Francisco Antonio Oliveira Lucena

Sistema de Informatização do Processo de Aquisição dos Dados dos Agentes de Endemias - SIADE: Módulo Aplicação Web

Pau dos Ferros - RN

2015

Francisco Antonio Oliveira Lucena

Sistema de Informatização do Processo de Aquisição dos Dados dos Agentes de Endemias - SIADE: Módulo Aplicação Web

Trabalho de Conclusão de Curso submetido ao Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, em cumprimento às exigências legais como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte - IFRN

Tecnologia em Análise e Desenvolvimento de Sistemas - TADS

Orientador: Prof. Me. Demétrios A. M. Coutinho

Coorientador: Prof. Esp. Jeferson Queiroga Pereira

Pau dos Ferros - RN

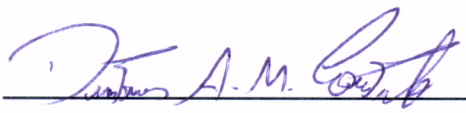
2015

Francisco Antonio Oliveira Lucena

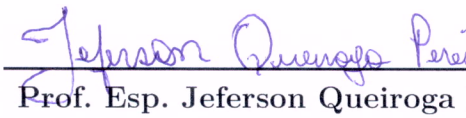
Sistema de Informatização do Processo de Aquisição dos Dados dos Agentes de Endemias - SIADE: Módulo Aplicação Web

Trabalho de Conclusão de Curso submetido ao Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, em cumprimento às exigências legais como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

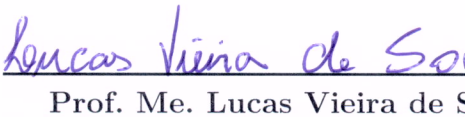
Trabalho aprovado. Pau dos Ferros - RN, 21 de Outubro de 2015:



Prof. Me. Demétrios A. M. Coutinho
Orientador
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte-IFRN



Prof. Esp. Jeferson Queiroga Pereira
Coorientador
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte-IFRN



Prof. Me. Lucas Vieira de Souza
Examinador interno
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte-IFRN

Pau dos Ferros - RN
Outubro/2015

Dedico este trabalho a minha família, em especial aos meus pais. pois, sempre me ofereceram apoio, força e não mediram esforços para a conclusão de mais uma etapa da minha vida.

Agradecimentos

Antes de tudo, é necessário agradecer a Deus por sentir sua presença sempre comigo, pela família e amigos que me concedes. Pela graça de enfrentar mais um desafio na caminhada da vida e com Ele ter chegado à mais uma conquista.

Àqueles que proporcionaram um enriquecimento intelectual e moral durante o desenrolar do curso, eternos professores, meus sinceros e eternos agradecimentos.

Em especial a dois Mestres, que além de seus ensinamentos didáticos, me motivaram a persistir sempre seguindo no melhor caminho. A vocês, devo além de tudo, carinho e admiração. Dra. Ayla Marcia Cordeiro Bezerra e Ms. Demétrios Magalhães Coutinho, vocês excederam os limites de sua profissão, foram além de tudo amigos e como costumo dizer, "Os pais que a vida me deu". Obrigado pela confiança e por fazer parte de cada momento da minha jornada.

Aos colegas de graduação, os companheiros com quem tive a satisfação de conviver por quase quatro anos, que, mesmo nos momentos árduos, não deixaram de ser irreverentes e amigos.

A vocês que me adotaram como filho e me acolheram tantas vezes quanto precisei em suas residências. Dona Cedma (Cedinha) e Dona Socorro e suas famílias. Que o grandioso Deus continue a vos abençoar.

Aos meus amigos de longa caminhada e aos adicionados em minha vida durante o desenrolar do curso. Vocês que estiveram comigo em muitas situações, dando-me força e apoio quando necessitei. Em especial a Artemio Moreira Soares (*in memorian*). Meu amigo, tu partiste tão cedo do plano terreno, mas continua vivo em nossos corações. Graças por suas amizades.

A Francisco Inácio, (*in memorian*). Meu avô "Chico Manel", agradeço por tantos ensinamentos. Sua vida terrena, foi um exemplo de luta, trabalho e amor à família. Sua conduta nunca será esquecida.

Por fim, àqueles que foram e são meu porto seguro, fonte de incentivo, minha base. Vocês que estiveram lado a lado nos momentos de dificuldade que enfrentei, tiveram a paciência e compreensão de me apoiar. Família, realmente amo vocês.

Com todos vocês, confraternizo a alegria de mais uma vitória em minha vida. Meu muito obrigado!

Bonito é melhor que feio.
Explícito é melhor que implícito.
Simples é melhor que complexo.
Complexo é melhor que complicado.
Linear é melhor do que aninhado.
Esparso é melhor que denso.
Legibilidade conta.
Casos especiais não são especiais o bastante para quebrar as regras.
Ainda que praticidade vença a pureza.
Erros nunca devem passar silenciosamente.
A menos que sejam explicitamente silenciados.
Diante da ambiguidade, recuse a tentação de adivinhar.
Deveria haver um e preferencialmente só um modo óbvio para fazer algo.
Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês.
Agora é melhor que nunca.
Embora nunca frequentemente seja melhor que "já".
Se a implementação é difícil de explicar, é uma má ideia.
Se a implementação é fácil de explicar, pode ser uma boa ideia.
Namespaces são uma grande ideia, vamos fazer mais desses!
(O Zen do Python, por Tim Peters)

Resumo

A dengue é uma doença endêmica, sendo um dos principais problemas de saúde pública no país. A política de prevenção utilizada atualmente é realizada através do trabalho dos agentes de combate a endemias. Estes são responsáveis por realizar vistorias em residências, depósitos, terrenos baldios e estabelecimentos comerciais na busca de focos endêmicos, realizam a inspeção em caixas d'água, calhas e telhados, e aplicam larvicidas e inseticidas. O processo de acompanhamento do agente é bastante oneroso e árduo, consistindo do preenchimento de formulários, onde são abordadas informações acerca das condições sanitárias do local visitado. Assim, esse projeto descreve o desenvolvimento do módulo web, do SIADE - Sistema de Informatização do Processo de Aquisição dos Dados dos Agentes de Endemias. O sistema surge para suprir a necessidade de informatizar e automatizar a geração de relatórios, visualização de dados e o controle e gerenciamento do trabalho dos agentes de endemias.

Palavras-chaves: *Framework* Django. Aplicações Web. Dengue.

Lista de ilustrações

Sumário

1 INTRODUÇÃO

A dengue constitui um dos principais problemas de saúde pública no Brasil e no mundo. O Brasil, teve a sua primeira epidemia documentada clínica e laboratorialmente nos anos de 1981-1982, na cidade de Boa Vista (RR). Em 1986, ocorreram epidemias atingindo o Rio de Janeiro e algumas capitais da região Nordeste. Desde então, a dengue vem ocorrendo no Brasil de forma continuada, intercalando-se com a ocorrência de epidemias (**??**).

Trata-se de doença viral de transmissão vetorial, e dentre estas, é a que mais causa impacto em termos de mortalidade na população mundial, fato que exige esforços e investimentos cada vez mais intensos dos serviços no combate à epidemia (**??**). O vírus é transmitido através da picada da fêmea contaminada do mosquito Aedes aegypti. Diversos fatores contribuem para o estabelecimento de uma situação epidemiológica, destacando-se: imóveis fechados; falta de saneamento básico e de políticas de combate efetivas; terrenos baldios com acúmulo de lixo; interrupção no fornecimento de água, condicionando a população a armazenar água sem os cuidados necessários; clima quente, com períodos chuvosos; dentre outros (**??**). O combate ao mosquito pode ser feito de duas maneiras: eliminando os mosquitos adultos e, principalmente, acabando com os criadouros de larvas (**??**).

Políticas públicas mais eficazes que possam auxiliar no combate ao mosquito se fazem cada vez mais necessárias, visto que, a cada ano o número de casos confirmados da doença cresce sensivelmente. Atualmente o governo dispõe de serviços do agente de combate de endemias, que realiza a vistoria em residências, depósitos, terrenos baldios e estabelecimentos comerciais para buscar focos endêmicos. Eles também realizam a inspeção em caixas d'água, calhas e telhados, aplicam larvicidas e inseticidas e ainda orientam quanto à prevenção e tratamento de doenças. Todos os dados obtidos pelo agente de endemias são registrados em planilhas para futuras análises estatísticas e controle de agentes químicos utilizados no combate.

Infelizmente esse trabalho ainda não é suficiente para controlar endemias como esta, visto que muitas vezes a população não realiza os procedimentos necessários para evitar a proliferação das larvas do mosquito.

No estado do Rio Grande do Norte, em 2013, foram 1.060 notificações de dengue contra 2.310, em 2012, uma redução de 45%. Contudo, o estado ainda possui uma das maiores incidências da doença e ocupa o segundo lugar no ranking da região Nordeste (**??**). Entre 2000 e 2011 ocorreram os seguintes números de casos: em 2011, 7.701; em 2010, 3.670; em 2009, 13.826; em 2008, 34.890; em 2007, 13.489; em 2006, 9.526; em 2005,

6.840; em 2004, 3.450; em 2003, 22.621; em 2002, 24.079; em 2001, 19.221; e em 2000, 8.151 casos de dengue em todo o estado (**??**). Nota-se uma oscilação entre as notificações de números de casos no estado, e diante desse fato, há a necessidade de se definir políticas administrativas orientadas ao efetivo controle da doença.

O processo de acompanhamento do agente de endemias é bastante oneroso e árduo, consistindo do preenchimento de formulários, onde são abordadas informações acerca das condições sanitárias do local visitado, inspeção sobre a existência de possíveis focos do mosquito e informações sobre aplicação de larvicidas, sua quantidade e tipo. Esses formulários após preenchidos são encaminhados à um supervisor que faz a inspeção e confirmação daqueles dados obtidos.

O trabalho de um agente de endemias na parte do preenchimento de formulários se baseia nos seguintes documentos: Registro Geral de Quarteirão (RG), que contém o registro de casas existentes no quarteirão; Formulário de Trabalho Diário (D1), que contém várias informações que são coletadas ao visitar as residências e outras que são preenchidas através dos dados que estão no RG; e ainda o Resumo Semanal (D7), que é resumo semanal contendo o somatório todos os D1 da semana, sendo feito sempre quando se conclui o trabalho da semana.

Cada dia possui um D1 diferente e possui campos na frente e no verso do formulário a serem preenchidos. O supervisor, ao final do preenchimento dos formulários confere se todas as informações contidas no D7 estão de acordo com os D1, e depois, informa todos os dados manualmente no sistema da Secretaria de Saúde do Estado, o SISFAD (Sistema de informações de Febre Amarela e Dengue). Após esse procedimento, é gerado um arquivo que é enviado para regional de saúde que, logo em seguida é enviado para a secretaria estadual de saúde em Natal.

Por ser um trabalho realizado diariamente, demanda uma grande quantidade de tempo em sua execução, visto que, é realizado de forma completamente manual. Por isso é necessária a aplicação de sistemas computacionais capazes de tratar a informação para melhorar a coleta, filtragem, processamento, e distribuição dos dados que são focos de vários estudos em diferentes áreas, inclusive na área da saúde e de combate a dengue.

A informatização da aquisição de dados e da análise dos mesmos se faz de extrema importância, pelo ganho de tempo e consequentemente possíveis focos e áreas de risco serão identificadas com maior rapidez resultando em um trabalho de combate mais eficaz. Existem vários estudos reportados na literatura envolvendo esses processos e cada vez mais é recorrente esse tema em pesquisas nas mais diversas áreas.

De acordo com **??**) alguns aspectos sobre a relevância dos sistemas de informação como ferramenta de apoio à gestão do trabalho dos profissionais de saúde, é de fundamental importância uma vez que é um recurso computacional capaz de potencializar o conhecimento

de forma rápida, fácil e segura as informações envolvidas. Isto se dá não apenas para obtenção de dados, mas também em sistemas de aquisição de dados e processamento de imagens para combate a dengue.

??) apresentam uma ferramenta capaz de adquirir e armazenar imagens das palhetas das ovitrampas, realizar a contagem semi-automática e automática dos ovos, sem a utilização do microscópio. O sistema desenvolvido é baseado em uma plataforma óptica, numa interface homem-máquina e um software de aquisição de imagem, com a contagem assistida dos ovos do mosquito. Esta contagem semi-automática gerou um ganho de velocidade na contagem de três vezes.

Existem também estudos na área de geoprocessamento com o objetivo de analisar os focos de dengue, verificando, assim, os locais com mais incidência do mosquito. **??**) publicaram um trabalho onde realizaram uma análise espaço-temporal dos casos de dengue no Estado do Rio Grande do Norte, no período de 2000 a 2004 e fizeram uma modelagem de tendência do risco de concentração e dispersão de epidemia. Verificou-se a partir deste trabalho, através da análise de incidência de casos de dengue em todo território estadual, é bastante acentuado nos locais mais vulneráveis a ocorrência do mosquito, por isso, todo o território do Estado é suscetível a esta epidemia.

É essencial que haja integração entre as instituições de ensino e os órgãos responsáveis por combater a dengue. Superintendência de Controle de Endemias (SUCEN), a Secretaria de Saúde da Prefeitura Municipal de Campinas e o Núcleo de Estudos de População (NEPO) da Universidade Estadual de Campinas (UNICAMP) realizaram a implantação de um Sistema de informação Geográfica (SIG) para o controle do dengue no município de Campinas. Essa implementação teve como finalidade a integração e o armazenamento dos dados convencionais existentes, com o intuito de facilitar o trabalho prático do dia a dia, contribuir para tomada de decisões mais adequadas, além de periodicamente proceder a análises mais ricas e acessíveis do quadro epidemiológico mais amplo da endemia. No entanto, observou-se, em decorrência das dificuldades encontradas, e na comparação com os serviços visitados, que para que um SIG funcione com eficiência, oferecendo todas as suas possibilidades e facilitando o andamento dos programas de controle é necessária uma decisão das autoridades municipais para que façam investimentos na obtenção de uma boa base de dados de eixo de ruas e na sua constante atualização.

Nesta perspectiva, o trabalho tem como objetivo principal descrever o desenvolvimento do sistema Web SIADE - Sistema de Informatização do Processo de Aquisição dos Dados dos Agentes de Endemias, que possui o proposito de agilizar a coleta de dados relacionados à prevenção da dengue, automatizar o processo de geração de relatórios e prover o acesso da população aos dados colhidos em tempo real. Além de fornecer ao supervisor acesso automático e atualizado dos agentes em campo.

1.1 Objetivos

A seguir são descritos os objetivos que direcionam o presente trabalho.

1.1.1 Objetivo Geral

O objetivo geral deste projeto de pesquisa é explorar o desenvolvimento de um Sistema Web para a informatização do processo de aquisição dos dados dos agentes de endemias, o SIADE.

1.1.2 Objetivos Específicos

- Estabelecer o conjunto de ferramentas e linguagens de desenvolvimento de aplicações, para a construção da aplicação;
- Facilitar o registro de residências da cidade de Pau dos Ferros por parte dos agente de endemias;
- Agilizar a coleta dos dados em campo pelo agentes de endemias;
- Facilitar o processo de emissão de relatórios;
- Permitir um maior gerenciamento do supervisor do trabalho diário realizado pelos agentes em campo;

1.2 Justificativa e Motivação

O Sistema de Informação em Saúde é um mecanismo de coleta, processamento, análise e transmissão de informação necessária para se organizar e operar os serviços de saúde. Neste sentido os sistemas de informação em saúde devem permitir que os trabalhadores e a população tenham acesso às atualidades, à profundidade das informações, para que sirva de apoio no processo decisório (**??**).

Muitos dos trabalhos desenvolvidos na área de endemias necessitam de dados consistentes e atualizados para a execução das análises estatísticas, como por exemplo, os trabalhos citados anteriormente de geoprocessamento. Contudo, o processo manual que é feito em várias cidades do Brasil, podem gerar dados duvidosos, errôneos e desatualizados. Além disso, há risco de ocorrer lentidão nas entregas das informações, seja por parte dos agentes de endemias ou dos supervisores. Esse tipo de trabalho é muito susceptível ao erro humano, pois há possibilidade do agente poder realizar procedimentos duvidosos, preenchendo os formulários com dados aleatórios, sem necessariamente ir a campo.

É necessário também relatar que há problemas também nos formulários. O RG é de difícil modificação e possui um limite de 60 residências por folha e cada agente de

endemias possui uma folha única referente ao seu quarteirão de trabalho. O D1 possui a limitação de 20 residências por folha, tendo que o agente possuir mais formulários D1 nos quarteirões que possuem mais de 20 residências. O formulário D7 é por bairro, enquanto que os agentes trabalham por quarteirão o que aumenta a chance de erro por parte do supervisor ao verificar as informações do D7 em cada D1. Isso tudo pode ficar ainda mais problemático quando se tem mais de um agente no mesmo bairro. Todos esses problemas geram lentidão na entrega das informações à secretaria de saúde, um custo considerável com gráficas e falta de formulários.

