Sistema Web para Gestão de Perfis em Ambientes Automatizados

Marcelo Silva de Siqueira
Curso de Sistemas de Informação
Universidade Franciscana
Santa Maria, Brasil
marcelo.siqueira@ufn.edu.br

Alexandre de Oliveira Zamberlan
Curso de Sistemas de Informação
Universidade Franciscana
Santa Maria, Brasil
alexz@ufn.edu.br

***Resumo*—Este trabalho visa complementar o sistema projetado e implementado por Aloísio Kneipp dos Santos [1], nominado de "Sistemas Pervasivos Integrados por Agentes Inteligentes em Jason e Raspberry Pi", via o projeto e a implementação de um sistema Web para gestão de usuários e seus perfis em ambientes, como cômodos em uma casa, no que se refere a preferências de iluminação e temperatura. O sistema Web integra dados de preferências e de usuários na simulação criada por meio de um Web Service. Para isso, as tecnologias Python, Django e Bootstrap foram utilizadas para a construção dessa plataforma. Os resultados gerados foram a análise de trabalhos relacionados, modelagem funcional e estrutural do sistema, além de um Portal Web para gestão de usuários, equipamentos, relés, cômodos, perfis e geração de arquivos ao Web Service.**

***Index Terms*—Sistema de Informação; Python; Django.**

I. Introdução

A simulação criada por Santos [1] avalia a relação de usuários e suas preferências (iluminação e temperatura) em cômodos de uma casa, por exemplo. Nesse trabalho, foi realizada a integração da teoria BDI (*Belief, Desire, Intention*) de agentes inteligentes com os fundamentos da Computação Pervasiva, no contexto da "Internet das Coisas"(do inglês, *Internet of Things* - IoT), em que dispositivos pudessem comunicar-se entre si de forma inteligente, autônoma, proativa e flexível. O trabalho foi composto de uma simulação de solução computacional que forneceu integração entre dispositivos Raspberry Pi e a tecnologia de Sistemas Multiagentes implementados em Jason.

Contudo, a pesquisa feita por Santos [1], trouxe como trabalhos futuros, a gestão de usuários, ambientes e perfis específicos desses usuários em ambientes pervasivos. Nesse sistema, deveria ser possível associar usuário a uma residência ou a um ambiente comercial e subdividi-lo em salas ou unidades. Cada sala, o usuário poderia especificar um perfil de climatização e de iluminação, tendo como parâmetros intensidade da luz, temperatura, etc.

Dessa forma, o objetivo geral deste trabalho foi projetar, desenvolver e implantar um sistema Web para gerenciar usuários, ambientes e perfis específicos dos usuários. Para que o objetivo geral fosse alcançado, identificaram-se alguns objetivos específicos:

- entender e testar a simulação criada por Santos [1], bem como todos os processos modelados;
- investigar como integrar usuários e seus perfis no sistema Web no ambiente de simulação criado (*Web service*);
- entender e testar o desenvolvimento de sistemas Web via tecnologia Python-Django-Bootstrap;
- mapear e compilar trabalhos relacionados.

II. Revisão bibliográfica

Esta seção trata dos principais fundamentos e processos trabalhados no texto. Inicialmente, apresenta os temas foco da pesquisa e na sequência as tecnologias necessárias para a construção da solução.

A. Computação em Nuvem, Internet das Coisas, Automação e Web service

É fato que muitos sistemas já executam via a Internet, como sistemas (*online*) distribuídos geograficamente. Dessa forma, esses sistemas estão em nuvem, pois executam em navegadores ou em aplicativos móveis. Esse conceito de Computação em Nuvem, ou *Cloud Computing*, refere-se ao processamento computacional e armazenamento de dados que são executados fora da infraestrutura local. Com isso, o processamento e armazenamento passa a ser em um servidor, não mais na máquina do cliente, permitindo manutenção e garantias de segurança do sistema em um ambiente centralizado, seguro e controlado [2].

Do inglês, *Internet of Things* (IoT), trata-se de um ecossistema ou uma rede de objetos (coisas ou dispositivos), que coleta, processa e compartilha dados via protocolos de comunicação da Internet (modelo TCP/IP) [3]. Isso se dá por meio de sensores e atuadores instalados nos ambientes, que monitoram dados e executam ações, respectivamente.

IoT trata duas categorias de computação: Pervasiva e Ubíqua, ambas disponíveis em qualquer lugar, a qualquer tempo, e que o usuário possa usar qualquer dispositivo para ter acesso ao seu ambiente computacional. Dessa forma, quanto mais móvel, mais ubíquo, quanto mais fixo, mais pervasivo [1].

A Internet das Coisas pode ser utilizada para uso empresarial, doméstico, industrial, nos processos de tomada de decisão, entre outros. Por exemplo, uma sala com um sistema de iluminação que se adapta (intensidade) a um celular (pertencente a uma pessoa) que entra no ambiente. Essa pessoa

configurou previamente seu perfil de iluminação para um determinado horário e naquela sala. Essa configuração de perfil foi transferida para o celular, que funcionou como um dispositivo de comunicação entre o ambiente e sistema de iluminação. Outro exemplo, essa mesma pessoa, para esse mesmo celular, para essa mesma sala, configurou um perfil de climatização. Uma vez que o sensor do ambiente percebe o celular dessa pessoa, é capaz de processar o perfil climático dela e atuar configurando o sistema de ar condicionado para entrar na temperatura personalizada.

