptor para o trabalho realizado pelo residente naquele período;
- Residente_has_Programa: tabela de relacionamento entre residente e um programa de residência. Nessa tabela é possível identificar o período (histórico) do residente em um programa específico;
- Setor_has_Hospital: novamente uma tabela de relacionamento entre Hospital, Setor e Especialidade.

Dessa forma, o diagrama Entidade-Relacionamento fornece uma visão de aspectos estruturais do sistema, no caso, o banco de dados.

Ressalta-se que a Figura 4 ilustra, ao mesmo tempo, as tabelas do banco de dados e as classes básicas do sistema, uma vez que Django possui a ferramenta de mapeamento objeto-relacional. Dessa forma, toda a modelagem orientada a objetos presente nos arquivos *models.py* é convertida automaticamente em tabelas do banco, para persistência dos dados (*makemigrations* e *migrate*).

Para a geração de um diagrama de classe equivalente ao diagrama ER, são ne-

cessárias algumas instalações e configurações no sistema[1]. Com isso, pode-se gerar um diagrama de classe fiel à estrutura implementada no sistema.

Por este projeto se tratar de um sistema Web com uso do *framework Django* e a linguagem de programação *Python*, há obediência quanto ao uso do modelo (exclusivo do *Django*) Model-View-Template (MVT), que é equivalente ao modelo teórico-prático Model-View-Controller (MVC). Dessa forma, há uma estrutura de pastas ou diretórios e seus arquivos que precisam existir e estar em locais previamente definidos, como mostra a Figura 5.

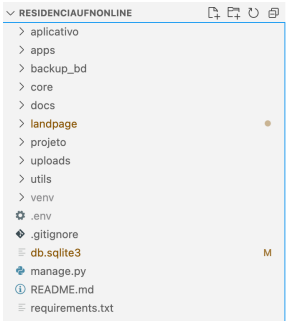


Figura 5. Estrutura de pastas e arquivos de projeto Django no contexto MVT.

Onde a pasta aplicativo contém arquivos referentes ao *front-end* da aplicação, com arquivos html, css, etc, ou seja, as camadas View e Template.

4.1. Interfaces Gráficas

Já em relação aos elementos de tela, apresentam-se as interfaces gráficas das principais funcionalidades do sistema, que são os *Create, Retrieve, Update, Delete* (CRUD) de usuários administrativos e residentes, hospitais, setores e especialidades, programas de residência. Além disso, as interfaces de ponto ou frequência

Registra-se que o sistema já está nos servidores da instituição sob o endereço https://residencia.lapinf.ufn.edu.br, e que o sistema entrou em funcionamento em primeiro de agosto de 2022. Também se destaca que o sistema Residência UFN On-line (com todas as suas funcionalidades projetadas e implementadas) foi registrado no Instituto Nacional de Propriedade Intelectual (INPI) sob o número BR512022001377-0.

Na Figuras 6, é possível visualizar o portal Residência On-line UFN, em que um usuário pode obter informações do projeto e acessar o sistema. A Figura 7 ilustra a funcionalidade em que Coordenadores ou outros usuários administrativos como a liberação respectiva podem gerenciar as escalas de residentes em hospitais e setores. A gestão contempla associar um residente de um programa, à um preceptor, em um determinado período, em um setor de um hospital.

A Figura 8 mostra a lista de todos os pontos realizados pelos residentes para usuários administrativos ou coordenadores. É possível acompanhar nome do residente, detalhes da escala (caso o ponto tenha sido realizado na escala), hora de entrada e saída, bem como total de horas efetivas.

[1]Há um passo-a-passo em https://ohmycode.com.br/gerando-o-diagrama-de-classes-do-seu-projeto-django.

Figura 6. Portal Residência On-line UFN.