A idéia desse projeto é informatizar todo esse processo, sanando assim, todos os problemas que ocorrem em virtude do trabalho manual. Por meio de dispositivos móveis o agente poderá resgatar todas as informações da residência sem ser necessário o RG para preencher os formulários manualmente. Ele também contará com o histórico daquela residência, podendo rever todas as ações já realizadas no local. Cada residência terá um código, o qual será usado para autenticar a visita do agente de endemias, se precavendo de agentes que queiram agir de maneira duvidosa. O agente não precisará, por exemplo, escrever o endereço ou CEP o qual se encontra, o código gerado será único para cada local, resgatando todas as informações necessárias para preenchimento do D1.

O formulário D7 será preenchido automaticamente pelo sistema à medida que os agentes o utilizam para o fornecimento dos dados. Percebe-se que dessa forma, o trabalho torna-se menos oneroso, pois algumas das informações obtidas poderão ser preenchidas pelo próprio sistema. Além disso, os dados coletados serão mais consistentes, sem possibilidade de falha humana ao contabilizar todas as informações, retirando, assim, a responsabilidade do supervisor de conferir todos os formulários semanalmente de todos os agentes. Dessa forma têm-se os dados atualizados e mais consistentes o quais podem ser utilizados para inúmeras aplicações, como: em geoprocessamento, análise estatística, correlacionamento de informações com dados fluviais, obter as zonas de risco maior caso de incidência, número de residências fechadas e quantidade de terrenos baldios

1.3 Organização do Texto

A organização do trabalho utiliza a estrutura de capítulos. Além desta introdução, que visa explicar o problema que gerou o desenvolvimento do sistema, o restante será desenvolvido da seguinte maneira:

- Capítulo 2 (Referencial Teórico): Apresenta um estudo sobre as características e o desenvolvimento de aplicações web SIADE com o uso de tecnologias. Além de uma abordagem da linguagem Python, e um maior detalhamento do framework Django;
- Capitulo 3 (Desenvolvimento da Aplicação Web): exibe informações sobre o que é

o sistema e seu fluxo através de casos de uso e modelagem conceitual; Além disso propõe uma visualização com maior riquezas de detalhes do uso do Django no desenvolvimento do sistema proposto.

- Capítulo 4 (Apresentação do SIADE): apresenta uma visualização do sistema em suas principais funcionalidades implementadas no contexto deste trabalho;
- Capitulo 5 (Conclusão): Aborda as considerações finais no desenvolvimento da aplicação, assinalando as contribuições da pesquisa e sugerindo possibilidades futuras de melhorias no sistema SIADE.

2 REFERENCIAL TEÓRICO

Este capitulo apresenta conceitos que fundamentam o desenvolvimento da aplicação web, centrado na modelagem e na apresentação das tecnologias utilizadas para implementá-la. Em seguidas são descritas peculiaridades pertinentes aos *frameworks*. Por fim, uma abordagem sucinta sobre a linguagem de programação Python e o *framework* Django.

2.1 Aplicações Web

A arquitetura distribuída da web tem herdado os benefícios das aplicações em rede como baixos custos de manutenção, descentralização e compartilhamento de recursos (**??**). Porém, o aumento da complexidade das atividades realizadas por meio de interfaces web mantém a necessidade de interações que permitisse uma maior usabilidade. Nesse contexto, os desenvolvedores foram motivados a criar interfaces gráficas ricas e aplicações com maior dinamismos aos usuários finais. Essa mudança fez com que simples documentos formatados se tornassem verdadeiras aplicações, exibindo dados armazenados em banco de dados e permitindo a sua manipulação *online*..

O termo Aplicação Web é utilizado para descrever o software em um ambiente web. Para **??**), uma aplicação Web pode ser desde uma simples página HTML (HiperText Markup Language) com alguns links ou um sistemas web conectado a servidores.

Em uma aplicação dinâmica baseada em HTML, denominada como tradicional, a interface da aplicação é um documento HTML processado pelo servidor a cada requisição do usuário (**??**).

Segundo **??**), as seguintes características são encontradas na grande maioria das aplicações Web:

> A interface da aplicação com o usuário é exibida em um *browser*, também chamado de navegador Internet;
>
> As requisições enviadas pelo usuário através da interface são tratadas por um servidor Web;
>
> Um grande número de usuários pode acessar a aplicação Web ao mesmo tempo;
>
> A quantidade de acessos à aplicação Web pode mudar completamente entre um dia e outro;
>
> Os usuários querem acesso na base de "24/7/365", ou seja, 24 horas por dia, 7 dias por semana e 365 dias por ano, devido a diferença de fuso horário entre os usuários da aplicação, que podem estar em qualquer lugar do mundo;
>
> A evolução da aplicação Web não se dá por meio de versões planejadas e cronologicamente espaçadas, e sim continuamente;

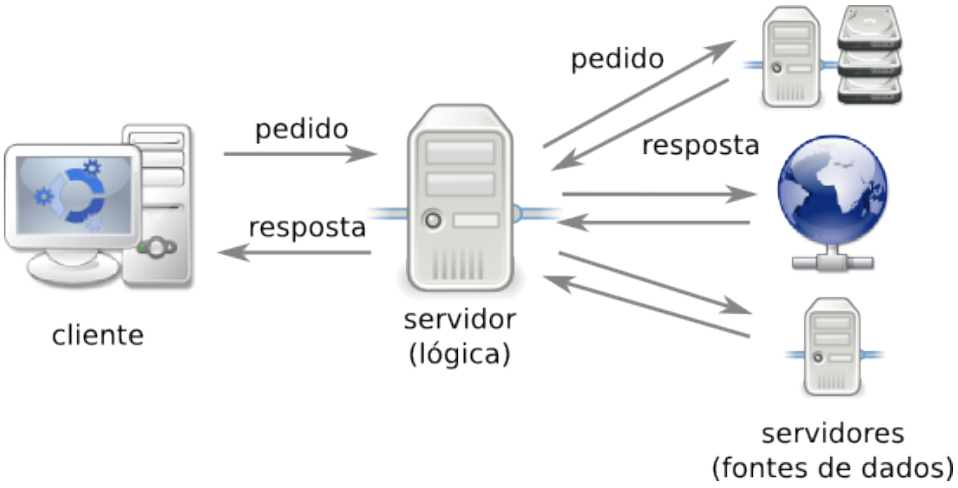
Necessidade de colocação no mercado rapidamente;

Dificuldade em limitar a população de usuários que podem ter acesso a uma aplicação Web, sendo necessária a utilização de fortes medidas de segurança em toda a infra estrutura que apoia uma aplicação Web;

A estética possui grande influência no sucesso da aplicação.

A seguir, segue a figura 1 que demonstra o funcionamento de uma aplicação Web.

Figura 1 – Funcionamento de requisições e respostas em uma Aplicação Web.



Fonte: (??)

Como observado na 1, além de uma infraestrutura de acesso a internet, é utilizada uma estrutura de serviços conhecida como Cliente/Servidor. Ou seja, de um lado está o Cliente que solicita (requisita) dados ao Servidor através do navegador web *(browser)*. Do outro, está o Servidor, que recebe todas as requisições, interpretando-as e retornando-as ao cliente. O papel do cliente é interceptar as ações do usuário, enviar as requisições para o servidor e apresentar as respostas.

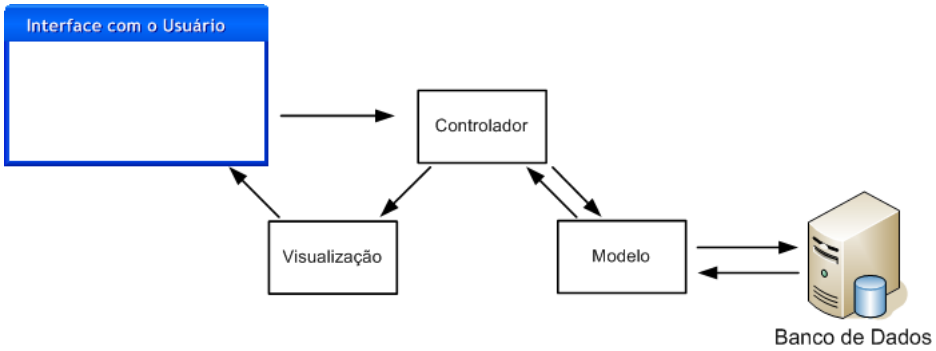
2.1.1 Padrão Arquitetural MVC

Muitas aplicações Web utilizam padrões de projeto *(design pattern)* que auxiliam em um melhor desenvolvimento, além de minimizar o problema do forte acoplamento entre a camada de interface, de lógica de negócio e de banco de dados. Por isso a sigla MVC, em que divide a aplicação nestas três unidades distintas: *Model, View, Controller*.

O padrão de projeto MVC é um padrão de projeto arquitetural. Esse tipo de padrão define os elementos, as relações e as regras a serem seguidas que já tiveram sua utilidade avaliada em soluções de problemas passados (??).

Por causa dessa separação, múltiplas visões e controles podem interagir com um mesmo modelo. Na figura 2 pode ser observado o padrão arquitetural MVC:

Figura 2 – Fluxo entre as camadas na Arquitetura MVC.



Fonte: (**??**)

A camada de Modelo é responsável por realizar a interface entre o banco de dados e a aplicação. Nesta camada, devem ser desenvolvidas todas as regras de negócio que se aplicam aos dados, para garantir que nenhum objeto da aplicação efetue uma alteração com valores inválidos no banco de dados (**??**). Por sua vez, mantêm o estado da aplicação e fornece ao controlador o acesso aos dados, para apresentar uma melhor apresentação ao usuário.

A visão está relacionada a exibição dos dados e todas as informações inerentes a aplicação. Em outras palavras, ela oferece uma interface a ser exibida ao usuário. Com isso, ele pode realizar todo o gerenciamento de dados na aplicação.

A camada de controle é responsável por fazer a ligação entre o modelo e a visualização, além de interpretar as ações do usuário e as traduzir para uma operação sobre o modelo, onde são realizadas mudanças e, então, gerar uma visualização apropriada. Esta camada é fundamental, pois não permite a interação direta dos elementos da camada de visão com os objetos da camada de modelo.

2.1.2 Tecnologias em Aplicações Web

No intuito de propor dinamismos as páginas Web, que antes eram estáticas e com a necessidade de uma maior rapidez em seus desenvolvimentos, várias tecnologias surgiram e continuam surgindo para tornar a web um ambiente mais agradável, interativo e amigável.

Sobre o uso da web, em que sistemas começaram a ser implementados de forma dinâmica, com isso o intuito de sanar todos os problemas enfrentados e auxiliando assim seus usuários **??**) declara:

> O uso da Web explodiu no meio da década de 90 depois do primeiro navegador gráfico ter aparecido. A necessidade de computação associada com documentos HTML, que, por si só, são completamente estáticos, rapidamente, tornou-se crítica. A computação do lado do servidor foi possível com o *Common Gateway Interface* (CGI), que permitiu a docu-

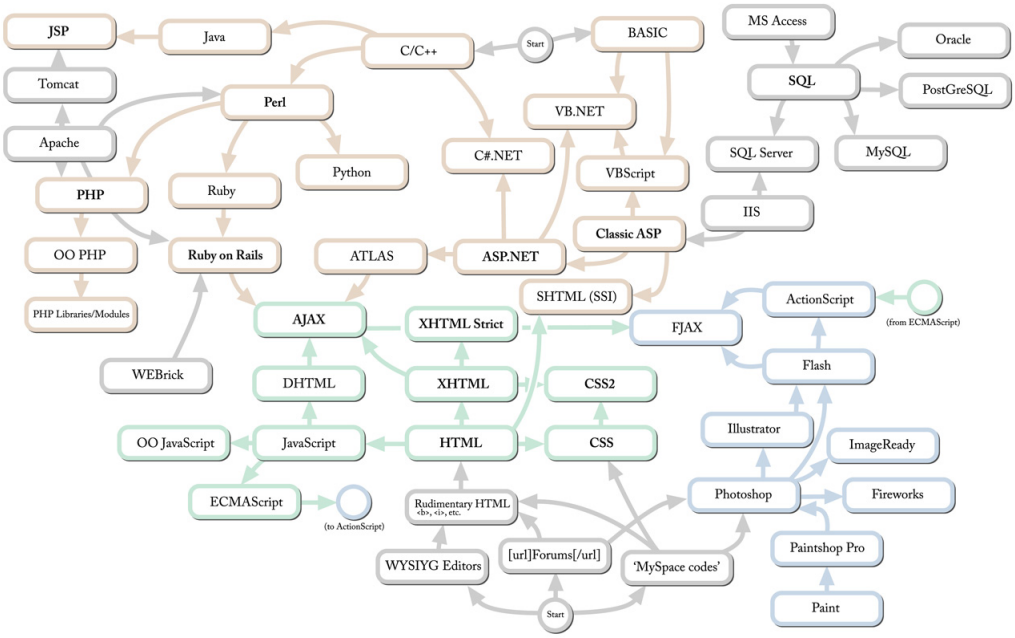
> mentos HTML requisitar a execução de programas no lado do servidor, com o resultado da computação retornando para o navegador em forma de documentos HTML. Computação do lado do navegador tornou-se disponível com o advento dos *applets* Java. Ambas destas abordagens estavam sendo lentamente substituídas por novas tecnologias em grande parte, por linguagens de *script*.

No intuito de facilitar o desenvolvimento de sistemas web, muitos desenvolvedores começaram a aplicar o uso de *frameworks* (uma estrutura de suporte definida em que um projeto pode ser organizado e desenvolvido). Juntamente com uma linguagem específica, ele provê auxílio ao desenvolvedor com um conjunto de bibliotecas de uso geral e específicos, que possuem a finalidade da construção mais flexível de grandes aplicações.

As linguagens dinâmicas eram vistas, no passado, apenas como linguagens *script*, usadas para automatizar pequenas tarefas, porém, com o passar do tempo, elas cresceram, amadureceram e conquistaram seu espaço no mercado, a ponto de chamar a atenção dos grandes fornecedores de tecnologia (**??**).

Na figura 3 podemos observar a quantidade de tecnologias existentes desde linguagens de programação, padrões, bibliotecas, *frameworks* e ferramentas que podem ser utilizadas.

Figura 3 – Teclonogias para o desenvolvimento WEB.



Fonte: (**??**)

Contudo, a escolha de uma tecnologia depende, primeiramente, da necessidade que será atribuída, dos recursos que estas podem oferecer na aplicação, como ainda a questão de segurança, integridade de dados e agilidade a ser introduzida.

2.2 *Frameworks*

Um *framework* consiste em um conjunto de classes que se relacionam e representam uma solução para uma aplicação, que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica. Ao contrário das bibliotecas, é ele que dita o fluxo de controle da aplicação, chamado de inversão de controle.

??) declara que um *framework* é uma coleção de artefatos de software que é utilizado por várias aplicações diferentes. Esses artefatos são em geral classes, juntamente com o software exigindo para utilizá-las.Esta ferramenta útil no momento em que você constrói ou utiliza certo componente em mais de uma vez, ocasionando uma maior produtividade.

??) afirma que os *frameworks* possuem interfaces e classes coesas, além da implementação de funções básicas e invariantes do sistema em desenvolvimento, ele define a separação de funções e comportamentos gerais da parte específica.

Além das características citadas por **??**), os *frameworks* possuem outras em comum, como: facilidade de uso, reutilização de código, grande documentação, extensibilidade, permite ao desenvolvedor implementar de acordo com as necessidades e os programas recorrentes no âmbito do projeto, e ainda deve apresentar segurança e eficiência.

Algumas destas ferramentas, como o Django, posuem uma biblioteca ORM (*Object Relational Mapping*). Ela é responsável pela conexão e conversão das classes em tabelas no banco de dados. Sendo assim, o desenvolvedor não se preocupa com linguagens de manipulação de dados, como a SQL (*Structured Query Language*).

Frameworks como: RoR (*Ruby On Rails*), *Grails* (*Groovy on Rails*), *CodeIgniter*, *Zend Framework*, *Symfony*, Django e CakePHP, estão ganhando espaço no desenvolvimento de aplicações web. Não pode deixar de destacar que todos eles utilizam o padrão arquitetural MVC. A figura 4 demonstra alguns *frameworks* mais utilizados no mercado de desenvolvimento:

Assim sendo, vale salientar que o uso destas ferramentas possui benefícios claros, isso é, quando se trata de redução de custos. Porém, a sua escolha é de extrema importância quando se questiona o sucesso no desenvolvimento. Não basta apenas utilizá-lo, mas, é preciso que se conheça qual o mais adequado a um dado domínio de aplicações.

2.2.1 Vantagens e Desvantagens do uso de *Frameworks*

A utilização de *frameworks* agiliza a vida de desenvolvedores, pela questão da facilidade oferecida. Mas, nem tudo propicia a benefícios.