Essa tecnologia, apesar dos benefícios nos processos de automação, apresenta pontos de reflexão, como a invasão de privacidade. Isso se dá, porque o contexto IoT pode recolher dados diversos, algumas vezes sem a autorização da pessoa que entrou no ambiente. Outro ponto de discussão é a segurança da conexão entre os dispositivos, principalmente quando ocorre uma conexão de atuação, em que processos podem alterar dispositivos a partir de programações de pessoas mal intencionadas.

É considerado um padrão de comunicação e integração de sistemas, permitindo que diferentes aplicações independente de plataforma compartilhem e troquem dados pela internet [2]. De acordo com Costa e Zamberlan [2], *Web service* é uma combinação de hardware e software, que possibilita que aplicações (homogêneas e/ou heterogêneas) possam trocar dados entre si, de maneira natural e transparente, sem a necessidade de protocolos complexos de comunicação. Há diferentes formas de se construir *Web Service*, mas em geral Extensible Markup Language (XML), JavaScript Object Notation (JSON) e Comma-Separated Values (CSV) são as mais utilizadas. Esses são documentos de marcação que dados podem estar etiquetados (identificados) ou separados, só que ao invés de estarem em tabelas de Bancos de Dados, estão em arquivos de texto puro, tendo um comportamento de base de dados. Entretanto, essa base é processada não com uma linguagem de manipulação de Banco de Dados, mas sim processada via métodos e atributos de classes da teoria da Orientação a Objetos.

B. Frameworks, Python, Django e Bootstrap

Um *framework* é uma ferramenta que funciona como facilitador ou agilizador no momento de desenvolver um software. Ele é formado por um conjunto de pacotes, classes e métodos implementados fornecendo recursos prontos para que não seja necessário desenvolver uma funcionalidade ou uma estrutura do zero [4]. Como mencionado, é um facilitador que tem como filosofia a reutilização de códigos com minímas alterações, produzindo uma nova e diferente funcionalidade, por exemplo. Dessa forma, diminuindo tarefas repetitivas utilizando de códigos disponíveis e já testados (consolidados) [4].

O uso de *framework* gera agilidade em confeccionar código, reduz esforço, segue boas práticas e/ou padrões, garantindo um software consistente, seguro, enxuto, etc. Entretanto, o uso incorreto de um *framework* no projeto pode gerar um código confuso e de processamento mais lento.

Python é uma linguagem de programação de alto nível, gratuita, interpretada, multiplataforma, tipada dinamicamente e orientada a objetos. É uma das linguagens mais demandadas por empresas e organizações do mundo inteiro por ser simples de aprender e aplicar, ter uma grande comunidade engajada e contar com muitas bibliotecas e *frameworks* [5].

Django é um *framework* gratuito para desenvolvimento rápido e seguro de aplicações na *Web* escrito em Python. Fornece inúmeros modelos de *design*, *drivers* de conexão com banco de dados, de processos de validação (como *login* e senhas), de aplicativos com CRUD *(Create, Retrieve, Update, Delete)* completo. Trabalha no modelo MVT (Model-View-Template):

- *Model*: responsável pelo mapeamento do banco de dados via classes e objetos do sistema;
- *Template*: interface (página HTML - visual) que o usuário vai interagir com a aplicação;
- *View*: parte lógica do sistema, onde se especifica a interação da parte visual com o modelo de dados, ou seja, as regras do negócio.

Além dessas características, Django inclui vários recursos e funcionalidades prontas que são facilmente incorporadas à aplicação, reduzindo consideravelmente o tempo de desenvolvimento e as linhas de código [6].

C. Nginx, Gunicorn e SQLite

O paradigma de computação em nuvem pode ser definido como um conjunto de recursos com capacidade de processamento, armazenamento, conectividade, plataformas, aplicações e serviços disponibilizados na Internet. Uma arquitetura em nuvem é formada por múltiplos servidores. Entre os vários conceitos que abrangem a computação em nuvem destacam-se balanceamento de carga, banco de dados aglomerados e escalonamento de armazenamento. Nesse quesito de balanceamento de carga, surge então o serviço NGINX, que faz a distribuição de requisições de maneira equitativa sobre os nós de um ambiente distribuído [7]. Em síntese, o NGINX é em uma aplicação *Web* o responsável por gerenciar a disponibilidade da aplicação ao usuário.

Além disso, esse serviço oferece certificado digital SSL (*Secure Sockets Layer*), permitindo que o tráfego entre o cliente final e o servidor seja utilizando conexão segura através do protocolo HTTPS (*Hypertext Transfer Protocol Secure*). O NGINX também foi projetado para servir e fazer cacheamento de arquivos estáticos como imagens, CSS (*Cascading Style Sheets*) e Javascript, além de servir o conteúdo dinâmico encaminhado pelo GUNICORN.

Por sua vez, GUNICORN é considerado um servidor Python para páginas e/ou sistemas via protocolo HTTP para sistemas UNIX (Linux). Ou seja, ele resolve de forma estática e dinâmica as solicitações HTTP de clientes via seus navegadores. Esse serviço é compatível com várias estruturas *Web*, com uso de recursos otimizados [7].

SQLite é um Sistema de Gerenciamento de Banco de Dados (SGBD) relacional, porém simples, portável e leve. Além disso, não exige uma configuração complexa ou um

servidor separado para sua operação. É o SGBD embutido no *framework* Django, muito útil para o processo de desenvolvimento. O SQLite, diferentemente dos SGBD tradicionais, armazena o banco de dados em um arquivo, que pode ser movido ou compartilhado entre sistemas. Finalmente, o SQLite oferece recursos consistentes, como suporte transacional, integridade referencial, índices, criptografia, disparadores (*triggers*) e visões (*views*).