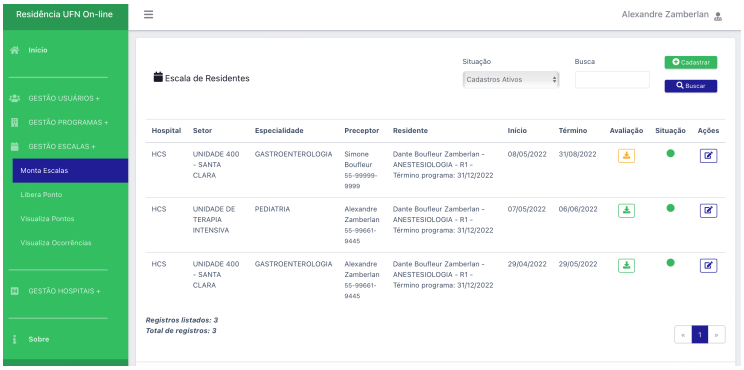


Figura 7. Visão administrativa para geração e gestão de Escalas dos Residentes.

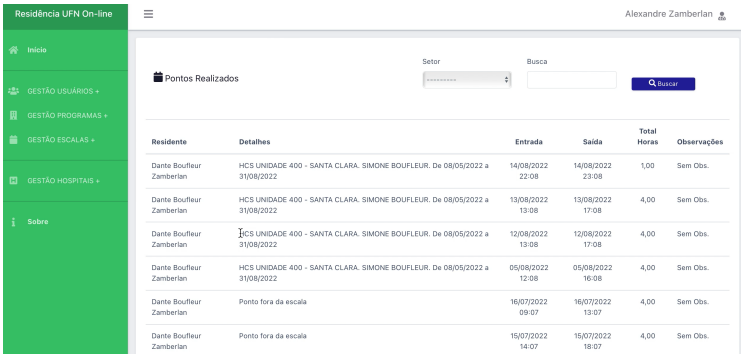


Figura 8. Visão administrativa para acompanhar Pontos realizados pelos Residentes.

Na Figura 9, na visão administrativa ou de coordenação, os gestores podem visualizar todas as ocorrências geradas no ponto eletrônico, o motivo da ocorrência, se já foram justificadas e/ou deferidas.

Na Figura 10, pode-se visualizar a página inicial do residente. Há alguns *widgets*[2] de acesso de funcionalidades, mas destacam-se 'Minhas Escalas', 'Escala Atual' e

[2]Um aplicativo, ou um componente de uma interface, que permite que um usuário execute uma função ou acesse um serviço [Porto 2020].

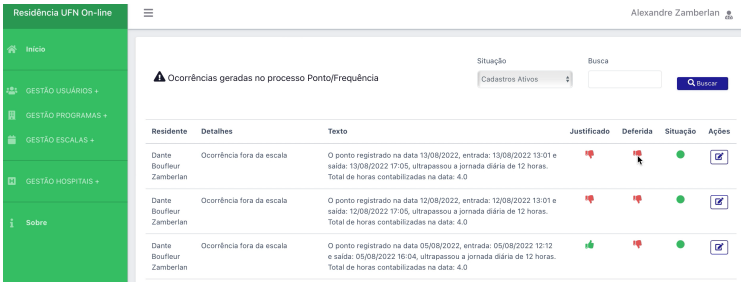


Figura 9. Visão administrativa para acompanhar eventos inconsistentes ao Ponto, ou seja, Ocorrências.

'Ocorrências'. A partir deles, o residente pode acessar tais funcionalidades. Registra-se que o projeto do sistema Residência UFN On-line contempla outras funcionalidades além do controle de ponto, que são funcionalidades de processos avaliativos dos residentes.

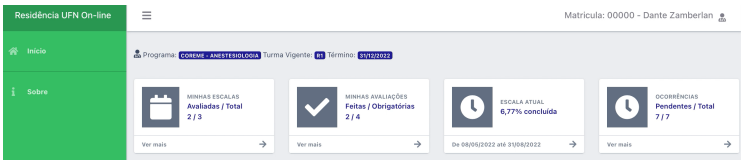


Figura 10. Visão do Residente com *widgets* para escalas, processos de avaliação, pontos e ocorrências.

Na Figura 11, o residente pode acompanhar os registros de pontos realizados, se houve ou não ocorrência e acessar o formulário para justificar a ocorrência (diretamente no botão 'Sim' da coluna 'Teve ocorrência?').