Suas principais características começam na facilidade de uso, reutilização de código, documentação e funcionalidades abstratas, onde o desenvolvedor as implementa de acordo com a sua necessidade, além disso, deve ser seguro, eficiente e completo, para atender

Figura 4 – Alguns *frameworks* mais utlizados no desenvolvimento web.

Fonte: (??)

todos os problemas propostos.

Em contra partida, possuem um alto deficit de aprendizado para iniciantes, pelo fato do uso de código de terceiros e/ou padrões desconhecidos. Muitas vezes é necessário a instalação de extensões, módulos ou ainda atualização da linguagem e do próprio sistema operacional para iniciar o desenvolvimento.

Seu uso pode trazer um ganho em desenvolvimento de aplicações. Porém, a pratica indevida, pode evidenciar numa perca gigantesca de produtividade. Este, deve ser inserido no desenvolvimento da aplicação de acordo com o grau problemático que possua, ou seja, as necessidades inerentes ao problema do sistema.

2.3 Aplicações Web com Python e Django

Além do crescimento da web e o surgimento de aplicações de grande porte, os desenvolvedores detectaram a necessidade da utilização de ferramentas que facilitassem o desenvolvimento de projetos de software.

Neste intuito, surge o *framework* Django. Possui como suporte e herança a linguagem de programação Python e contribui nesta nova realidade de desenvolvimento. Utiliza como padrão arquitetural o MVC. As subsecções posteriores, apresentam um breve estudo sobre a linguagem Python e o Django.

2.3.1 Python

Python é uma linguagem interpretada e orientada a objetos. Os objetos possuem dados próprios e são definidos para estruturar o programa. Sendo uma linguagem respeitada

pela comunidade de *Software* Livre, possui fontes de pesquisas amplas e seguras, com suporte que vão desde tutoriais a programas completos.

??) advoga: "Python é uma linguagem de altíssimo nível (em inglês, *Very High Level Language*) orientada a objeto, de tipagem dinâmica e forte, interpretada e interativa."

O nome da linguagem origina-se do nome da série humorística britânica *Monty Python's Flying Circus* (**??**) e foi lançada em 1991 pelo holandês Guido van Rossum, sendo hoje mantida pela organização sem fins lucrativos *Python Software Foundation* (PSF) sob licença livre.

O Python compila e funciona em praticamente todas as principais plataformas atualmente em uso. Pois sua implementação é escrita em ANSIC portável, que incorpora módulos, exceções, tipos de dados e classe de alto nível.

Suas principais características incluem uma sintaxe clara e legível, capacidade de introspecção, orientação a objetos, modularidade completa com suporte a hierarquia de pacotes, dados dinâmicos, biblioteca padrão e módulos de terceiros extensa, além de módulos escritos em C, C++ (ou Java para Jython ou .NET para IronPython) (**??**). Assim, o Python não prioriza apenas desempenho, mas a junção de todas estas caraterísticas, facilitam o desenvolvimento rápido em múltiplas áreas.

2.3.1.1 Interpretador Python

Para um maior entendimento ao desenvolver, Python utiliza para separação de blocos de códigos indentação ou espaços em branco. Outras linguagens de programação como JAVA, C, C++, usam blocos aninhados com delimitadores, como chaves e palavras reservadas.

Por não servi-se de delimitadores, erros em Python se tornam frequentes. Para que isso não ocorra, é indispensável a utilização de IDES, pois possuem conjuntos de extensões internas que realizam a indentação automática, minimizando a margem de erros. Além disso, as IDES permitem automaticamente verificar se existem infração ou código em excesso desnecessário.

Por padrão o Python já encontra-se instalado em quase todas as versões do Linux e na maioria dos Mac OS. Assim o usuário só precisaria executar o comando python em seu terminal e testá-la.

Os testes no Python podem ser realizados no seu interpretador interativo. As entradas de comandos pode ser realizadas a partir dos *prompts* ('»> ' e '...'). Ao executar um comando, o *prompt* mostrará uma saída. É importante salientar, que não é necessário a criação de classes para realizar testes.

No intuito de testar o interpretador do Python, pode-se criar um programa com

extensao .py, em qualquer editor de texto ou em uma IDE específica para a linguagem.

2.3.2 Django

É com a seguinte frase: "Um *framework* de desenvolvimento web para perfeccionistas com prazo de entrega", que a *Django Software Foundation* faz sua apresentação.

Django é uma tecnologia web de alto nível, que estimula o desenvolvimento rápido, limpo, e sem repetições corriqueiras de código.

De acordo com o site oficial do Django, o nome django foi inspirado no músico de jazz Django Reinhardt e foi criado no *Lawrence Journal-World*, para gerenciar o site jornalístico na cidade de Lawrence, no Kansas.

Assim, como outros *frameworks*, o Django é construído sobre uma linguagem de programação, neste caso, a linguagem Python. Isto significa, que seus sistemas possuem acesso completo a biblioteca padrão Python e da própria específica do Django (**??**).

A **??**) acrescenta que existe uma filosofia por trás das ideias Django, princípios como: DRY (*Don,t Repeat Your Self*: Não seja repetitivo), que faz com que o código produzido seja voltado para o reuso, economia e aumento de produtividade.

O propósito do Django, é prover um conjunto coeso de interfaces e métodos que facilitam a realização de tarefas comuns no desenvolvimento de aplicações.

2.3.2.1 Características do Django

O Django se torna completo, por ser composto de vários componentes menores.

??) aborda alguns destes componentes como:

> Mapeamento Objeto-Relacional (ORM): Com o ORM do Django você define a modelagem de dados através de classes em Python. Com isso é possível gerar suas tabelas no banco de dados e manipulá-las sem necessidade de utilizar SQL (o que também é possivel).
>
> Interface Administrativa: No Django é possivel gerar automaticamente uma interface para administração para os modelos criados através do ORM.
>
> Formulários: É possível gerar formulários automaticamente através dos modelos de dados.
>
> URL's Elegantes: No Django não há limitações para criação de URL's elegantes e de maneira simples.
>
> Sistema de *Templates*: O Django tem uma linguagem de *templates* poderosa, extensível e amigável. Com ela você pode separar *design*, conteúdo e código em Python.
>
> Sistema de *Cache*: O Django possui um sistema de cache que se integra ao *memcached* ou em outros *frameworks* de cache.
>
> Internacionalização: Django tem total suporte para aplicações multi-idioma, deixando especificar *strings* de tradução e fornecendo ganchos para funcionalidades específicas do idioma.

??) afirma que o Django possui características que possibilitam ao desenvolvedor em uma maior agilidade, custo e menor tempo ao iniciar o desenvolvimento de uma aplicação web.

Ele oferece uma interface administrativa padrão, montada com os objetos determinados nas classes. A partir dos atributos declarados em cada classe Python, o Banco de dados é modelado. O Django gera formulários simples a partir dos objetos acessados no BD. Sem falar no suporte a diversos idiomas e a não limitação do programador sobre como ele necessita modelar as URLs, que podem ser criadas com o auxílio de expressões regulares. Tudo realizado de forma simples e sem dificuldades.

2.3.2.2 Vantagens e Desvantagens do *Framework* Django

Por passar muitos anos após a sua criação, o Django possui uma comunidade extremamente ativa, em que propicia melhorias, aperfeiçoamento e criação de novos recursos no *framework*.

Segundo **??**), a utilização do *framework*, traz uma série de vantagens na construção de aplicações web. Por auxiliar na criação e validação de formulários. Assim como na criação de validações específicas para cada atributo de uma classe.

O django possui alguns aspectos como: funcionalidades de fábrica, flexibilidade, simplicidade na geração de arquivos e pastas no projeto, a organizado do projeto em *apps*, tornando-o bem estruturado. Em contra partida, não é favorável para pequenos projetos, pois perde simplicidade e considera a necessidade de um banco relacional, além de diversas outras configurações.

O *framework* é facilmente customizável, tanto no *back-end*, quanto no *front-end*. Os campos dos objetos podem ser facilmente exibidos. Sem deixar de ressaltar que os filtros e buscas são fáceis de serem usados e customizados, de acordo com o critério de cada funcionalidade. Também vale frisar o funcionamento por *templates, views*, contextos e rotas.

2.3.2.3 Django e a Arquitetura MVC

Django é baseado no design *pattern* MVC (Model, View, Controller).

O *model* (M) são classes Python que herdam de *model.Model*, empregadas para interagir com o banco de dados, por isso é composta pelo ORM (Object Relational Mapping).

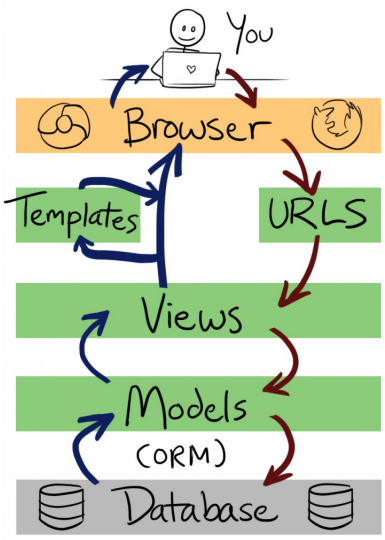
Já o *controller* (C), é responsável pela lógica de aplicação e o envio de requisições e respostas. No Django esta tarefa é realizada por um arquivo chamado *views.py* em conjunto com o arquivo de mapeamento de requisições *urls.py*, que representam o *controller*. Desta forma o C do MVC seria o V *(views)* do Python. Nesta camada, ainda estão todos os

outros mecanismos entre a requisição e a resposta, como: *handler*, *middlewares* e URL *dispatcher*.

E por fim a *View* (V), que é a camada que apresenta dados e interage com o mundo exterior. Esta tarefa é executada por um sistema de *Templates* (T) no django. Desta forma o V do MVC equivale a camada de *templates* T.

Por estes motivos o Django possui um modelo interno equivalente chamado MTV, realizando as mesmas tarefas do conhecido *design pattern* MVC (**??**). Ver figura 5:

Figura 5 – Models, Views e Controller no Django.



Fonte: (**??**)

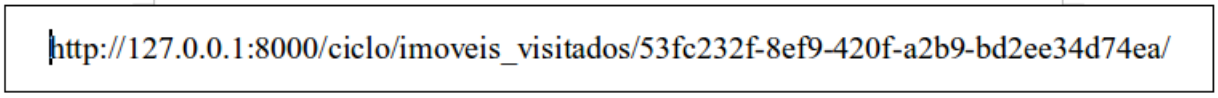
Assim, cada camada pode ser desenvolvida para uma página específica, ou seja, cada uma possui esse modelo interno. Além dos Models, Views e Templates, é acrescentado no fluxo de sistema as URLs *(Universal Resource Locator)*, que trata do mapeamento das rotas e recursos do sistema.

2.3.2.4 Funcionamento do Framework Django

Toda página em uma aplicação web, pode ser acessada por meio de endereços específicos. Para que isto ocorra, é necessária a realização de uma ação conhecida como requisição, que é acompanhada de várias informações. Uma das principais é a URL, onde trafega os principais dados a ser requisitados. Uma URL é composta especificamente pelo protocolo, o domínio, a porta, o caminho e os parâmetros.

A figura 6 seguinte, descreve os componentes de uma URL completa.

Figura 6 – Exemplo de URL.



Fonte: Elaborado pelo autor.

Detalhamento dos componentes:

```
Dominio: < 127.0.0.1 >
Porta: < 8000 >
Caminho:  < /ciclo/imoveis\_visitados/ >
Parametros: < /53fc232f-8ef9-420f-a2b9-bd2ee34d74ea/ >
```

As requisições passam primeiro pelo *Handler*, só depois são repassadas para os *middlewares*.

Os *Handlers* determinam o que acontece em cada mensagem em um registrador. Ele descreve um comportamento de registro especial, como escrever uma mensagem para a tela, para um arquivo ou uma tomada de rede (**??**).

Os *Middelewares* são pequenos trechos de código que analisam a requisição nas entradas (*HttpRequest*) e as respostas (*HttpResponse*) na saidas. Cada componente *middleware* é responsável por fazer alguma função específica. Por exemplo, o Django inclui um componente *middleware*, *AuthenticationMiddleware*, que associa os usuários com solicitações, usando sessões (**??**).

Um dos *middlewares* faz com que toda a segurança e autenticação de usuários seja feita. Outro adiciona a função de sessão, uma forma de memorizar o que o usuário está fazendo para atendê-lo melhor. Outro memoriza as respostas no Cache. Caso a próxima requisição tenha o mesmo caminho e os mesmos parâmetros, ele retorna a resposta memorizada, evitando o trabalho de ser processada novamente pela *view* (**??**).

A requisição só chega na URL *Dispatcher*, quando passa pelos *middlewares*. Este componente também é responsável por verificar o endereço da URL e também o arquivo *urls.py* do projeto, a fim de encontrar a *view* que será chamada para responder a requisição. No arquivo *view* contém as funções que recebem uma requisição e retorna como resposta uma página.

Em se tratar de *views*, o Django possui um recurso conhecido com *Generic Views* (Visões Genéricas em Português). São visões prontas com funções preestabelecidas, ou seja, com um funcionamento já conhecido. As visões genéricas são bastantes uteis em autenticação de usuários, nos CRUDs (*Create, Update, Read, Delete* – Criar, editar, ler, deletar), em redirecionamentos, nas funções de *logoff* e *login* de sistemas, e etc.

Um *template* Django é uma sequência de texto que se destina a separar a apresentação de um documento de seus dados. Define espaços e vários pedaços de lógica básica (*tags*), que regulam o modo como o documento deve ser exibido. Normalmente, os modelos são utilizados para produzir o HTML, mas os modelos Django são igualmente capazes de gerar qualquer formato baseado em texto (**??**).

Para persistir e consultar informações do BD, não é necessário linguagens de consulta de banco, como o SQL. O ORM (Mapeamento Objeto-Relacional) interpreta a programação de acesso aos objetos no código em Python e converte em consultas ao banco de dados. O models.py é o arquivo responsável pelo mapeamento através do ORM.

2.3.2.5 Django ORM e QuerySets

Como tratado na subsecção anterior, o ORM é uma tecnologia que existe no Django, que traduz os objetos das classes Python, para consultas em linguagem SQL no BD.

O ORM oferece uma gama de facilidades como: independência em relação ao banco de dados SQL, acesso direto a objetos relacionados, implementação fácil e flexível de operações CRUD (*Create, Read, Update, Delete* - as quatro operações básicas da persistência de dados), validação de campos e transações ACID (atomicidade, consistência, isolação e durabilidade - propriedades que asseguram a confiabilidade do processamento de transações) (**??**).

Para que ocorra a persistência das informações em banco, faz-se o uso de *QuerySets*. Um *QuerySet* é em essência, uma lista de objetos de um determinado *model*. Seus métodos, fazem a comunicação com o banco de dados, permitindo efetivar as persistências, consultas, filtros e ordenação.

Conforme **??**), os métodos podem ser encadeados uns aos outros, de forma a criar uma complexa e comprida sequência de filtros e regras.

O métodos mais empregados nos *QuerySet* são 4: *.all()*, *.filter()*, *.get()*, *order_by*. O *.all()*, que retorna todos os objetos de uma classe no *model*. O *.filter()*, devolve uma lista de *queryset* de um determinado *model*. Para isso faz uso de critérios, como se gerassem uma clausula *WHERE* em SQL. O *.get()*, recupera um objeto do *model*. Se nenhum objeto é encontrado, é levantada uma exceção afirmando que não existe. Caso contrário, retorna o objeto. E o *order_by*, que determina a ordenação de objetos.

Várias são as condições utilizadas como parâmetros nos métodos de consulta. A figura 7, lista as condições e os resultados da utilização de métodos *QuerySets* disponíveis no Django.

Figura 7 – Condições de consulta aos dados.