D. Scrum

De acordo com Sutherland [8], Scrum é uma estrutura ágil de gerenciamento de projetos criada no início dos anos 90, considerada um *framework* flexível e funcional, não apenas para desenvolvimento de software, como também útil para qualquer trabalho em equipe que necessite ser gerido desta forma.

Na prática, o Scrum funciona da seguinte forma: mantém-se uma programação regular de pequenas reuniões (*sprints*), com a finalidade de inspecionar o projeto em andamento, descobrir possíveis falhas ou erros, a fim de corrigi-los com agilidade e fazer entregas ao cliente sempre no prazo combinado. Essa filosofia de trabalho garante maior comunicação entre colaboradores e clientes; promove transparência e visibilidade, ou seja, todos os integrantes do projeto sabem o que cada membro está realizando ou executando. Com isso, a satisfação do cliente é considerável, primeiro que participa realmente do processo de construção da solução, segundo porque recebe com frequência protótipos e/ou produtos estáveis para avaliar.

Conforme Palharini [9], essa metodologia começa com o *Product Owner* determinando o que deve e não deve fazer parte do produto final. Além de definir os requisitos do projeto, o *Product Owner* elabora uma lista com todas as exigências e implementações que o produto/serviço final deve apresentar, o que se chama de *Product Backlog*. O *Product Backlog* necessita obedecer uma ordem de prioridades. Após os requisitos serem mapeados, a equipe de desenvolvimento planeja os *Sprints*. Após o término de cada *Sprint*, é realizada uma retrospectiva para analisar e discutir o que pode ser melhorado [9].

E. Trabalhos Relacionados

No trabalho [1] realizou a integração da teoria BDI (*Belief, Desire, Intention*) de agentes inteligentes com os fundamentos da Computação Pervasiva, no contexto da "Internet das Coisas", em que dispositivos pudessem comunicar-se entre si de forma inteligente, autônoma, proativa e flexível. O trabalho gerou uma simulação de solução computacional que fornece integração, no contexto IoT, entre dispositivos Raspberry Pi e a tecnologia de Sistemas Multiagentes implementados em Jason. Nessa pesquisa, foram utilizados a metodologia PROMETHEUS para modelagem de Sistemas Multiagentes, as linguagens de programação Java e AgentSpeak(L), o interpretador Jason e o computador Raspberry Pi.

Já no trabalho [9], foi apresentado uma solução Web, construída com Python, Django e Bootstrap para gestão informatizada de Trabalhos Finais de Graduação, com mecanismos de busca de assuntos e/ou tecnologias trabalhadas em pesquisas conduzidas por alunos e seus orientadores. Essa pesquisa utilizou a metodologia SCRUM com o suporte da técnica Kanban para gestão de atividades.

O trabalho [2] faz parte das pesquisas realizadas no *Multi-agent System for Polymeric Nanoparticles* (MASPN), que é composto por uma ferramenta de simulação e um *website*. Dessa forma, esse trabalho teve o foco no projeto e na implementação de uma integração dos dois ambientes, um feito para *Desktop* (Java) e outro para Internet (Python, Django e MySQL). O portal Web gerencia fármacos, pesquisadores, polímeros, experimentos com nanopartículas poliméricas, já na ferramenta MASPN, a partir desses dados cadastrados no portal, possibilita a simulação do comportamento das nanopartículas poliméricas. Para a construção da integração, *Web Service*, foram utilizadas as tecnologias JSON, mysqldump, mysql2sqlite, dumpdata, Python, Django e Java, todas contextualizadas em boas práticas do *framework* REST.

Finalmente, o trabalho [3] discorre sobre o contexto da Internet das Coisas, apontando evolução da área, seus impactos e benefícios, abordando a conectividade de diversos equipamentos com a Internet. A metodologia da pesquisa utilizada foi o levantamento bibliográfico com enfoque na pesquisa exploratória e analítica com abordagem qualitativa. Com isso, buscou-se auxiliar na compreensão dos princípios da IoT.

Dos trabalhos pesquisados, destacam-se de Santos [1], de Palharini [9] e de Costa [2], uma vez que o primeiro é a base desta pesquisa, o segundo possui todas as estratégias, tecnologias e boas práticas de desenvolvimento de sistema Web via tecnologia Python-Django, que são as utilizadas nesta pesquisa, e o terceiro tem o passo-a-paso para a construção de um recurso de integração de sistemas heterogêneos via *Web Service*, como desejado nesta pesquisa.

III. Metodologia e proposta de trabalho

Este trabalho é um estudo de caso, com desenvolvimento de um sistema Web e um sistema de integração (*Web Service*), tendo como referência a pesquisa realizada em [1]. Já no projeto e desenvolvimento do Sistema Web, foram utilizados a metodologia SCRUM [8] com a técnica Kanban para gestão de atividades, prazos e responsáveis. As ferramentas utilizadas foram:

- ambiente de aplicação da técnica Kanban: Trello;
- ambiente de diagramação *Unified Language Model* (UML): Astah;
- linguagem de programação Python e seus *frameworks* Bootstrap e Django para o sistema Web;
- serviços Gunicorn e Ngnix em sistema operacional Linux;
- sistema gerenciador de banco de dados: SQLite;
- ambiente de versionamento de código: Github;
- ambiente de desenvolvimento: Visual Studio Code e suas extensões Python-Django.