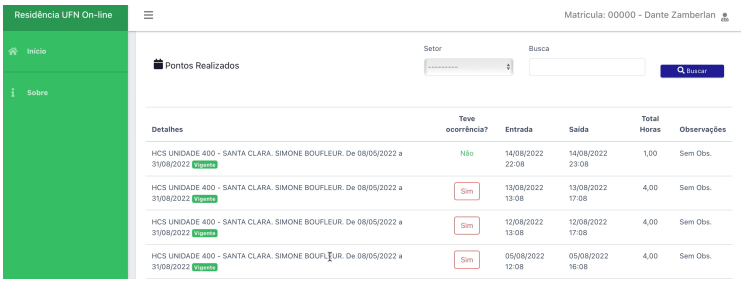


Figura 11. Visão do Residente para acompanhar seus pontos realizados.

Na Figura 12, o residente pode preencher campos para justificar a sua ocorrência do ponto, inclusive carregar arquivo com imagem de atestado, por exemplo.

A Figura 13 apresenta as interfaces gráficas em que a secretária ou técnico da unidade libera página modal[3] para que residentes realizem o ponto, via autenticação de usuário por senha.

4.2. Implementação

Nesta seção, busca-se destacar alguns trechos de código que significam o trabalho, fornecendo aspectos de implementação da funcionalidade do ponto eletrônico.

[3]Modals são páginas que são posicionadas acima de todos os elementos do documento, removendo a barra de rolagem para que o conteúdo não role.

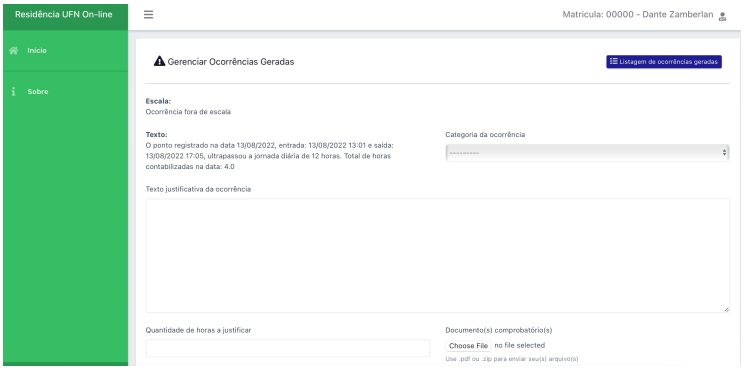


Figura 12. Visão do Residente com formulário para entrar com justificativa da falha no ponto.

Figura 13. Visão Secretária ou Técnico: imagem da esquerda é o modal para o residente autenticar no sistema; a imagem da direita é outro modal para o residente efetivamente realizar o ponto.

Na Figura 14, é possível destacar a implementação orientada a objetos da classe Ponto, referente à camada *model* do *framework* Django. Há atributos escala (relação de associação à classe Escala), residente (relação de associação à classe Residente), entrada, saída, turno, total_horas. Esses atributos são mapeados ao banco de dados pelos processos de migração do mapeamento objeto-relacional do Django. Já a Figura 15, há a classe responsável em gerar objetos referentes às falhas de marcação de ponto ou frequência. Essas falhas estão vinculadas ao registro do ponto fora da escala, carga horária excedente à 12 horas, carga mínima de 4 horas, entre outras. Nessa classe, tanto residentes, quanto preceptores acessam seus dados. O residente pode visualizar o motivo da ocorrência e justificá-la, enquanto o preceptor defere ou não a justificativa registrada. Os principais atributos dessa classe são ponto, residente e escala, em que todos são chaves estrangeiras ao banco de dados (ou classes), respectivamente a Ponto, Residente e Escala. Há também, texto (motivo da ocorrência, que é gerado pelo sistema de forma automática quando a marcação do ponto fica inconsistente), tipo, texto_justificativa, qtd_horas_justificadas, arquivo_documento (com algum comprovante, médico por exemplo), entre outros. Registra-se que os métodos com a anotação '*@property*', são métodos que podem ser invocados na camada *template*, ou seja, em arquivos *.html*.