Condição	Resultado
exact	Valor do campo utilizado na condição é exatamente igual ao informado
iexact	Possui a mesma funcionalidade do *exact*, porém desconsidera diferenças entre maiúsculas e minúsculas
contains	Conteúdo do campo utilizado na condição possui o valor informado
icontains	Possui a mesma funcionalidade do *contains*, porém desconsidera diferenças entre maiúsculas e minúsculas
gt	Valor do campo utilizado na condição é maior que o valor informado
gte	Valor do campo utilizado na condição é maior ou igual ao valor informado
lt	Valor do campo utilizado na condição é menor que o valor informado
lte	Valor do campo utilizado na condição é menor ou igual ao valor informado
in	Valor do campo utilizado na condição seja igual a um dos ítens de uma lista
startswith	Valor do campo utilizado na condição inicie com o valor informado
istartswith	Possui a mesma funcionalidade do *startswith*, porém desconsidera diferenças entre maiúsculas e minúsculas
endswith	Valor do campo utilizado na condição termine com o valor informado
iendswith	Possui a mesma funcionalidade do *endswith*, porém desconsidera diferenças entre maiúsculas e minúsculas
range	Valor do campo utilizado na condição esteja entre a faixa de valores informada (inclusive)
year	Para campos do tipo *date* e *datetime*, o ano seja igual ao valor informado, utilizando quatro dígitos
month	Para campos do tipo *date* e *datetime*, o mês seja igual ao valor informado, utilizando números de 1 (Janeiro) à 12 (Dezembro)
day	Para campos do tipo *date* e *datetime*, o dia seja igual ao valor informado
isnull	Valores nulos no campo utilizado na condição sejam considerados na consulta, informando *True* ou *False*
search	Utilizado para busca por conteúdo apenas no SGBD MySQL, pelo fato de utilizar a indexação do texto disponível pelo SGBD, desde que este tenha sido configurado

Fonte: (**??**)

Ao disponibilizar uma camada de abstração, entre a aplicação e o sistema gerenciador de banco de dados, o Django possibilita ao desenvolvedor criar, consultar, atualizar e excluir registros no banco de dados, sem a necessidade de escrever códigos na linguagem SQL. Esta característica, permite que a aplicação se torne independente da tecnologia utilizada pelo SGBD (Sistema Gerenciador de Banco de dados). Caso haja a necessidade da migração para outro SGBD, não haverá impacto na aplicação.

3 DESENVOLVIMENTO DA APLICAÇÃO WEB

O trabalho teve como objetivo o desenvolvimento da Aplicação Web SIADE utilizando o framework Django. Sendo assim, neste capítulo serão descritos as técnicas e meios envolvidos no sistema.

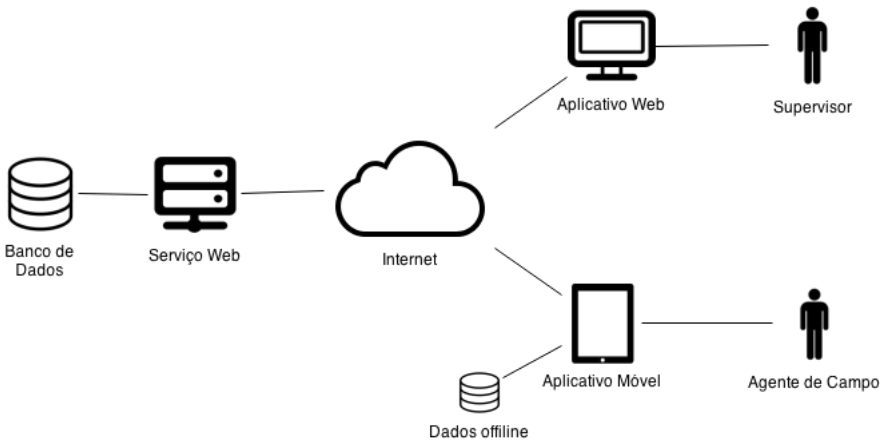
3.1 A Aplicação Web

O SIADE (Sistema de Informatização do Processo de Aquisição dos Dados dos Agentes de Endemias) é uma aplicação web voltada para á área da saúde. O sistema possui como objetivo principal, automatizar a coleta de dados realizados pelos agentes de endemias das Secretarias Municipais de Saúde. Sendo assim, ele permite uma maior agilidade e controle sobre os dados coletados. Sem deixar de destacar um melhor monitoramento sobre o trabalho realizado pelos agentes.

Por meio de dispositivos moveis, o agente pode coletar informações das residências sem o auxílio de formulários de papel, cujos os quais, sem o uso do sistema, eram preenchidos manualmente. A aplicação conta com o histórico dos dados obtidos de uma residência específica, podendo rever todas as ações já realizadas no local.

A aplicação é composto por três módulos principais: serviço web, aplicação web e a aplicação móvel. Porém, o foco principal desta seção será o Aplicação Web. Pode ser visualizado na figura 8 a arquitetura do sistema proposto.

Figura 8 – Arquitetura do sistema proposto



Fonte: Elaborado pelo autor.

A Aplicação Web fornece operações cadastrais de bairros, quadras, ruas, imóveis, agentes. Permite ao supervisor, o gerenciamento das informações coletadas pelos agentes nas visitas realizadas em campo. Para isto são auxiliados pela emissão de relatórios diários e/ou semanais.

O módulo servidor é responsável por armazenar os dados e controlar toda a lógica de negócio existente no sistema e na troca de mensagens com o cliente.

É relevante destacar que o sistema possui um elevado grau de complexidade. Logo, o tempo de desenvolvimento cai, pois a necessidade de um maior estudo sobre as tecnologias empregadas para iniciar a implementação. Sendo assim, é importante utilizar um Framework como o Django, o qual possui diversos recursos pré-configurados, para melhorar a agilidade do desenvolvimento da aplicação. O SIADE possui uma lógica relacional complexa, com a utilização do Framework, não foi necessário se importar com a modelagem do banco, pois, como visto anteriormente, ele já utiliza o ORM.

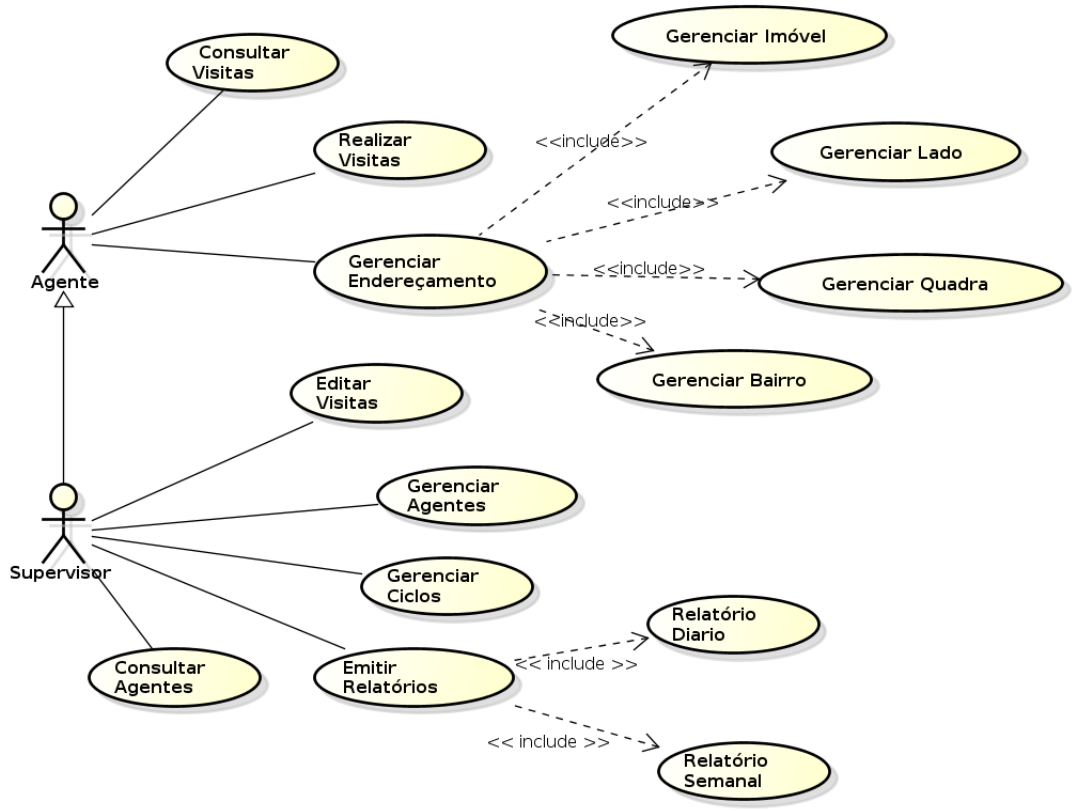
3.2 Modelagens e Diagramas

As modelagens e diagramas nas subseções posteriores possuem a finalidade de provê uma representação parcial do sistema. Os diagramas apresentam múltiplas visões, as quais, podem ser examinadas e modificadas na finalidade de compreender seu desenvolvimento, sua arquitetura e auxiliar principalmente, em uma melhor interpretação da aplicação.

3.2.1 Diagrama de Casos de Uso

A figura 9 mostra o diagrama de casos de uso do sistema, ao qual descreve os atores agentes e supervisores. O ator supervisor mantém o maior nível de acesso e realiza todas as tarefas que o sistema disponibiliza. Já o agente, realiza todas as tarefas determinadas pelo supervisor.

Figura 9 – Diagrama de Casos de Uso do Sistema



Fonte: Elaborado pelo autor.

Abaixo podemos ver detalhadamente os casos de uso do ator Agente, com uma breve explicação do que será feito em cada caso de uso.

- Realizar visita: O agente pode realizar uma visita em imóveis. O intuito é coletar os dados sobre o tratamento realizado.
- Consultar visitas: O usuário(agente) poderá consultar uma visita especifica e ver o trabalho realizado anteriormente.
- Gerenciar endereçamento: O agente pode realizar a inserção, remoção, e atualização de imóveis, bairros, ruas, quadras e seus respectivos lados. Este é essencial para a realização de uma visita.

Em seguida é apresentado os casos de uso do ator Supervisor:

- Editar visitas: O supervisor poderá editar os dados coletados pelos agentes nas visitas já realizadas.

- Gerenciar Agentes: O supervisor poderá criar agentes e caracterizá-los, determinar aonde e quais bairros e quadras que eles irão trabalhar.

- Gerenciar endereçamento: Assim como os Agentes, os Supervisores podem realizar a inserção, remoção, e atualização de imóveis, bairros, quadras e seus respectivos lados.

- Gerenciar Ciclos: O supervisor poderá iniciar e encerrar o ciclo. Um ciclo é essencial para a realização da especificação trabalho de cada agente. É nele que especificamos o período a ser trabalhado, os bairros, imóveis e quadras de um respectivo agente, e ainda o tipo de tratamento a ser realizado.

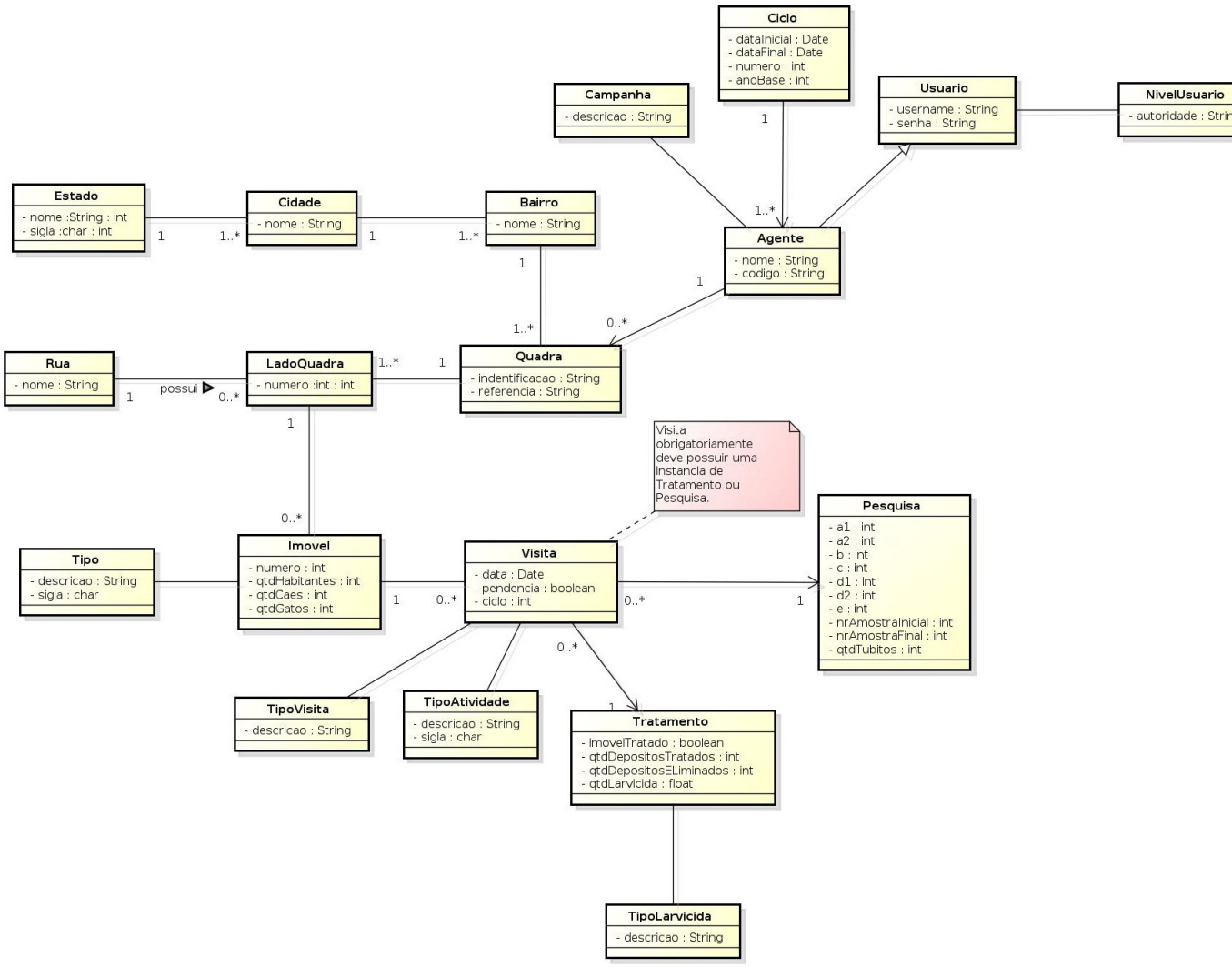
- Consultar Agentes: O supervisor poderá procurar por um determinado agente e visualizar o andamento do seu trabalho.

- Emitir Relatórios: O supervisor poderá emitir documentos (Relatórios) a partir dos dados coletados pelos agentes. Se dividem em relatórios do tipo D1 (diário) e D7 (semanal). O primeiro informa a quantidade de dados coletados em um dia de trabalho realizado. O segundo é a soma dos dados coletados em sete dias (uma semana) trabalhados por um agente.

3.2.2 Diagrama de Classes

Além dos Diagramas de Caso de Uso foi utilizado outro diagrama da UML (Unified Modeling Language), o diagrama de classes, que é muito utilizado no desenvolvimento de sistemas orientados a objetos. O Diagrama 10 é uma parcial das classes do sistema.

Figura 10 – Diagrama de Classes do Sistema.



Fonte: Elaborado pelo autor.

Na Figura acima, estão representadas as classes relacionadas à manipulação de dados do sistema. As operações de cada classe foram omitidas por questões de exibição, entretanto, os atributos foram considerados para demonstrar as propriedades fundamentais que diferem as classes.

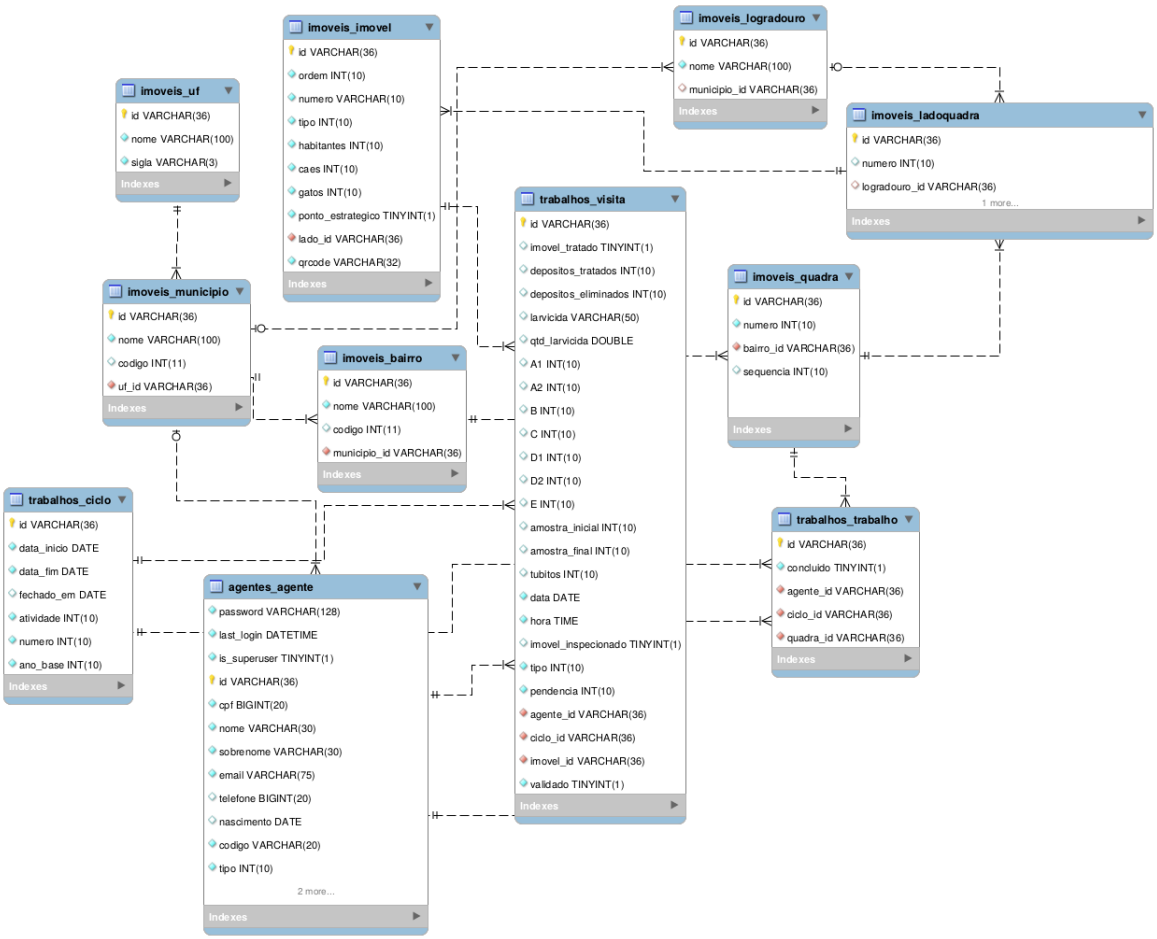
Todas as classes possuem associações que representam o relacionamento entra ambas. É notório a importância destas no funcionamento de todos os módulos no sistema, pelo fato das informações que os estruturam.