Seguindo as boas práticas da metodologia, e sempre com apresentação e discussão das atividades desenvolvidas, as atividades *Product Backlog* foram geridas pela ferramenta *Web* Trello (técnica Kanban) para controle de prazos. O aluno teve

o papel de desenvolvedor do sistema. O professor assumiu o papel de *Scrum Master* e de desenvolvedor.

Registra-se que os processos de verificação e de validação, conforme [10], têm como objetivo assegurar que um sistema seja adequado e se atende às necessidades ou às especificações levantadas no processo de mapeamento de requisitos. A verificação tem foco nos requisitos funcionais e não funcionais dos sistema, enquanto a validação certifica-se se o sistema atende as necessidades e expectativas do cliente. Por fim, esses processos não são separados e independentes [10].

A. Proposta do sistema

Nesta seção, buscou-se apresentar detalhes de alguns aspectos funcionais e estruturais. A Figura 1 ilustra os atores e as principais funcionalidades do sistema. O diagrama apresenta três pacotes: i) sistema pervasivo proposto por [1], que é a simulação projetada, implementada e executada, como mencionado na seção Trabalhos Relacionados; ii) o ambiente de simulação construído na ferramenta Jason; iii) sistema proposto, em que vale ressaltar que o componente *Web Service* presente no outro pacote, faz parte deste trabalho. Buscou-se ilustrar as funcionalidades para gestão de usuários, cômodos e perfis, sendo que ao final, os dados cadastrados estarão disponíveis ao simulador via o *Web Service*. A ideia inicial é gerar conteúdo ao *Web Service* sempre que uma simulação for iniciada.

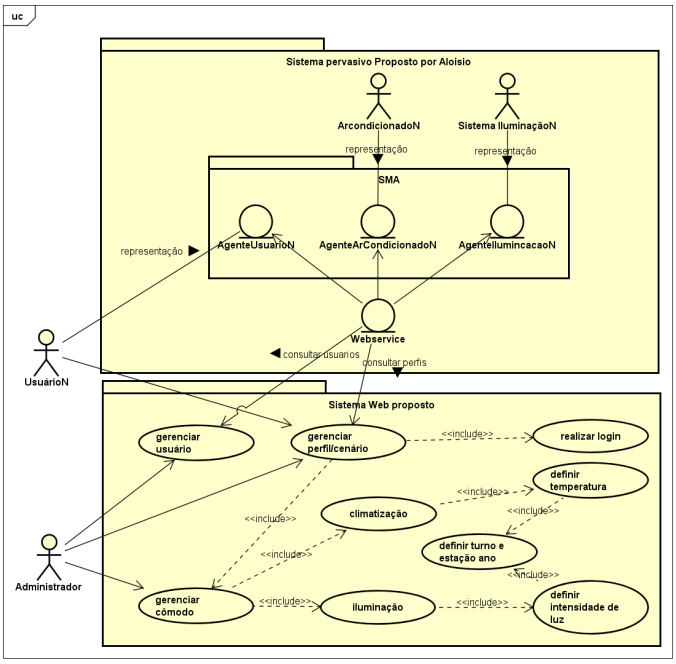

Figura 1. Diagrama de Casos de Uso ilustrando tanto as funcionalidades, quanto a ideia geral da arquitetura da integração do simulador com o sistema Web.

Um diagrama importante em relação aos aspectos funcionais, é o diagrama de atividades, que ilustra o fluxo de trabalho pretendido com o sistema (Figura 2). Nesse diagrama, foi apresentado a forma como a simulação faz a relação ou comunicação com o sistema proposto. Destaca-se que a simulação é independente do sistema.

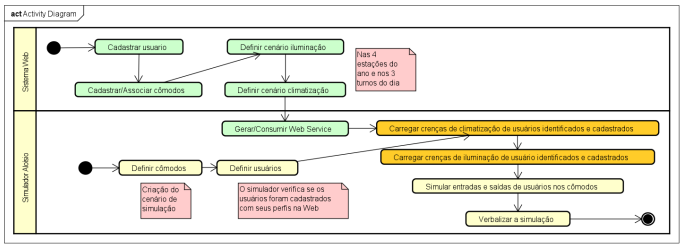

Figura 2. Diagrama de Atividade.

Uma vez definidas e especificadas as funcionalidades, segue a descrição de aspectos estruturais. Um diagrama importante para isso é o de Classe (Figura 3).

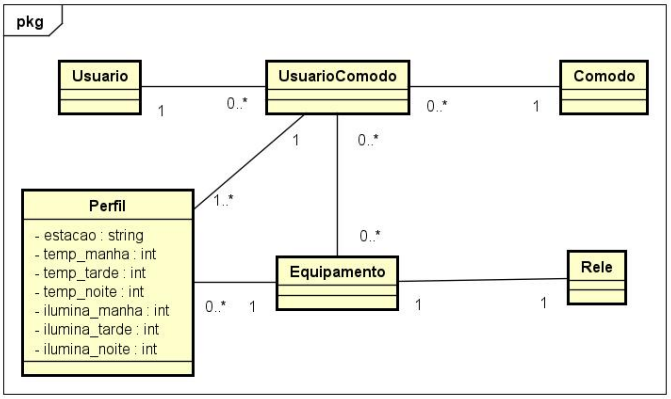

Figura 3. Diagrama de Classe.