Por fim, a Figura 16, que é o foco deste trabalho, representa o processo que deve

```python
class Ponto(BaseModel):
    escala = models.ForeignKey('escala.Escala', null=True, on_delete=models.PROTECT)
    residente = models.ForeignKey('residente.Residente', null=True, blank=False, on_delete=models.PROTECT)
    entrada = models.DateTimeField('Entrada')
    saida = models.DateTimeField('Saída')
    turno = models.CharField('Turno', max_length=10, choices=choices.TURNOS)
    total_horas = models.DecimalField('Total de horas', decimal_places=2, max_digits=5)
    obs = models.TextField('Observações diárias', max_length=400, null=True, blank=True)

    def setar_total_horas(self):
        self.total_horas = (self.saida - self.entrada).seconds / 60 / 60

    def setar_turno(self):
        horas_manha = [7, 8, 9, 10, 11, 12]
        horas_tarde = [13, 14, 15, 16, 17, 18]
        horas_noite = [19, 20, 21, 22]
        horas_madrugada = [23, 0, 1, 2, 3, 4, 5, 6]

        if self.entrada.hour in horas_manha:
            self.turno = 'MANHA'
        elif self.entrada.hour in horas_tarde:
            self.turno = 'TARDE'
        elif self.entrada.hour in horas_noite:
            self.turno = 'NOITE'
        elif self.entrada.hour in horas_madrugada:
            self.turno = 'MADRUGADA'

    def save(self, *args, **kwargs):
        self.setar_total_horas()
        self.setar_turno()
        super().save(*args, *kwargs)
```

Figura 14. Classe que representa o Ponto ou Frequência.

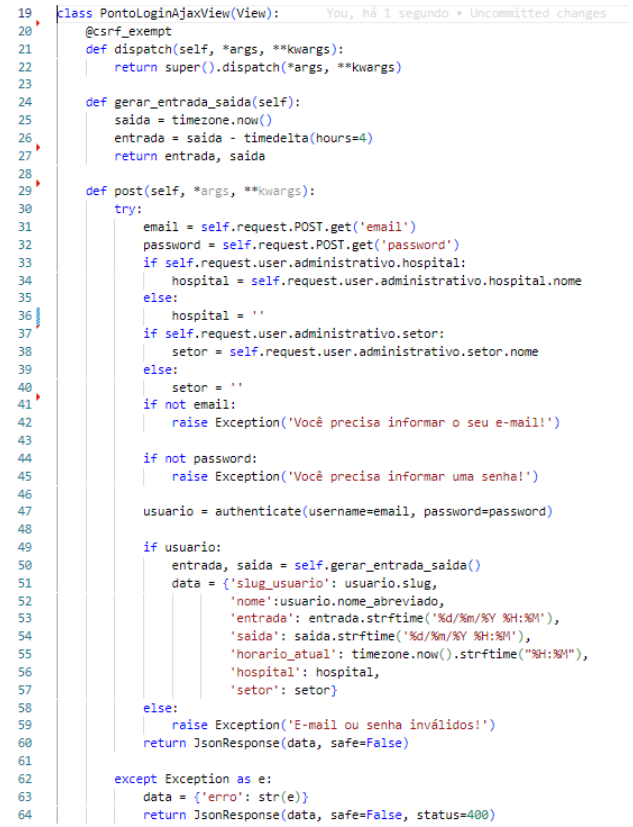
```python
class Ocorrencia(BaseModel):
    ponto = models.ForeignKey('escala.Ponto', verbose_name='Ponto *', null=True, blank=False,
                              on_delete=models.PROTECT)
    residente = models.ForeignKey('residente.Residente', verbose_name='Residente *', null=True, blank=False, on_delete=models.PROTECT)
    escala = models.ForeignKey('escala.Escala', verbose_name='Escala *', null=True, blank=False, on_delete=models.PROTECT)
    texto = models.TextField('Ocorrência gerada', max_length=400, null=True, blank=True)
    tipo = models.CharField('Categoria da ocorrência', max_length=35, choices=choices.CATEGORIA_OCORRENCIA)
    texto_justificativa = models.TextField('Texto justificativa da ocorrência', max_length=400, null=True, blank=False)
    qtd_horas_justificadas = models.PositiveIntegerField('Quantidade de horas a justificar', null=True, blank=False)
    arquivo_documento = models.FileField('Documento(s) comprobatório(s)', null=True, blank=True, upload_to='mídias', help_text='Use .pdf ou .zip pa
    deferida = models.BooleanField('Ocorrência deferida', default=False, help_text='Preceptor ou coordenador, defira ou não a justificativa de ocor
    texto_indefericao = models.TextField('Texto para justificar a ocorrência não deferida', max_length=1000, null=True, blank=True, help_text='Prec

    objects = models.Manager()
    pendentes_residente = OcorrenciaPendenteResidente()

    def save(self, *args, **kwargs):
        super().save(*args, *kwargs)

    @property
    def get_absolute_url(self):
        return reverse('aplicativo:ocorrencia_update', kwargs={'slug': self.slug})

    @property
    def get_justificativa_url(self):
        return reverse('aplicativo:ocorrencia_residente_update', kwargs={'slug': self.slug})

    @property
    def get_detail_url(self):
        return reverse('aplicativo:ocorrencia_detail', kwargs={'slug': self.slug})
```