A principal classe é a Visita, ela por sua vez é composta pelo relacionamento das classes, Tratamento, Pesquisa, TipoAtividade, TipoVisita e Imovel. A classe Agente, está ligada a Ciclo e Quadra que faz relação com a de Imóvel. Sendo assim, os atributos que cada uma possui, constitui as tabelas criadas em banco. É a partir destas que o Supervisor gerencia o trabalho realizado por cada agente.

3.2.3 Diagrama de Modelo Entidade Relacionamento (MER)

O diagrama a seguir descreve o relacionamento entre os objetos (entidades) e suas características (atributos). Sendo assim representa de forma abstrata a estrutura que possuirá o banco de dados da aplicação. Pode-se deixar claro que o banco de dados contém várias entidades, estas possuem chaves e tabelas intermediárias. Observe o Diagrama 11, que apresenta o modelo entidade/relacionamento do SIADE.

Figura 11 – Diagrama entidade/relacionamento do SIADE



Fonte: Elaborado pelo autor.

A tabela “imoveis_uf” contém os dados referentes a Unidade Federativa de um imóvel. Sabe-se que esta também faz referência na tabela “imoveis_municipio”, que trata das informações de cada município cadastrado. A “imoveis_bairro” refere-se aos bairros cadastrados no sistema, com seu respectivo município. A tabela “imoveis_imovel” contém todos os imóveis cadastrados no sistema. Esta faz referência a tabela “imoveis_lado_quadra”, pois cada imóvel está localizado em um lado de uma quadra. Por sua vez uma quadra deve está inserida em um bairro, como mostra a tabela “imoveis_bairro”. Podemos visualizar ainda a tabela “trabalhos_ciclos” que refere-se ao ciclo que o agente irá trabalhar, e a

tabela dos dados dos agentes, a “agentes_agente”.

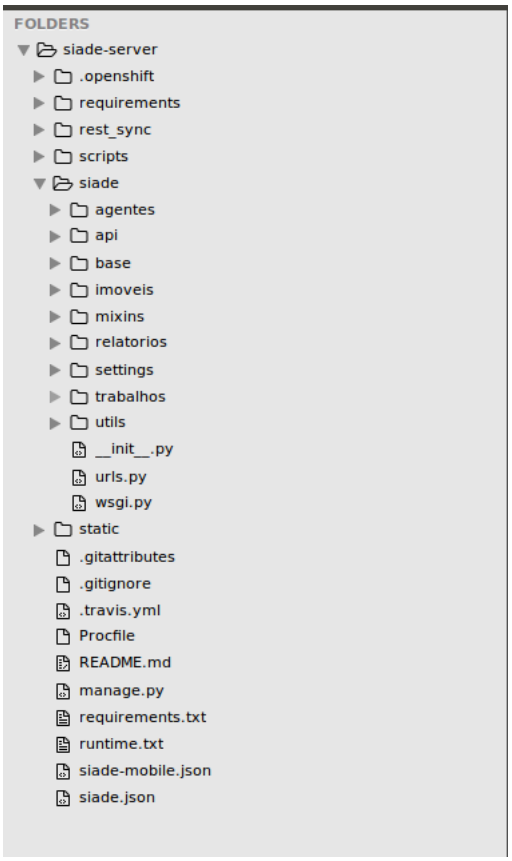
Pode-se notar que a tabela “trabalhos_visita” possuem um identificador para as tabelas “trabalhos_ciclo” , “agentes_agente” e “imoveis_imovel”. A “trabalhos_visita” possui relevância na geração dos relatórios. Ao ser consultada, ela contém todas as informações dos dados coletados pelos agentes e podem ser emitidos em forma de relatório.

3.3 Organização do Projeto com o Django

Como forma de estruturar a organização do projeto no Django, a aplicação foi dividade em módulos (Apps). Estes, visam um maior reuso de código na aplicação. Podendo ser utilizados na criação de novas funcionalidades futuras ou em outros projetos.

Desta maneira, o projeto SIADE é uma App formado pela composição de pequenos módulos internos. Cada um possui funções distintas, conforme podemos visualizar na figura 12:

Figura 12 – Estrutura base do projeto django SIADE.



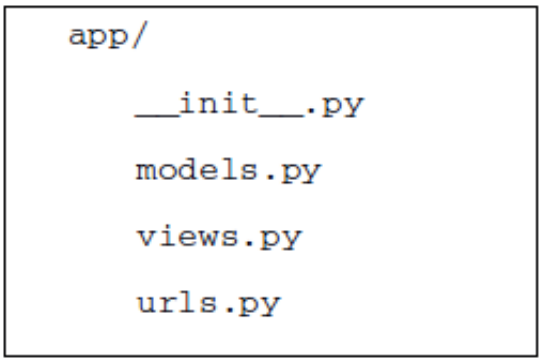
Fonte: Elaborado pelo autor.

Como observado na figura 12, é visível a App *siade*. Seus principais Apps são: *agentes*, *api*, *base*, *imoveis*, *relatorios* e *trabalhos*. São os módulos responsáveis pelas

principais funcionalidades do sistema.

Por padrão, um App ao ser criado, é constituído de quatro arquivos importantes ao seu funcionamento. Como observado na figura 13:

Figura 13 – Estrutura de uma app no Django.



Fonte: Elaborado pelo autor.

É imprescindível salientar que, pode haver a necessidade da importação de arquivos extras (*models, views, urls, pacotes*) para o desenvolvimento das funcionalidades em um App. Outro ponto que merece destaque, é a atualização do BD. Esta acontece de acordo com o especificado na criação das classes nos *models*.

O *___init___.py* é um arquivo vazio. Ele informa um diretório que deve ser inicializado e se é considerado um pacote do Python.

Com o intuito de exemplificar os principais arquivos contidos em um módulo, nas subseções posteriores, será detalhado a App *relatorios* por apresentar uma maior riqueza de detalhes. Este lida com a criação dos Relatórios Diários (D1) e Semanais (D7). É observável que todo e qualquer App do SIADE possui a mesma composição.

3.3.1 model.py

O *models.py*, é o arquivo que contém classes Python. Ele é responsável pelo detalhamento das funcionalidades do sistema, ou seja, os campos essenciais e o comportamento de cada atributo. Deve-se destacar que, cada classe mapeia para uma tabela única do banco de dados.

A App *relatorios* não contém um arquivo *model* específico. Porém, faz uso de todas as classes mapeadas no banco, com o auxílio de importações. Isso acontece, pois manipula os dados de outras classes para a geração de relatórios.

Na figura 14, é exemplificado a estrutura das classes de um *model* da App *imovel*. Por sua vez importado no módulo *relatorios*.

Figura 14 – Exemplo do *Model* da App *relatorios*.

```
10  class UF(BaseModel):
11      '''
12      Uma Unidade Federativa
13      '''
14      nome = models.CharField(max_length=100)
15      sigla = models.CharField(max_length=3)
16
17      def __unicode__(self):
18          return self.nome
19
20      class Meta:
21          verbose_name = 'estado'
22          ordering = ('nome',)
23
24
25  class Municipio(BaseModel):
26      '''
27      Município de uma UF
28      '''
29      nome = models.CharField(max_length=100)
30      uf = models.ForeignKey(UF, verbose_name='UF')
31      codigo = models.IntegerField(
32          blank=True, null=True, verbose_name='código')
33
34      def __unicode__(self):
35          return "%s, %s" % (self.nome, self.uf.sigla)
36
37      class Meta:
38          verbose_name = 'município'
```

Fonte: Elaborado pelo autor.

No *model* da figura acima, é observado o detalhamento de duas classes distintas, a UF (Unidade Federativa) e Municipio. A UF é composta pelos atributos nome e sigla. Já a Municipio por nome, código e o uf. O atributo uf, possui uma *ForeignKey* (chave secundária) da classe UF. Sendo assim, cada município cadastrado, possuirá uma Unidade Federativa.

Na criação de um atributo em uma classe, é necessário especificar uma opção de *field*, correspondente ao tipo da variável. Os *fields* dos atributos nome e sigla na classe UF, são do tipo *CharField*. O primeiro, pode possuir até 100 caracteres, o segundo apenas 3. Já na classe Municipio, os atributos nome, uf, e código, são respectivamente um *CharField* de 100 caracteres, uma *ForeignKey* para a classe UF e um *IntergerField*.

Os *fields* possuem características peculiares. O *verbose_name*, indica o nome que o atributo terá na página. A variável *null*, pode ser do tipo *True* (verdadeiro) ou *False*, sugere salvar um atributo como nulo no Banco de Dados. Outra variável importante é a *max_length*, usada para declarar o tamanho máximo/mínimo dos campos de textos e números. Alguns tipos de *fields* são obrigatórios, outros opcionais.

A classe meta é filha da classe principal. Possui o objetivo de organizar o código. Aprimoram o entendimento por parte dos programadores, sobre o local da alteração dos

nomes das Apps, considerando no plural ou singular. Determina ainda, o atributo da classe que irá manter a ordenação das listas de objetos apresentadas nas páginas e em um determinado App.

3.3.2 view.py

O arquivo *views.py*, representa uma visão do sistema e está associado ao comportamento de pelo menos uma URL. É responsável por solicitar informações a partir do *model* criado e apresentar os dados em um *templates* (HTML) aos usuários. Além do mais, é na *view* que são realizadas consultas ao banco de dados. Na figura 15, é um exemplo da *view* do arquivo *diario.py* da App *relatorios*.

Figura 15 – Exemplo de uma view da app relatorio.

```
11  def form(request):
12      if request.method == 'POST':
13          form = D1Form(request.POST)
14          if form.is_valid():
15              # ao invés de redicionar executar a view
16              return imprimir(request)
17      else:
18          form = D1Form()
19      return render(request, 'relatorios/d1_form.html', {'form': form})
20
21
22  def imprimir(request):
23      data = datetime.strptime(request.POST.get('data'), '%d/%m/%Y').date
24      agentes = request.POST.getlist('agentes')
25      visitas = Visita.objects.filter(data=data)
26      context = {
27          'visitas': visitas.filter(agente__in=agentes)
28                            .order_by('agente', 'imovel__lado__quadra__bairro'),
29          'data': data,
30      }
31      return render_html_or_pdf(request, 'relatorios/d1_imprimir.html',
32                                context, fmt='html')
```

Fonte: Elaborado pelo autor.

Na *view diario.py* do exemplo da figura acima, é notável que ela é composta por dois métodos: def form, e def imprimir.

No método def form, pode-se visualizar o tratamento de formulários através do campo *form*. É a partir desse, irá gerar um formulário do modelo D1Form, contendo todos os seus atributos declarados. No código, o form tratado utiliza o método POST, se este for verdadeiro o formulário é validado e criado, e em seguida executado o método def imprimir. Caso a condição seja falsa, apenas renderiza o *template* de relatórios D1Form.html, passando o próprio form.

No método def imprimir, é feito um request buscando uma data (dia, mês, ano) e a lista de todos os agentes. Após é criado uma variável visita, que filtra todas as visitas pela a data passada. Conseguinte é utilizado o método return, que contém o caminho do arquivo

HTML. Assim, é passado via dicionário *(context)*, todos os valores que foram tratados (as visitas filtradas pelos agentes e ordenadas pelo bairro) na *view*. E posteriormente serão mostrados nas páginas HTML.

3.3.3 urls.py

O arquivo de URLs (urls.py), é composto de endereços de páginas do sistema. Sempre poderão ser visualizados na barra de endereços do seu navegador. O Django usa expressões regulares configuradas no módulo *urls.py* para analisar as URLs das requisições e invocar a *view* apropriada para cada padrão de URL. Como veremos na figura 16 as URLs da App *relatorios*:

Figura 16 – Exemplo de urls da app relatorio.

```
7  urlpatterns = patterns(
8      '',
9      url(r'^diario/$', diario.form, name='diario'),
10     url(r'^semanal/$', semanal.form, name='semanal'),
11
12     url(r'^qrcode/img/$', qrcodes.qrcode_img, name='qrcode-img'),
13     url(r'^qrcode/gerar/$', qrcodes.qrcode_form, name='gerar-qrcode'),
14     url(r'^qrcode/imprimir\.(?P<fmt>\w+)$', qrcodes.qrcode_print,
15         name='imprimir-qrcode'),
16     url(r'^qrcode/imprimir\.pdf$', qrcodes.qrcode_print,
17         name='imprimir-qrcode'),
18 )
```

Fonte: Elaborado pelo autor.

Temos na imagem acima várias URLs distintas. Cada uma, renderiza para um *template* diferente, de acordo com o que foi especificado no arquivo *view*. Pode-se perceber na linha 9 uma URL que começa com *diario/*. Nessa URL, o django se encarrega de verificar o método form do arquivo *diario.py* e renderizar para o *template* declarado no próprio método.

O python utiliza algoritmos que determinam qual o código irá ser executado para buscar uma URL. Esta, por sua vez, é requisita por um usuário ao acessar uma página no sistema. Assim, o django percorre cada URL presente no arquivo *urls.py*, até encontrar a correspondente solicitada. Após encontrar, é importado a *view* relacionada ao caminho desta URL e a lógica de programação. No final da lógica, estará apresentada a estrutura HTML correspondente à URL solicitada, formando então a página HTMl acessada pelo usuário.

3.3.4 Templates (HTML)

Em aplicações web é comum o uso de *templates*. Telas HTML que contém o código necessário para a montagem de uma página web. Os arquivos HTML possuem a finalidade

de exibir informações e dados do sistema para os usuários finais. Por isso, as ultimas linha das *views*, possuem um método *return*. Ele renderiza para um *template* especifico toda a lógica da página. Muitas vezes variáveis, que são substituídas por valores, e *tags*.

Para a mostragem de informações nos *templates*, além de códigos HTML, as páginas utilizam *template tags* e *template filters*. Estas ferramentas usufruem de lógicas criadas em métodos no arquivo *views*, que são renderizados para os *templates* a partir do *return* no fim de cada método.

Template tags são utilizadas para condições a partir do comando if e repetições com o uso do for. Para realizar loops de listas oriundas das lógicas de programação realizadas nas *views*, deve-se utilizar as *template tags*, elas iniciam e terminam com chaves e porcentagem (%...%). Nas figuras 17 e 18, pode ser visualizado exemplos de *template tags* e *templates filters*. Ambos utilizados para mostrar informações no *template* de relatórios d1_imprimir.html.

Figura 17 – Exemplo de template tags.

```
13  {% regroup visitas by agente as visitas_por_agente %}
14  {% for visita_a in visitas_por_agente %}
15  {% regroup visita_a.list by imovel.lado.quadra.bairro as visitas_por_bairro %}
16  {% for visita_b in visitas_por_bairro %}
```

Fonte: Elaborado pelo autor.

Na figura 17, o *template* d1_imprimir.html, recebeu um dicionário de visitas renderizado pela *view diario.py* na App *relatorios*. É utilizado *template tags* para reagrupar todas as visitas por agente. Na segunda linha é percorrido a lista de visitas por agentes com um for, para isso faz uso da variável visita_a. Conseguinte, na terceira linha, é reagrupado toda a lista de visitas, antes havia sida por agentes, agora por bairro. E por fim, é criado novamente um laço de repetição, desta vez percorrendo toda a lista de visitas por bairro, a partir da variável visita_b.

As variáveis criadas nas *template tags*, como visita_a, e visita_b, podem ser utilizadas para as mostrar as informações contidas em cada for, a partir de *template filters*. Como demonstrado na figura 18, abaixo.