No diagrama, é possível destacar as classes:

- Usuario: representando os usuários de um cômodo;
- Comodo: com dados para descrever e identificar a peça em uma casa ou escritório;
- Equipamento: representando um equipamento como ar condicionado ou como sistema de iluminação;
- UsuarioComodo: representa a relação do usuário com um cômodo, ou seja, qual o nível de importância que um determinado tem sobre um cômodo. Isso é importante, pois quando mais de um usuário entrarem no cômodo, o simulador precisa entender que tem prioridade de escolha;
- Perfil: representa os desejos de um usuário sobre um cômodo, separando por estação do ano, temperaturas para os três turnos e intensidade de iluminação, também para os três turnos. N
- Rele: representa os dados do relé de acionamento que recebe ações do simulador para disparar algum equipamento.

Por exemplo, um usuário, tem preferências de temperatura e iluminação para a sua sala de jantar, nas quatro estações, além

de cada turno do dia, com temperatura específica e intensidade de iluminação para cada turno.

B. Avaliação do sistema proposto

O processo de simulação criado em Santos [1], gera uma dinâmica (via Jason) em que usuários entram e saem de um ambiente (sala ou quarto, por exemplo) e que esses ambientes percebem (via sensores) esses usuários, capturando seus perfis, com preferências para climatização e/ou iluminação, atuando (via atuadores) nos sistemas de iluminação e climatização, configurando assim, temperatura e intensidade de iluminação. Diferente do que ocorria na simulação [1], em que os usuários e seus perfis eram previamente cadastrados na simulação (no ambiente do Jason), agora, os usuários e seus perfis estão cadastrados no sistema Web proposto e são compartilhados ou consumidos (seus dados) com a simulação via um *Web Service*.

A ideia do fornecimento dos dados dos usuários ao simulador é via *Web Service*, que é gerado de tempos em tempos pelo sistema Web e consumido pelo simulador. Uma outra alternativa é, quando o simulador entrar em funcionamento, ele solicita ao sistema Web que gere o serviço.

C. Resultados e discussões

Os resultados do trabalho referem-se a modelagem dos aspectos funcionais e estruturais do sistema proposto. Os aspectos funcionais podem ser visualizados em Diagramas de Atividades, figura 2 e de Casos de Uso, figura 1. Quando questões funcionais são modeladas ou mapeadas, destacam-se os atores que fazem relação com o sistema, as funcionalidades ou serviços que o sistema deve atender e o fluxo de funcionamento ou de interação do sistema com esses atores. Também foi modelado um aspecto estrutural, via diagrama de componentes (classe), em que foi possível identificar as classes, que o sistema Web deve tratar.

Como o sistema foi construído via a linguagem Python e o *framework* Django, cada uma dessas classes deve ser um aplicativo (*app*) do sistema, contendo as três camadas do modelo MVT: *model* (classes e futuramente tabelas); *view* (todas as regras do negócio, em especial os métodos *Create, Retrieve, Update, Delete* - CRUD); *template* (arquivos HTML e CSS de comunicação com o usuário).

Também é possível registrar como resultados, as principais telas do sistema. A Figura 4 apresenta a tela de *login* combinada com uma *landpage*, contendo informações do projeto. Na Figura 5, é possível visualizar a gestão de usuários do sistema, bem como o menu de serviço, com a possibilidade de gestão de equipamentos (Figura 6), relés (Figura 7), cômodos (Figura 8), relação de usuários e cômodos (Figura 9).

Já a Figura 10 mostra o formulários para a configuração da iluminação e da climatização preferencial de um usuário a um cômodo.

As Figuras 11 e 12 mostram, respectivamente, a geração dos arquivos .asl (crenças) e .json (dados da base) das preferências de climatização e de iluminação de um usuário. E são esses os arquivos que devem ser disponibilizados e consumidos no *Web Service*.

Na Figura 13, há o código em que os arquivos .asl e

Figura 4. Login e Landpage do sistema.

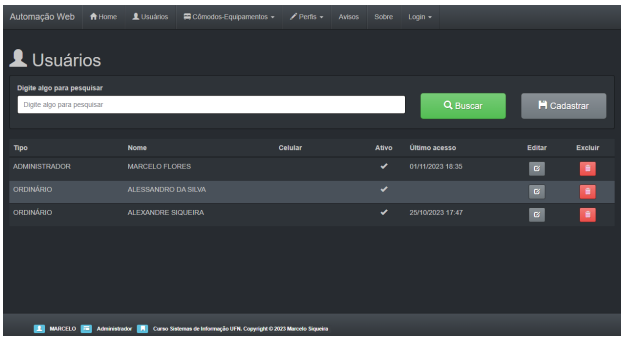

Figura 5. CRUD de Usuários.

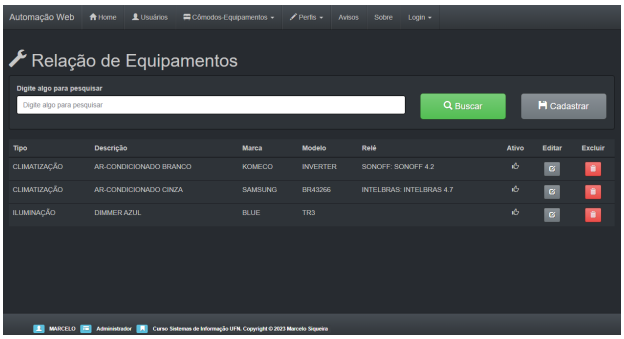

Figura 6. CRUD de Equipamentos.