Figura 15. Classe para registro de ocorrências nas falhas de Ponto ou Frequência.

capturar e tratar registros de pontos ou frequências. Decidiu-se por usar uma interface *modal*, como apresentado na Figura 13. Nesse código, há a classe *PontoLoginAjaxView* que tem dois métodos: *gerar_entrada_saida* e *post*. Esse último método que autentica o residente no sistema antes de abrir o modal para realização do ponto.

4.3. Resultados e discussões

Os resultados estão na modelagem do sistema, seja tanto nos aspectos estruturais (banco de dados), quanto nos aspectos funcionais (os principais serviços da ferramenta) e nas funcionalidades implementadas, avaliadas e entregues. Para isso, foram apresentados diagramas UML e telas dessas principais funcionalidades. Também registra-se que o sistema está à disposição dos usuários de residência, desde agosto de 2022. Já em 23 de setembro deste mesmo ano, o sistema passou por um processo de teste piloto no Hospital Casa de Saúde, com 10 pessoas do Programa de Residência Multiprofissional em Atenção Clínica Especializada com Ênfase em Infectologia e Neurologia, envolvendo residentes, coordenador, preceptores e tutor. O piloto durou 30 dias (período das escalas de plantões dos residentes) e nele foram possíveis apontar e corrigir equívocos de processo, dificuldades



```python
class PontoLoginAjaxView(View):      You, há 1 segundo • Uncommitted changes
    @csrf_exempt
    def dispatch(self, *args, **kwargs):
        return super().dispatch(*args, **kwargs)

    def gerar_entrada_saida(self):
        saida = timezone.now()
        entrada = saida - timedelta(hours=4)
        return entrada, saida

    def post(self, *args, **kwargs):
        try:
            email = self.request.POST.get('email')
            password = self.request.POST.get('password')
            if self.request.user.administrativo.hospital:
                hospital = self.request.user.administrativo.hospital.nome
            else:
                hospital = ''
            if self.request.user.administrativo.setor:
                setor = self.request.user.administrativo.setor.nome
            else:
                setor = ''
            if not email:
                raise Exception('Você precisa informar o seu e-mail!')

            if not password:
                raise Exception('Você precisa informar uma senha!')

            usuario = authenticate(username=email, password=password)

            if usuario:
                entrada, saida = self.gerar_entrada_saida()
                data = {'slug_usuario': usuario.slug,
                        'nome':usuario.nome_abreviado,
                        'entrada': entrada.strftime('%d/%m/%Y %H:%M'),
                        'saida': saida.strftime('%d/%m/%Y %H:%M'),
                        'horario_atual': timezone.now().strftime("%H:%M"),
                        'hospital': hospital,
                        'setor': setor}
            else:
                raise Exception('E-mail ou senha inválidos!')
            return JsonResponse(data, safe=False)

        except Exception as e:
            data = {'erro': str(e)}
            return JsonResponse(data, safe=False, status=400)
```

Figura 16. Código da classe PontoLoginAjaxView para o modal do registro do ponto eletrônico.

visuais e de acesso do sistema ponto pelos residentes. A dinâmica de funcionamento foi via uso de um *tablet* (com o sistema ponto habilitado aos residentes), em posse da secretária ou técnico responsável pelo setor em que os residentes trabalharam. Sempre no final do expediente do turno, o residente dirigia-se até a secretária ou técnico e solicitava o *tablet* para efetivar seu ponto.