Figura 18 – Exemplo de template filters.

```
<table class="table">
  <thead>
      <tr>
    <th>Agente:</th>
    <th>Ciclo:</th>
    <th>Data:</th>
    <th>Bairro:</th>
    <th>Atividade:</th>
      </tr>
  </thead>
  <tbody>
      <tr>
      <td>{{ visita_a.grouper }}</td>
      <td>{{ visita_a.list.0.ciclo }}</td>
      <td>{{ data }}</td>
      <td>{{ visita_b.grouper }}</td>
      <td>{{ visita_a.list.0.ciclo.get_atividade_display }}</td>
      </tr>
  </tbody>
</table>
```

Fonte: Elaborado pelo autor.

É verificável, na figura acima, um exemplo de como *template filters* são utilizado no sistema. Com o intuito de mostrar dados em tela ao usuário. O primeiro *template filter*, visita_a, identifica uma lista de visitas por agente, especificado na *template tag*, informações como, agente, ciclo, e atividade. Já a *template filter* visita_b, mostra todas as visitas agrupadas por bairro, declarada na *template tag* vistas anteriormente.

3.3.5 Consultas em Banco

A consulta das informações presentes nos relatórios, é realizada através do ORM e do modelo de dados (*Objetcts*). A partir dos métodos contidos no *QuerySet*, foram recuperados no BD, todos os registros de uma tabela específica, ou que satisfaçam uma ou mais condições. Ver figura 19:

Figura 19 – Exemplos de colsultas ao Banco de dados.

```
def imprimir(request):
    data = datetime.strptime(request.POST.get('data'), '%d/%m/%Y').date
    agentes = request.POST.getlist('agentes')
    visitas = Visita.objects.filter(data=data)
    context = {
        'visitas': visitas.filter(agente__in=agentes)
                          .order_by('agente', 'imovel__lado__quadra__bairro'),
        'data': data,
    }
    return render_html_or_pdf(request, 'relatorios/d1_imprimir.html',
                              context, fmt='html')
```

Fonte: Elaborado pelo autor.

Na figura 19, a variável *agentes* na terceira linha, está recebendo a lista dos agentes recuperados com o auxílio do método *.getlist()*. A condição que satisfaz a consulta está no fim da mesma linha e entre parênteses, no caso ('agentes').

Na segunda linha do código, temos uma variável do tipo *data*. Como pode ser verificado, na linha 4, ela é fator limitante do filtro *.filter*, na coleção de objetos *(objects)* da tabela *Visita* no banco. Assim, é observado que o *QuerySet* em questão, possui filtros, que limita a busca dos registro consultados. Desta forma, está sendo recuperado uma coleção de visitas, filtradas por uma data específica.

Na linha 6 do código, o método *.order_by()* foi utilizado em conjunto com o *.filter()*. O resultado dessa junção, seria uma consulta no *QuerySet* visitas (linha 4). Por sua vez, filtrada novamente por agentes (linha 3). E por fim, o *.order_by*, ordena a consulta, por agentes e bairros. Os parâmetros utilizados nos métodos *.filter()*, são informados seguindo a estrutura *campo___condição = valor*, ou seja, *agente___in = agentes*. Há dois caracteres de sublinhado (_) entre agente e in. O ORM do Django usa esta sintaxe para nomes de campos separados ("agente") e as operações ou filtros ("in").

Todos os dados consultados para a representação dos relatórios, seja D1 ou D7, faz o uso de métodos do *QuerySet*. Estes, são essências na manipulação das consultas no Banco de Dados. Em uma analogia com a linguagem SQL, o *QuerySet* representa o comando *SELECT* e os filtros representam as cláusulas *WHERE*.

4 APRESENTAÇÃO DO SIADE

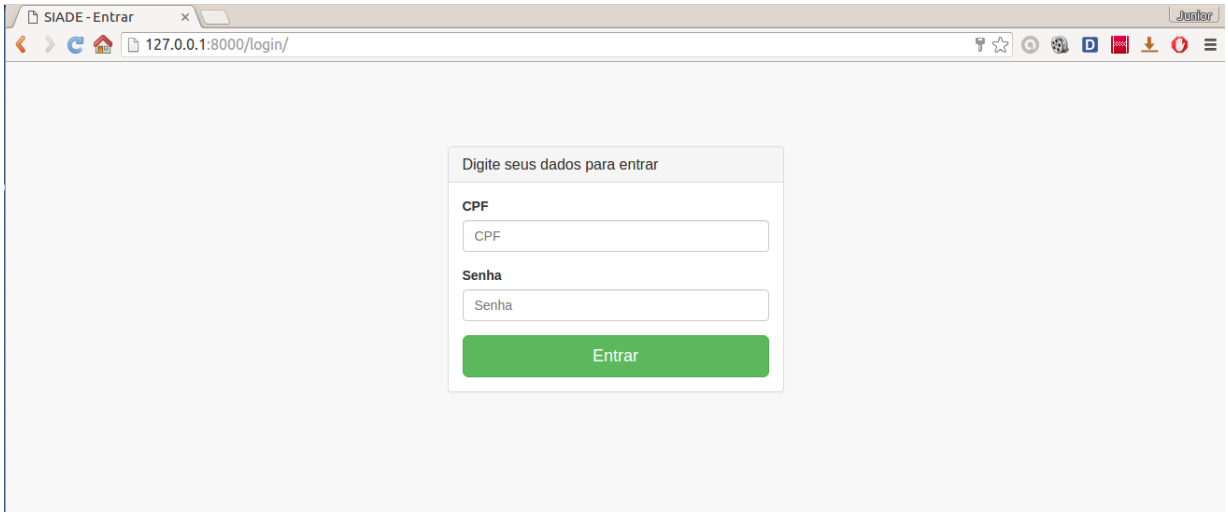
Este capítulo possui o objetivo de realizar uma apresentação do SIADE, focando no desenvolvimento de todos os módulos da aplicação.

A aplicação atendeu as expectativas e os requisitos dos Agentes de Endemias da cidade de Pau dos Ferros. Estes puderam fazer testes reais no sistema em execução. Sendo assim, houve a coleta de dados por parte dos agentes e os supervisores puderam visualizar as informações no sistema. A seguir serão apresentadas as telas do SIADE.

4.1 SIADE

Na figura 20, pode ser vista a tela de login usada por Supervisores (Admin) e Agentes.

Figura 20 – Login da Tela de Administração do SIADE.

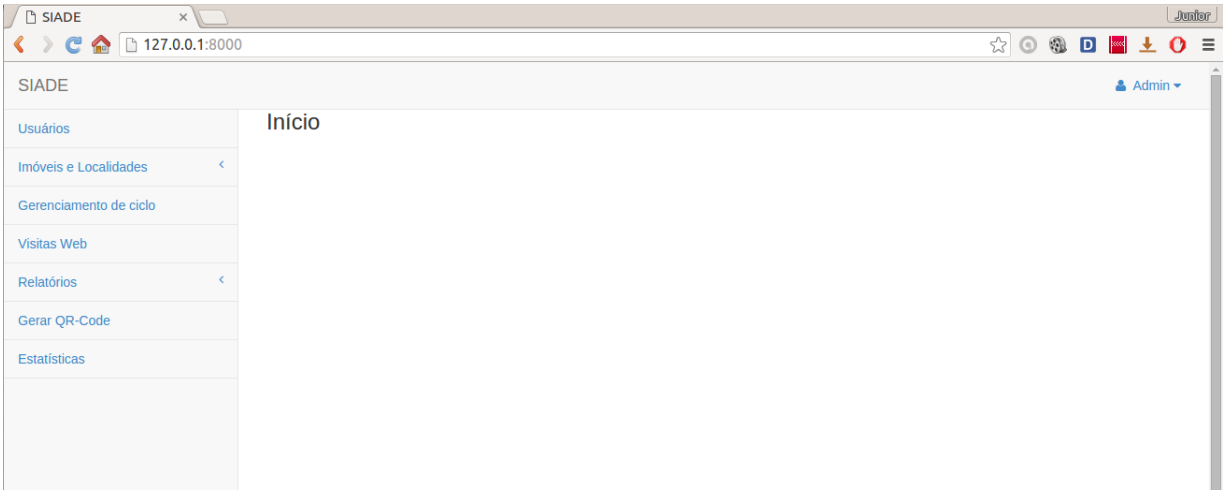


Fonte: Tela de Login do SIADE.

Para o primeiro acesso ao sistema, o usuário necessita logar com o CPF e a senha. Através do Admin do Django, é gerada uma conta de administrador, cadastrando o Supervisor e os agentes.

Ao estar logado no sistema, o usuário pode navegar a partir de um menu lateral, como pode ser observado na figura 21.

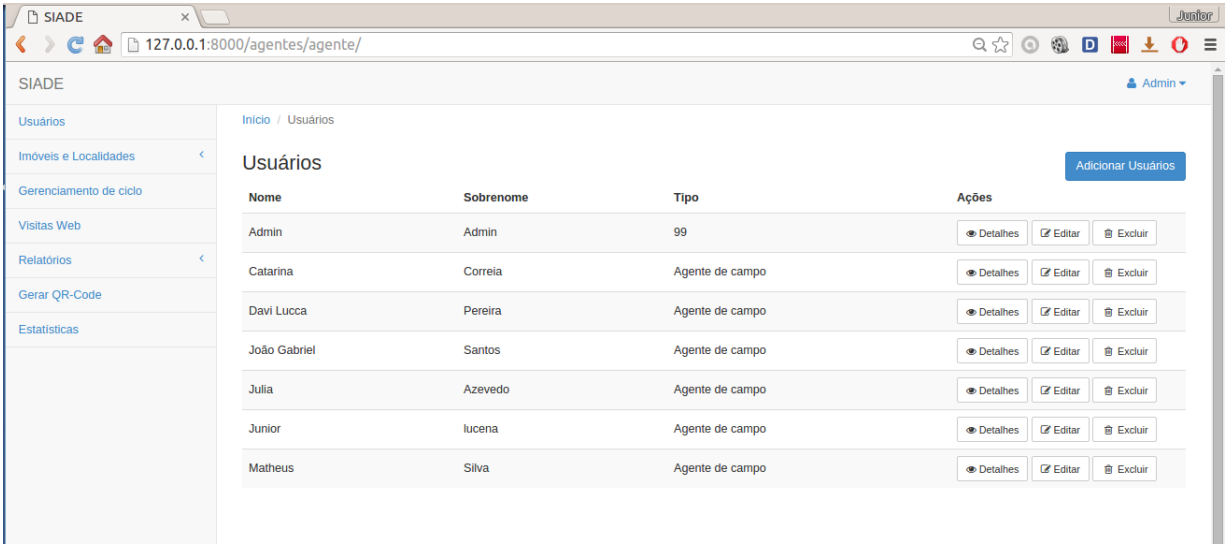
Figura 21 – Tela de Administração do SIADE.



Fonte: Tela de Administração do SIADE.

Este menu foi criado no intuito de facilitar a usabilidade dos usuários, provendo um maior visualização de conteúdo e manuseio no sistema. Deve-se deixar claro, dependendo do tipo de usuário ou da permissão que ele possua, algumas funcionalidades do sistema são bloqueadas para uma maior segurança. Na figura 22, será mostrado a listagem dos tipos de usuários cadastrados na aplicação.

Figura 22 – Tela da listagem de Usuários do SIADE.



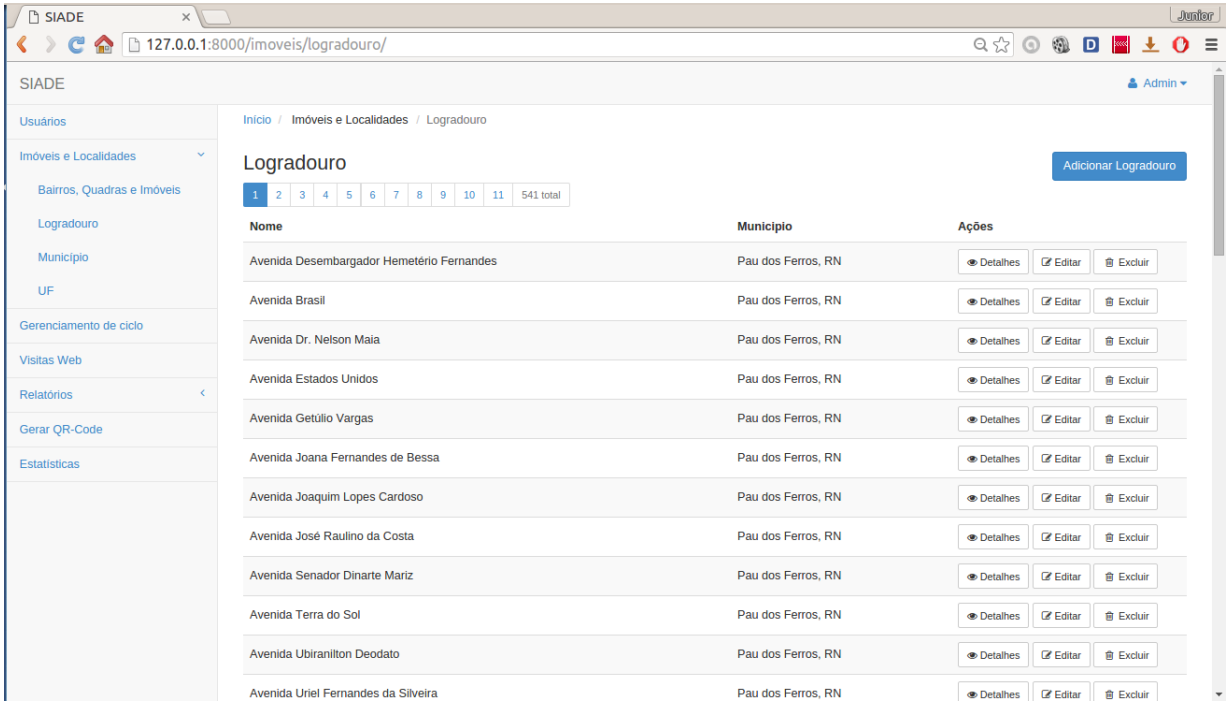
Fonte: Tela do SIADE.

No caso do usuário Supervisor (admin), ele possui o acesso total as funcionalidades do sistema. Diferente do agente de campo que não possui acesso a tela de gerenciamento de ciclo. Nesta função pode-se alterar as informações de um usuário já cadastrado, removê-lo ou apenas visualizar os detalhes de seus dados. Ainda é possível adicionar um novo usuário,

por meio de formulários com os campos: CPF, nome, sobrenome, nascimento, e-mail, telefone, código do usuário, tipo do usuário e o município a que ele pertence.

Conseguinte, podemos observar o menu Imóveis e Localidades. Neste pode ser cadastrado: estados, municípios, cidades, bairros, ruas, quadras, lados de uma quadra e um imóvel. Dados estes que são fundamentais nas visitas de campo realizadas pelos agentes. Na figura 23 é apresentada a listagem de Ruas (Logradouros) da Cidade de Pau dos Ferros.

Figura 23 – Tela da listagem de Ruas do SIADE.

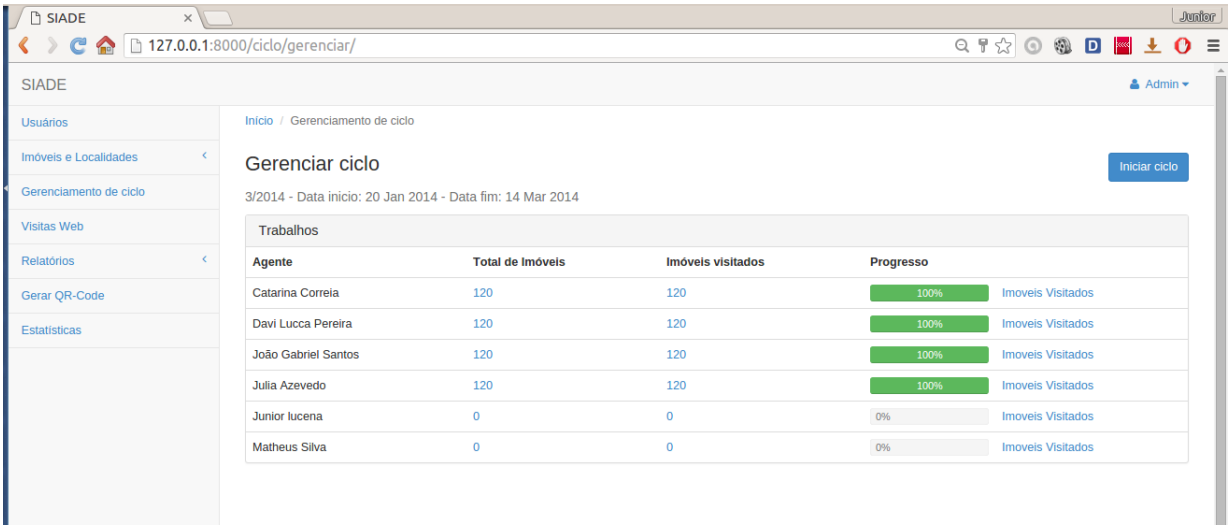


Fonte: Tela do SIADE.