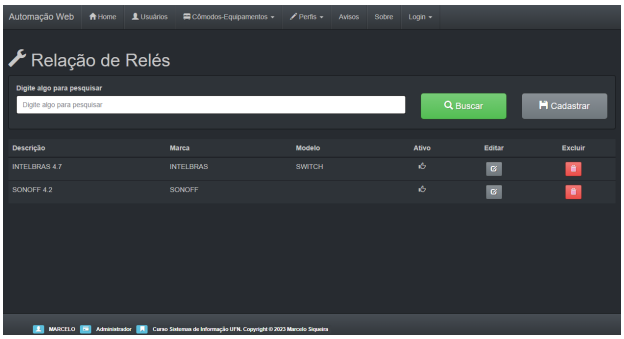

Figura 7. CRUD de Relés.

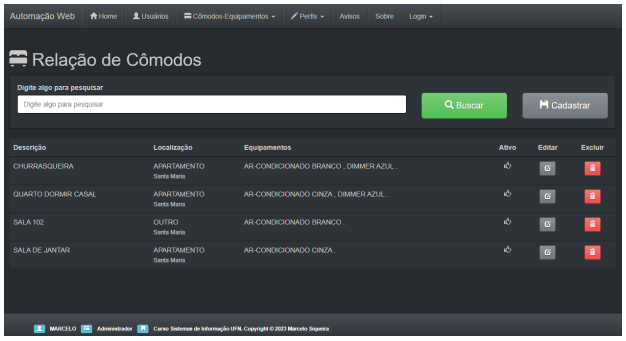

Figura 8. CRUD de Cômodos.

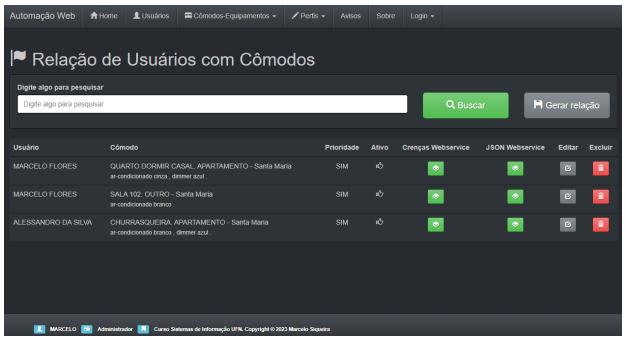

Figura 9. CRUD de Usuários-Cômodos.

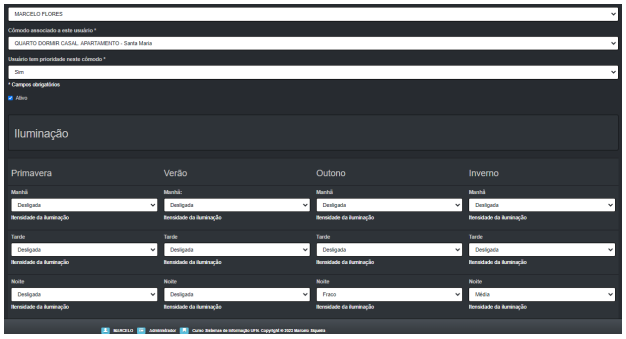

Figura 10. *Form* para configurar iluminação e climatização de Usuário a um Cômodo.

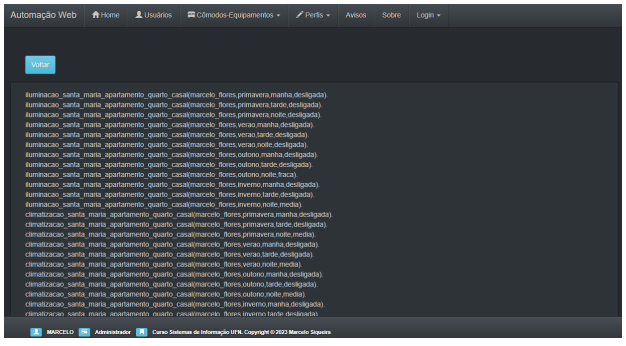

Figura 11. Arquivo .asl com crenças de iluminação e climatização de um cômodo para usuário.

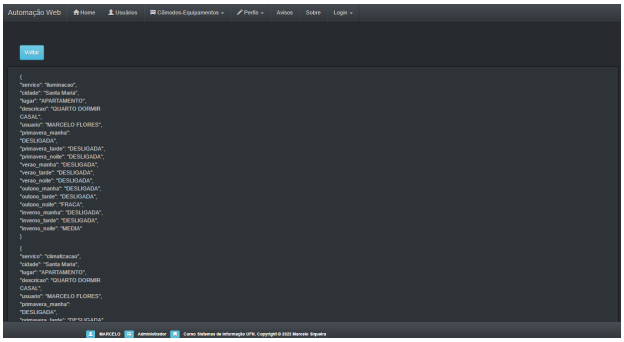

Figura 12. Arquivo .json com dados da base de iluminação e climatização de um cômodo para usuário.

.json são criados e disponibilizados no servidor. Aqui, há um ponto absolutamente importante, que é a criação de um usuário (previamente) no servidor com permissões somente para o serviço de transferência de arquivos pelo protocolo sftp (Secure File Transfer Protocol).

```
82    def criar_arquivo_webservice(self, texto, usuario, tipo):
83        # print(platform.system(), platform.release())
84        try:
85            if ('Windows' in platform.system()):
86                nome_arquivo = "projeto/uploads/"+usuario + "." + tipo
87            else:
88                nome_arquivo = "/home/webservice/webservice/"+usuario + "." + tipo
89            # print(nome_arquivo)
90            writer = open(nome_arquivo, "a", encoding='utf8')
91            writer.write(texto)
92            writer.close()
93        except:
94            print("Problemas para salvar os arquivos....\n")
95
```

Figura 13. Função Python, na camada de modelo do app perfil, para gerar o arquivo *Web service*, tanto .asl (crenças), quanto .json (base).