O próximo passo, é liberar o uso do sistema aos demais programas de residência e hospitais escola, totalizando um total de 47 usuários administrativos do tipo, coordenador, tutor e preceptores, 40 usuários residentes nos 11 programas de residência (multiprofissional e médica), 3 hospitais escola, 18 setores ou alas e 30 especialidades.

5. Conclusões

Ao longo do texto, buscou-se discutir a proposta do sistema, mas sempre contextualizando-o no tipo de Sistema de Informação a que ele pertence: mescla de Sistema de Informação Gerencial com Sistema de Apoio à Decisão. Também, foram apresentados fundamentos sobre boas práticas de desenvolvimento de sistemas, passando pela metodologia SCRUM, técnica Kanban, uso de *frameworks*.

Sabe-se que o sistema proposto possui peculiaridades e alguns desafios que precisam ser superados, principalmente na parte de aceitação dos residentes, uma vez que presença e carga horária em suas escalas são monitoradas. Para isso, foram realizadas reuniões com todos os membros do Conselho de Residência Multiprofissional (CO-

REMU), em que além da apresentação do sistema, houve momento de conscientização da importância do sistema.

Portanto, pode-se afirmar que os objetivos (geral e específicos) do trabalho foram atingidos. Além disso, todas as funcionalidade mapeadas na modelagem foram implementadas, testadas e aprovadas durante o piloto do sistema.

Finalmente, o sistema foi desenvolvido sob a coordenação do Laboratório de Práticas da Computação e com a assessoria da Empresa ER Sistemas. E isso contribuiu para a qualidade das funcionalidades pensadas, implementadas e entregues.

Referências

Cezário, R. and Zamberlan, A. (2021). *Algoritmos De Mineração De Dados Em Sistema De Monitoramento De Diabetes*. Trabalho de Conclusão de Curso Sistemas De Informação - Universidade Franciscana (UFN), Santa Maria, RS, Brasil. Disponível em https://tfgonline.lapinf.ufn.edu.br.

dos Santos, R. E., de Oliveira Zamberlan, A., Vieira, S. A. G., Kurtz, G. C., and Frohlich, R. (2021). Proposta de uma plataforma de cloud computing para disponibilização de um sistema online para consultórios e clínicas por meio do modelo saas. *Tópicos em Ciências da Saúde Volume 24*, page 7.

Framework Django (2020). Django: the web framework for perfectionists with deadlines. *Disponível em https://www.djangoproject.com*.

Laudon, J. P. and Laudon, K. C. (2014). *Sistemas de Informação Gerenciais*. Pearson, 11 edition.

Molinari, L. (2007). Gerência de configuração-técnicas e práticas no desenvolvimento do software. *Florianópolis: Visual Book*.

O'brien, J. A. and Marakas, G. M. (2013). *Administração de sistemas de informação*. AMGH, São Paulo.

Porto (2020). *Infopédia*. Porto Editora, Disponível em https://www.infopedia.pt/dicionarios/lingua-portuguesa/refatorar. Acessado em Junho de 2022.

Rezende, E. d. A. (2018). Análise de sistema de informação para controle do registro de ponto: estudo em empresa de segurança privada em belém-pa.

Santiago, C. P., Veras, N. L., de Aragão, A. P., Carvalho, D. A., and Amaral, L. A. (2020). Desenvolvimento de sistemas web orientado a reuso com python, django e bootstrap. *Sociedade Brasileira de Computação*.

Sutherland, J. (2016). *Scrum: a arte de fazer o dobro do trabalho na metade do tempo*. Leya.

Wykowski, T. and Wykowska, J. (2019). Lessons learned: Using scrum in non-technical teams.

Zamberlan, A., Perozzo, R. F., dos Santos, R. E., and Kurtz, G. C. (2021). Sistema web para gestão de documentos atas construído por muitas mãos: Experiência extensionista na computação. *Disciplinarum Scientia— Ciências Humanas*, 22(2):165–177.