Como todos os cadastros do Menu Imóveis e Localidade possui as mesmas características, ou seja, CRUD. Onde os dados podem ser atualizados, removidos ou ser inserido um novos cadastros. Optou-se por mostrar apenas a tela da listagem de Ruas.

Seguindo o Menu lateral, podemos visualizar o Gerenciamento de Ciclo. É usado pelo Supervisor para gerenciar o ciclo dos agentes de endemias. Sempre será mostrado em tela o Ciclo atual, seja este aberto ou fechado, como é observável na figura 24:

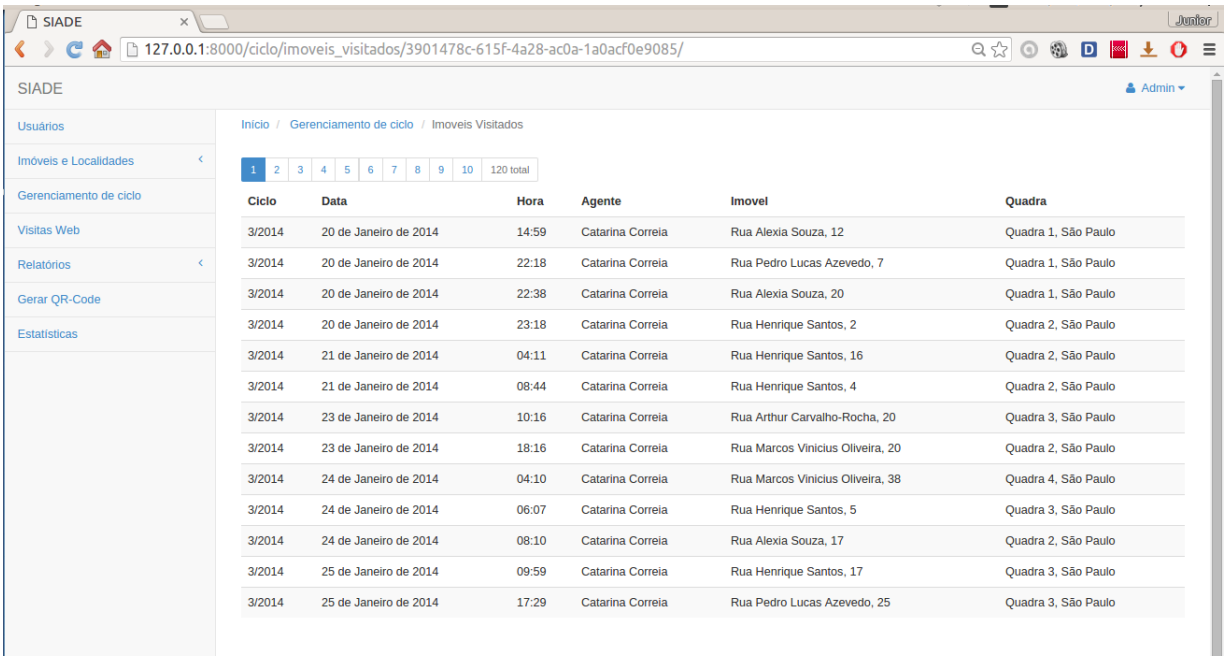
Figura 24 – Tela da Gerenciamento de Ciclo do SIADE.



Fonte: Tela do SIADE.

Quando o Supervisor entra na tela de Gerenciamento de Ciclo, ele visualiza a data inicial e final do ciclo, os agentes que foram dispostos a executar a visita e coletar os dados, a quantidade total de imóveis e a quantidade de imóveis visitados. É mostrado também o progresso em porcentagem das visitas realizadas por cada agente se aproximando do fim. Na figura 25, podemos observar os imóveis visitados por um agente específico.

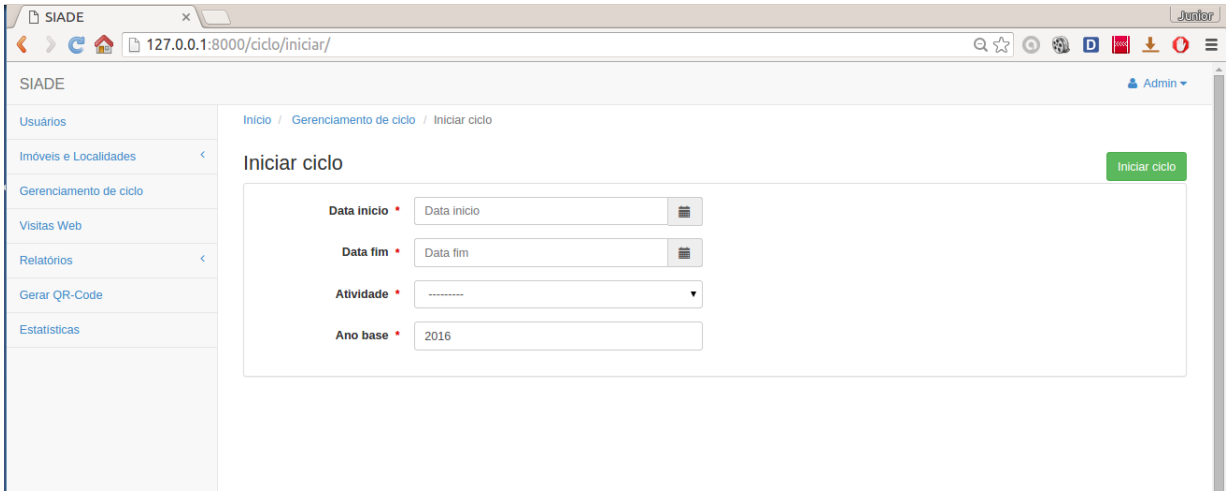
Figura 25 – Tela dos imóveis visitados por um agente no Gerenciamento de Ciclo do SIADE.



Fonte: Tela do SIADE.

Quando não existe ciclo cadastrado e os agentes necessitam iniciar os trabalhos, um novo ciclo é iniciado. Como observável na figura 26:

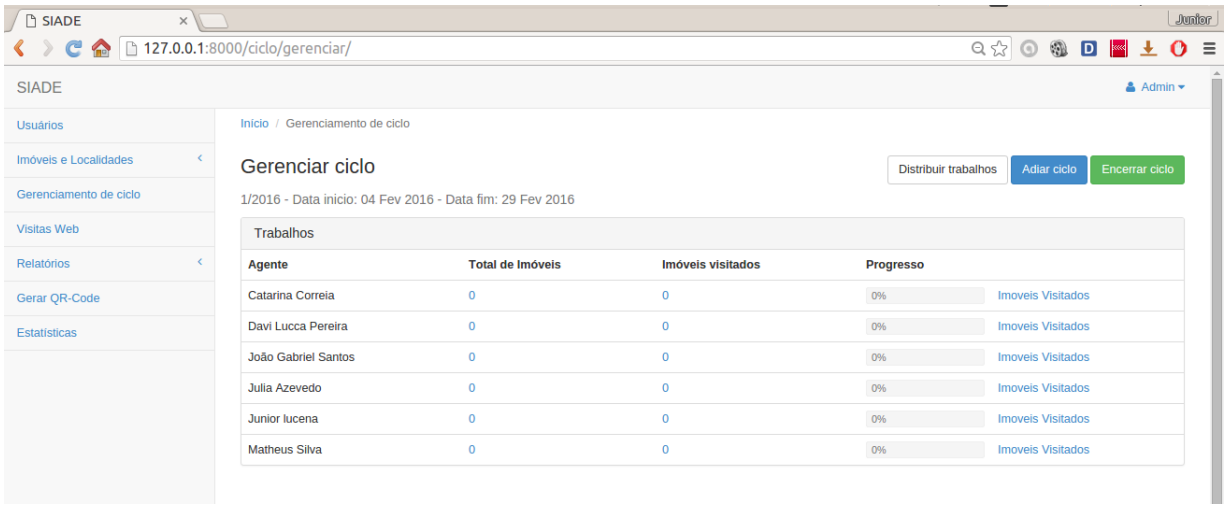
Figura 26 – Tela da Criação de um novo Ciclo do SIADE.



Fonte: Tela do SIADE.

Após a abertura do novo Ciclo, o Supervisor poderá gerenciá-lo. Ver figura 27:

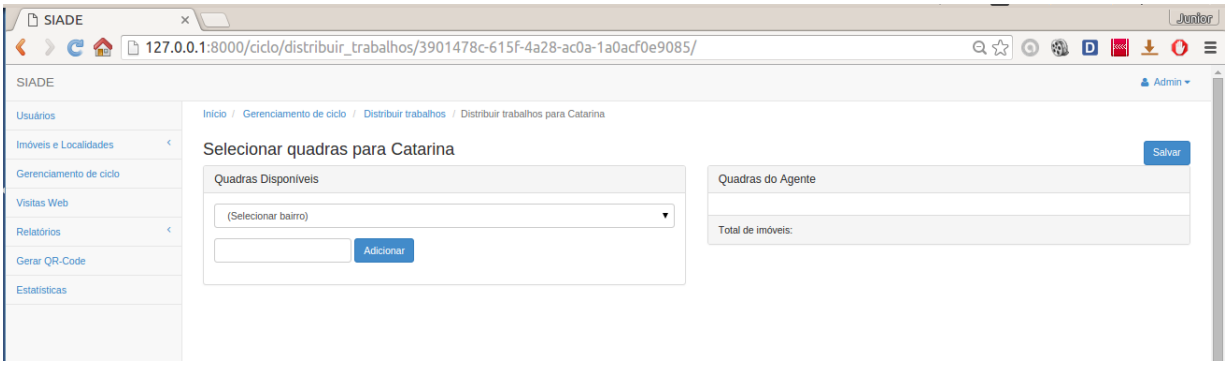
Figura 27 – Tela da Criação de um novo Ciclo do SIADE.



Fonte: Tela do SIADE.

Neste gerenciamento, o Supervisor escolherá os agente que irá trabalhar e distribuir os imóveis para a realização das respectivas visitas em campo. Ele poderá escolher o Bairro para o agente e a quadra, automaticamente os lados da quadra e seus respectivos imóveis serão associados. Como podemos visualizar na figura 28:

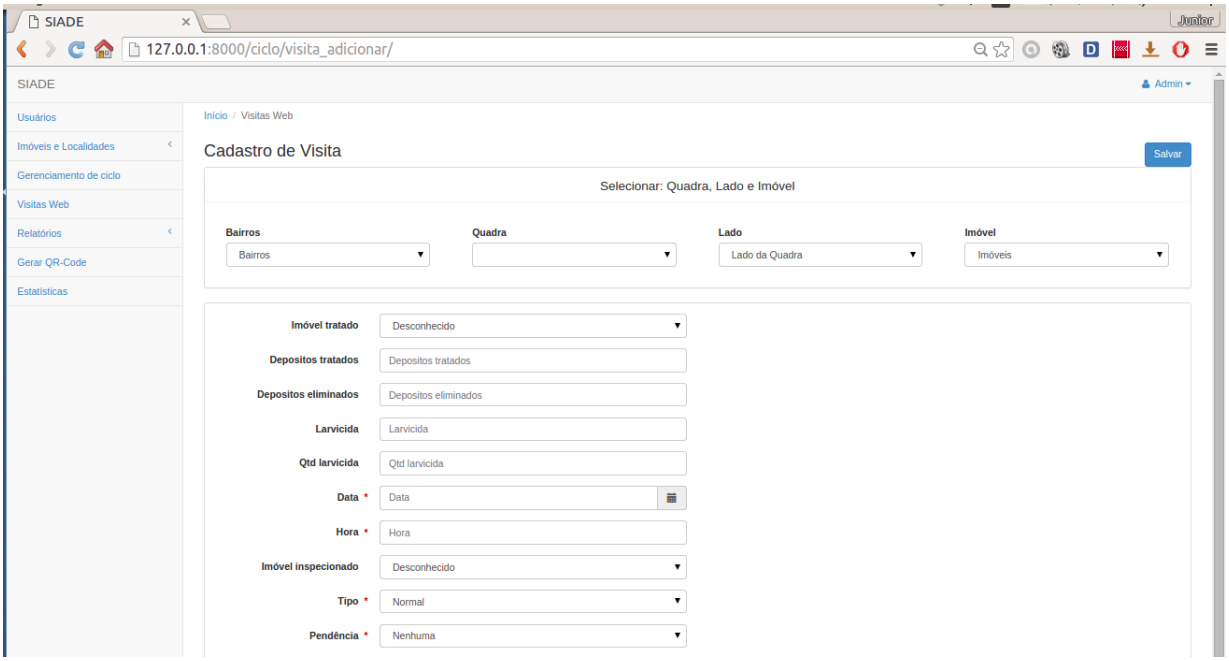
Figura 28 – Tela da Distribuição de Trabalho no Gerenciamento de Ciclo do SIADE.



Fonte: Tela do SIADE.

Seguindo a ordem do Menu lateral, após o Gerenciamento de Ciclo temos o Visitas Web. Ver figura 29:

Figura 29 – Tela da Criação de um novo Ciclo do SIADE.



Fonte: Tela do SIADE.

Neste caso, o usuário (Supervisor ou Agente) pode acrescentar uma nova visita ao sistema. Após preencher as informações, basta apenas salvar os dados, assim a visita no imóvel estará completa.

Pode-se também a partir do Menu Lateral Gerar QrCode, criar uma imagem que possui um código ou informação em seu interior, um QRCode. Neste caso é utilizado para identificar um imóvel no ato da realização da visita de um agente. Bastando apenas que o Agente aponte com o leitor de QrCodes já instalado no tablet, assim conseguirá

identificar os dados do imóvel. Seguindo, é observável o Menu Estatísticas, porém é apenas um módulo do sistema que está em desenvolvimento. No seção posterior é apresentado os Relatórios do tipo D1 (Diário) e D7 (Semanal).

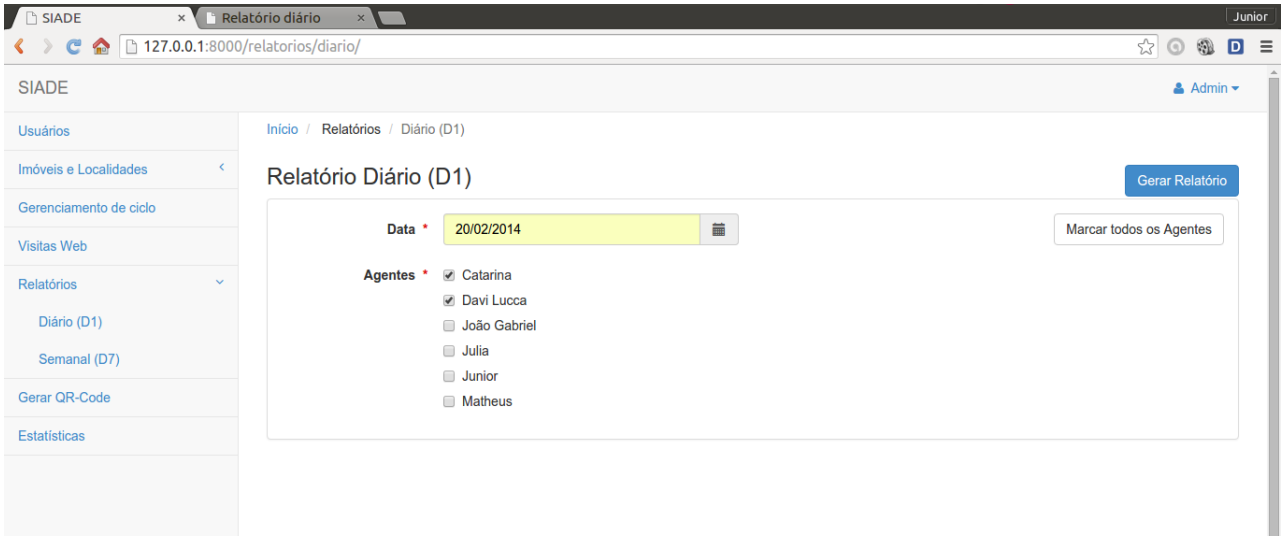
4.2 Módulo Relatório

O módulo relatório do SIADE, é dividido em relatórios do tipo D1 e D7. Trata de todas as informações coletadas pelos agente de campo (agente de endemias), cujas quais podem servir de apoio ao monitoramento das informações e o trabalho realizado pelos agentes.

4.2.1 Relatório D1 (Diário)

No relatório D1 (Diário), o supervisor pode visualizar as informações coletadas por um ou mais agentes. Dados de um dia de trabalho. Na figura 30 estará representado o filtros de busca do relatório diário.

Figura 30 – Filtros de busca ou pesquisa do Relatório D1.



Fonte: Tela do filtros de busca do SIADE.

O supervisor poderá selecionar uma data que esteja ou não dentro de um ciclo trabalhado por um agente. Em seguida, deve escolher todos ou quantos agentes. Para facilitar a seleção de todos os agentes, existe um botão *Marcar todos os agentes*, na parte superior do lado direito da tela. Após isso, ele pode gerar o relatório.