Na Figura 14, há o código de geração de estrutura json para iluminação e climatização. Nesse código, utilizou-se a biblioteca json do Python e o método *dumps* para a conversão (linhas 137 e 138).

Finalmente, a Figura 14 mostra código de geração de estrutura de crenças para iluminação e climatização.

```
def gerar_json_webservice(self):
    crenca_iluminacao = {
        "servico"   : "iluminacao",
        "cidade"    : self.comodo.cidade,
        "lugar"     : self.comodo.lugar,
        "descricao" : self.comodo.descricao,
        "usuario"   : self.usuario.nome,
        "primavera_manha"   : self.primavera_iluminacao_manha,
        "primavera_tarde"   : self.primavera_iluminacao_tarde,
        "primavera_noite"   : self.primavera_iluminacao_noite,
        "verao_manha"   : self.verao_iluminacao_manha,
        "verao_tarde"   : self.verao_iluminacao_tarde,
        "verao_noite"   : self.verao_iluminacao_noite,
        "outono_manha"   : self.outono_iluminacao_manha,
        "outono_tarde"   : self.outono_iluminacao_tarde,
        "outono_noite"   : self.outono_iluminacao_noite,
        "inverno_manha"   : self.inverno_iluminacao_manha,
        "inverno_tarde"   : self.inverno_iluminacao_tarde,
        "inverno_noite"   : self.inverno_iluminacao_noite,
    }

    crenca_climatizacao = {
        "servico"   : "climatizacao",
        "cidade"    : self.comodo.cidade,
        "lugar"     : self.comodo.lugar,
        "descricao" : self.comodo.descricao,
        "usuario"   : self.usuario.nome,
        "primavera_manha"   : self.primavera_climatizacao_manha,
        "primavera_tarde"   : self.primavera_climatizacao_tarde,
        "primavera_noite"   : self.primavera_climatizacao_noite,
        "verao_manha"   : self.verao_climatizacao_manha,
        "verao_tarde"   : self.verao_climatizacao_tarde,
        "verao_noite"   : self.verao_climatizacao_noite,
        "outono_manha"   : self.outono_climatizacao_manha,
        "outono_tarde"   : self.outono_climatizacao_tarde,
        "outono_noite"   : self.outono_climatizacao_noite,
        "inverno_manha"   : self.inverno_climatizacao_manha,
        "inverno_tarde"   : self.inverno_climatizacao_tarde,
        "inverno_noite"   : self.inverno_climatizacao_noite,
    }
    json_crenca_iluminacao = json.dumps(crenca_iluminacao, indent=4)
    json_crenca_climatizacao = json.dumps(crenca_climatizacao, indent=4)

    #gerar o arquivo crencas_usuario.json na pasta do usuario webservice
    resposta = json_crenca_iluminacao + "\n\n" + json_crenca_climatizacao + "\n"
    nome_usuario = self.usuario.nome.split()[0].lower() + '_' + self.usuario.nome.split()[-1].lower()
    self.criar_arquivo_webservice(resposta, nome_usuario, "json")

    return resposta
```

Figura 14. Função Python, na camada de modelo do app perfil, para gerar a estrutura json dos dados de iluminação e de climatização.

```
def gerar_crenca_webservice(self):
    vetor_cidade = self.comodo.cidade.lower().split(" ")
    if len(vetor_cidade) > 1:
        cidade = vetor_cidade[0] + "_" + vetor_cidade[-1]
    else:
        cidade = vetor_cidade[0]

    vetor_lugar = self.comodo.lugar.lower().split(" ")
    if len(vetor_lugar) > 1:
        lugar = vetor_lugar[0] + "_" + vetor_lugar[-1]
    else:
        lugar = vetor_lugar[0]

    vetor_descricao = self.comodo.descricao.lower().split(" ")
    if len(vetor_descricao) > 1:
        descricao = vetor_descricao[0] + "_" + vetor_descricao[-1]
    else:
        descricao = vetor_descricao[0]

    usuario = str(self.usuario.get_primeiro_nome.lower()) + '_' + str(self.usuario.get_sobrenome.lower())

    crenca = ''
    #iluminacao nas 4 estacoes
    crenca += 'iluminacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',primavera,manha,'+self.primavera_iluminacao_manha.lower() + ').\n'
    crenca += 'iluminacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',primavera,tarde,'+self.primavera_iluminacao_tarde.lower() + ').\n'
    crenca += 'iluminacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',primavera,noite,'+self.primavera_iluminacao_noite.lower() + ').\n'
    crenca += 'iluminacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',verao,manha,'+self.verao_iluminacao_manha.lower() + ').\n'
    crenca += 'iluminacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',verao,tarde,'+self.verao_iluminacao_tarde.lower() + ').\n'
    crenca += 'iluminacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',verao,noite,'+self.verao_iluminacao_noite.lower() + ').\n'
    crenca += 'iluminacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',outono,manha,'+self.outono_iluminacao_manha.lower() + ').\n'
    crenca += 'iluminacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',outono,tarde,'+self.outono_iluminacao_tarde.lower() + ').\n'
    crenca += 'iluminacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',outono,noite,'+self.outono_iluminacao_noite.lower() + ').\n'
    crenca += 'iluminacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',inverno,manha,'+self.inverno_iluminacao_manha.lower() + ').\n'
    crenca += 'iluminacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',inverno,tarde,'+self.inverno_iluminacao_tarde.lower() + ').\n'
    crenca += 'iluminacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',inverno,noite,'+self.inverno_iluminacao_noite.lower() + ').\n'
    #climatizacao nas 4 estacoes
    crenca += 'climatizacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',primavera,manha,'+self.primavera_climatizacao_manha.lower() + ').\n'
    crenca += 'climatizacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',primavera,tarde,'+self.primavera_climatizacao_tarde.lower() + ').\n'
    crenca += 'climatizacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',primavera,noite,'+self.primavera_climatizacao_noite.lower() + ').\n'
    crenca += 'climatizacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',verao,manha,'+self.verao_climatizacao_manha.lower() + ').\n'
    crenca += 'climatizacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',verao,tarde,'+self.verao_climatizacao_tarde.lower() + ').\n'
    crenca += 'climatizacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',verao,noite,'+self.verao_climatizacao_noite.lower() + ').\n'
    crenca += 'climatizacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',outono,manha,'+self.outono_climatizacao_manha.lower() + ').\n'
    crenca += 'climatizacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',outono,tarde,'+self.outono_climatizacao_tarde.lower() + ').\n'
    crenca += 'climatizacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',outono,noite,'+self.outono_climatizacao_noite.lower() + ').\n'
    crenca += 'climatizacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',inverno,manha,'+self.inverno_climatizacao_manha.lower() + ').\n'
    crenca += 'climatizacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',inverno,tarde,'+self.inverno_climatizacao_tarde.lower() + ').\n'
    crenca += 'climatizacao_' + cidade + '_' + lugar + '_' + descricao + '(' + usuario + ',inverno,noite,'+self.inverno_climatizacao_noite.lower() + ').\n'

    #gerar o arquivo crencas_usuario.asl na pasta do usuario webservice
    self.criar_arquivo_webservice(crenca, usuario, "asl")

    return crenca
```