Figura 31 – Relatório D1 gerado

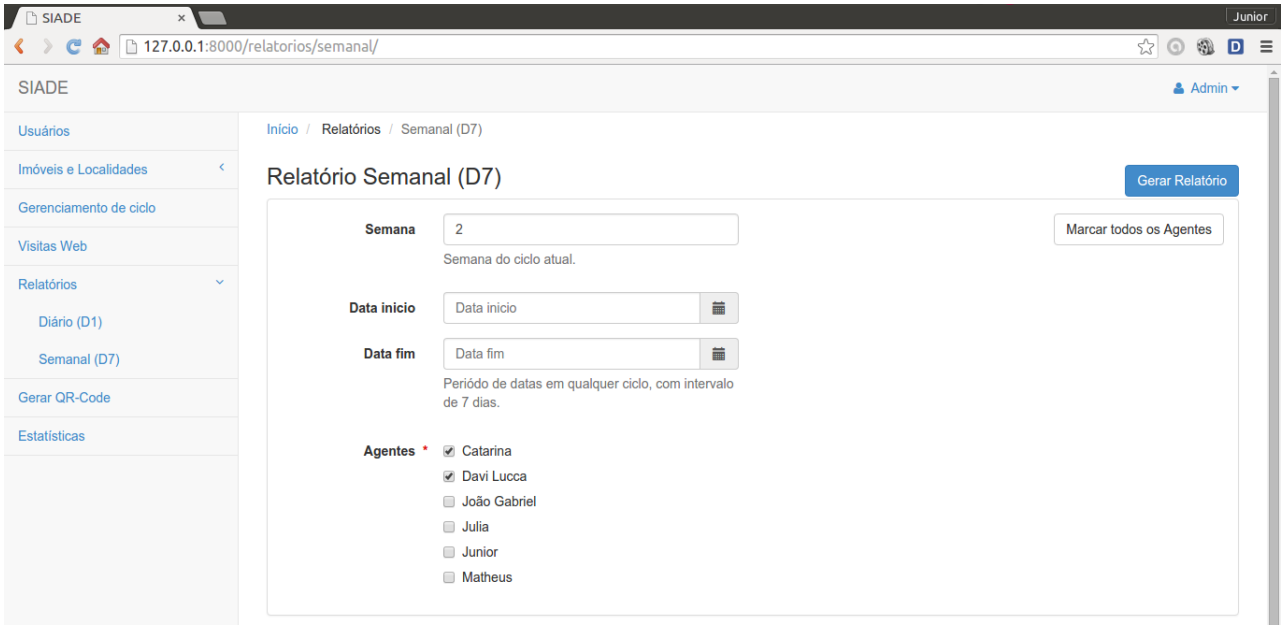
Fonte: Relatório D1 do SIADE.

A figura 31, apresenta o relatório gerado, o qual, contém todas a informações pertinentes ao supervisor, como: Os dados do agente, ciclo trabalhado, data do trabalho realizado, o tipo de atividade e bairro. Como ainda os dados da pesquisa entomológica realizada pelo agente em campo.

4.2.2 Relatório D7 (Semanal)

O relatório D7 (semanal), permite ao supervisor visualizar a informações de sete dias consecutivos trabalhados por um ou mais agentes. Abaixo, na figura 32 é verificado os filtros de busca ou pesquisa do relatório.

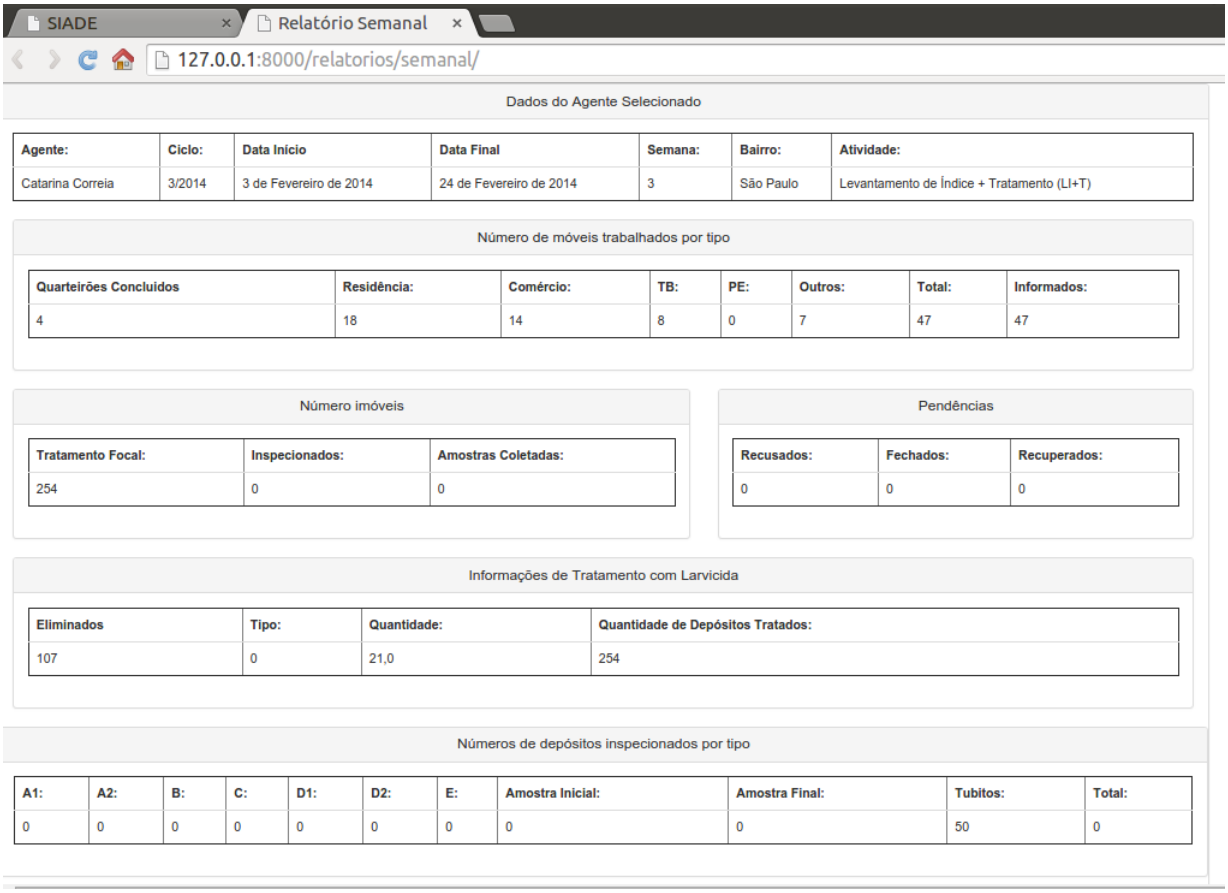
Figura 32 – Filtros de busca ou pesquisa do Relatório D7.



Fonte: Tela do filtros de busca do SIADE

Nesse formulário, o supervisor poderá informar o número semana do ano ou o período do ano, o qual não pode passar de sete dias. Para facilitar a seleção de todos os agentes, existe um botão *Marcar todos os agentes*, na parte superior do lado direito da tela. É também necessário escolher os agentes, ou apenas um agente para gerar o relatório. Podemos observar o relatório gerado na figura 33.

Figura 33 – Relatório D7 gerado



Fonte: Relatório D7 do SIADE..

Acima temos a visualização de todas as informações consultadas em banco e mostradas no relatório semanal. Todas as informações são de extrema importância ao acompanhamento dos agentes por parte do supervisor. Pois podem fazer um acompanhamento mais preciso sobre o trabalho realizado pelo agente, se está atingindo os objetivos esperados pela Secretaria de Saúde.

CONCLUSÃO

Sob todo o conjunto de informações apresentadas até o momento, podemos salientar que o sistema web SIADE proporcionou um vasto conhecimento e aprimoramento por parte dos desenvolvedores, devido o uso de novas tecnologias, bem como pela contribuição para os usuários finais do Software.

O sistema possui uma grande importância tanto para os agentes de endemias por agilizar um trabalho que antes se tornava cansativo, repetitivo e demorado, lidando manualmente com uma vasta quantidade de informações. Como pelo fato de ser um novo sistema de saúde que favorece o controle das informações dos focos de dengue e a divulgação destes informativos para a população em tempo real.

Em relação as ferramentas utilizadas para o desenvolvimento do sistema, foi possível mostrar que em conjunto, estas podem aumentar a flexibilidade na utilização e produtividade de uma aplicação. Como ainda, estruturas (apps django) flexíveis são capazes de facilitar uma implementação mais precisa, ágil e eficiente nas relações entre os dados.

Pode-se observar que em termos de desenvolvimento, houve uma conclusão das necessidades principais propostas para o funcionamento sistema, como gerenciamento de informações pelo supervisor, a coleta dos dados da população e das suas respectivas residências, geração de relatórios, monitoramento e acompanhamento do trabalho realizado pelos agentes. O projeto continua em desenvolvimento, na finalidade de atingir um sistema mais preciso, eficiente e detalhista, que possa ser incrementado de acordo com as necessidades dos serviços de saúde. Não pode deixar de especificar que houve uma base conceitual de pesquisa na implementação, afim de criar um sistema que atenda as necessidades mais urgentes da problemática.

Quando houver a conclusão completa do sistema, espera-se que possa atender não somente a problemática da dengue, mas outras doenças transmitidas a partir do Aedes Aegypti como: Calazar, microcefalia, Zica vírus, Chico Cungunha, febre amarela. Estas são todas doenças virais, que podem ser disseminadas a partir do mosquito. Espera-se que possa atender a estes propósitos, havendo um crescimento gradativo na diminuição dos focos de contaminação e se transforme em um produto ou serviço que possa ser utilizado em qualquer meio e/ou em qualquer problema epidêmico.

Como trabalhos futuros ressaltam-se a implementação de algumas funcionalidades como a marcação de focos de dengue em mapa geológico da cidade de Pau dos Ferros, estatísticas, geração dinâmica de gráficos.

Referências

ALCHIN, M. “Pro Django”. *Apress*, 2 nd edition Citado na página 28.

BENITO, G. A. V., LICHESKI, A. P. Sistemas de Informação apoiando a gestão do trabalho em saúde. *Revista Brasileira de Enfermagem*, 62, 3, p. 447-50. Santa Catarina, 2009. Citado 2 vezes nas páginas 16 e 18.

BORGES, Luis E. Python para desenvolvedores. 2a. ed. Rio de Janeiro – RJ: Edição do Autor, 2010 Citado 2 vezes nas páginas 24 e 27.

BRANDÂO, J. M. N. Aprendendo Django no Planeta Terra. vol 2, 1. ed. [s.n.], 2009 Citado 2 vezes nas páginas 31 e 32.

BRASIL. Diretrizes Nacionais para a Prevenção e Controle de Epidemias de Dengue. *Ministério da Saúde*, Brasília/DF, 2009. Citado na página 15.

BRAUDE, Eric. Projeto de Software: Da programação à arquitetura: uma abordagem baseada em Java. *Bookman*, Porto Alegre, 2005. 619 p. Citado na página 25.

CAIXETA, D. M.; SOUSA, F. G. A utilização de ferramentas e técnicas de geoprocessamento na identificação e análise das áreas de maior ocorrência de casos de dengue em Goiânia-GO. *In: Simpósio Brasileiro de Sensoriamento Remoto (SBSR)*, 13., 2007, Florianópolis. Anais... Florianópolis: INPE; 2007. Artigos, p. 2373-2379. CD-ROM, On-line. ISBN 85-17-00031-7. Disponível em: <http://marte.dpi.inpe.br/col/dpi.inpe.br/sbsr@80/2006/11.16.00.40.42/doc/2373-2379.pdf>. Acesso em Agosto de 2015. Citado na página 15.

COMAI, Sara; CARUGHI, Giovanni Toffetti. A Behavioral Model for Rich Internet Applications. *Web Engineering Lecture Notes in Computer Science*, v. 4607, p. 364-369, 2007. Citado na página 21.

Corrêa, P. R. L; Franca E.; Bogutchi, T. F. Infestação pelo Aedes aegypti e ocorrência da dengue em Belo Horizonte. *Revista da Saúde Pública*, v. 39, n.1. p. 33-40. 2005. Citado na página 15.

CRUZ, J. L. D. Django Web. Disponível em: <http://djangoweb.blogspot.com/2009-/01/um-pouco-de-historia.html>, acesso em 14 Setembro 2015. Citado 2 vezes nas páginas 28 e 29.

CUSTODIO, Glauco. Django Web. Disponível em: <http://blog.glaucocustodio.com/2012/07/31/porque-usar-um-framework/>, acesso em 10 Setembro 2015. Citado na página 26.

DIAS, Marcos A. M. Introdução a Python e Django. *Slideshare*, slide 16. Disponível em: <http://pt.slideshare.net/ledsifes/introduo-a-python-e-django>, acesso em Setembro de 2014. Citado na página 30.

Django Documentation. Documentação do Django. Disponível em: <https://docs.djangoproject.com/en/1.8/>, acesso em 13 Setembro 2015. Citado na página 31.

Django Software Foundation. Django Software Foundation. *"Sítio do Django"*. Disponível em: <https://www.djangoproject.com/>, acesso em 14 Setembro 2015. Citado na página 28.

GERMOGLIO, G. Arquitetura de software. , Texas: Rice University, 2010 Citado na página 22.

HOLOVATY, A. and KAPLAN_MOSS, J. 2009. "The Definitive Guide to Django: Web Development done right". *Apress*, 2 nd ed. 2009. Citado 3 vezes nas páginas 29, 30 e 32.

LARMAN, C. Utilizando UML e Padrões.. *Bookman*, 3. ed. Brasil, 2005. 695 p. Citado na página 25.

LINAJE, Marino; PRECIADO, Juan Carlos; SÁNCHEZ-FIGUEROA, Fernando. A method for model based design of rich internet application interactive user interfaces. *Lecture Notes in Computer Science*, v. 4607, p. 226-241, 2007. Citado na página 21.

MAZIERO, Carlos. Introdução aos Serviços de Rede. Disponível em: <http://wiki.inf.ufpr.br/maziero/doku.php?id=espec:introducao>, acesso em Setembro de 2015. Citado na página 22.

NASCIMENTO, P. S. R.; PETTA, R. A.Uso de Sistema de Informação Geográfica na dispersão de casos de dengue no Estado do Rio Grande do Norte. *Anais XV Simpósio Brasileiro de Sensoriamento Remoto – SBSR*, INPE p. 8421, Curitiba, 2011 Citado 3 vezes nas páginas 15, 16 e 17.

Operating Dev. Web Technology Family Tree. Disponível em: <http://www.operatingdev.com/2013/01/humans-vs-technology-can-we-standardize-one-without-the-other/web-technology-family-tree/>, acesso em 28 Setembro 2015. Citado na página 24.

PRESSMAN, R. S. Engenharia de Software. *McGraw-Hill*, 6a . ed. São Paulo, 2006 Citado na página 21.

Python Software Foundation. Python Programming Language – Official Website. jan. 2010. Disponível em: <http://www.python.org>, acesso em Setembro de 2015. Citado na página 27.

RAMALHO, Luciano. Django ORM: o básico. Disponível em: <http://turing.com.br/material/acpython/mod3/django/orm1.html>, acesso em 19 de outubro de 2015. Citado na página 32.

SANTANA, Osvaldo; GALESI, Thiago. Python e Django: desenvolvimento ágil de aplicações web. *Novatec*, São Paulo, 2010 Citado na página 27.

SEBESTA, R. W. Concepts of Programming Languages. *Pearson Education*, 7. ed. [S.l.], 2006. ISBN 0-321-312511. Citado na página 23.

SILVA, M. G. N. M.; BENNEDETI, Rodrigues, M. A. B.; ARAUJO, R. E. Sistema de aquisição e processamento de imagens de ovitrampas para o combate a dengue. *Revista Brasileira de engenharia Biomédica*, 28,4, p. 364-374, 2012. Citado na página 17.

THOMAS, D.; HANSSON, D. H. Agile Web Development with Rails. *Dallas: The Pragmatic Programmers LLC*, 2a . ed. 2007. ISBN 0-9776166-3-0. Citado na página 23.

THOMAZ, Fábio Eduardo. Estudo do Framework Django e da sua Utilização no Desenvolvimento de uma Aplicação Web para o Controle de Alocação de Professores do Instituto Superior Tupy. *Instituto Superior Tupy*, Sociedade Educacional de Santa Catarina - SOCIESC. Joinville - SC, 2007. 95 p. Citado na página 33.

UOL. Casos de Dengue Caem 59 por cento no RN em 2013. Disponível em: <http://ne10.uol.com.br/canal/cotidiano/nordeste/noticia/2013/02/26/casos-de-dengue-caem-59_porcento-no-rn-em-2013-diz-ministerio-da-saude-401496.php>. Acesso em Agosto de 2015. Citado na página 15.