Figura 15. Função Python, na camada de modelo do app perfil, para gerar a estrutura de crenças asl de iluminação e de climatização.

IV. Conclusões

Ao longo do texto, foram discutidos os principais assuntos que norteiam este trabalho, como Computação em Nuvem, IoT, *Web service*, *Frameworks*, os serviços NGINX e GUNICORN, a metodologia Scrum e uma avaliação dos trabalhos relacionados.

Na sequência modelou-se o sistema para gestão de perfis em ambientes automatizados, tendo como base a simulação realizada por Santos [1]. Dessa modelagem, destacam-se diagramas dos aspectos funcionais e estruturais, que dão suporte para o entendimento de como o sistema deve ser construído: quais seus atores, quais funcionalidades, quais componentes, como classes, etc.

Também, foi implementado e disponibilizado um Sistema Web para a gestão de usuários, equipamentos, cômodos, relés e perfis (http://automacaoweb.techvictris.com.br/). No servidor, um usuário especial foi criado e configurado para receber os arquivos .asl e .json, de forma que conexões sftp sejam realizadas a partir do simulador. Dessa forma, a relação de usuários e suas preferências são compartilhadas com o simulador. Os códigos do sistema estão em disponíveis de forma gratuita em https://github.com/M-Siqueira/TFG_Marcelo.

Finalmente, registram-se algumas possibilidades de trabalhos futuros, como:

- a partir dos dados armazenados no Portal, implementar técnicas de mineração ou de descoberta de conhecimento para identificar padrões de comportamento, e, atém mesmo, recomendar melhores configurações de perfis;
- realizar testes no simulador proposto em Santos [1] de forma que ele consumo o *Web Service* entregue;
- identificar outros trabalhos que poderiam receber tal integração.

Referências

[1] A. K. D. Santos and A. Zamberlan, *Sistemas Pervasivos Integrados Por Agentes Inteligentes Em Jason E Raspberry Pi*. Santa Maria, RS, Brasil. Disponível em https://tfgonline.lapinf.ufn.edu.br: Trabalho de Conclusão de Curso Ciência Da Computação - Universidade Franciscana (UFN), 2021.

[2] Éderson Guterres Costa and A. Zamberlan, *Webservice Para Integração De Dados Entre A Ferramenta De Simulação Maspn E Seu Portal Web*. Santa Maria, RS, Brasil. Disponível em https://tfgonline.lapinf.ufn.edu.br: Trabalho de Conclusão de Curso Sistemas De Informação - Universidade Franciscana (UFN), 2021.

[3] M. G. De Godoi and L. S. Araújo, "A internet das coisas: evolução, impactos e benefícios," *Revista interface tecnológica*, vol. 16, no. 1, pp. 19–30, 2019.

[4] R. Pressman and B. Maxim, *Engenharia de Software-8ª Edição*. McGraw Hill Brasil, 2016.

[5] Programming language python. [Online]. Available: http://www.python.org/about/

[6] Framework django. [Online]. Available: https://www.djangoproject.com/

[7] R. E. dos Santos, A. de Oliveira Zamberlan, S. A. G. Vieira, G. C. Kurtz, and R. Frohlich, "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*, p. 7, 2021.

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

[9] E. Palharini and A. Zamberlan, *TFG Online: Ambiente para gestão eletrônica de trabalhos finais de graduação*. Disciplinarum Scentia, 2021.

[10] Importância da validação e da verificação. [Online]. Available: https://www.devmedia.com.br/a-importancia-da-validacao-e-da-verificacao/24559