CNPJ Insight - Relatório

Dávila Merieles Eduardo Adame Kayo Yokoyama Lucas Braga

10 de dezembro de 2023

1 Introdução

O projeto CNPJ Insight tem como objetivo analisar os dados de empresas brasileiras disponibilizados pela Receita Federal. O projeto foi desenvolvido principalmente com Django e sua versão mais atualizada está hospedada na nuvem (utilizando Microsoft Azure) e pode ser acessada em `http://20.195.169.122:8000/`.

2 Descrição do projeto

2.1 Estórias de usuário

Abaixo, exibimos uma versão simplificada de nossa tabela de Estórias de usuário, que pode ser acessada de forma completa em: `https://docs.google.com/spreadsheets/d/15gOOOYpY2VHKhWmfb38dpLy1ZO2NmH_nlopBd-CqTgw/edit?usp=sharing`

Como um (Tipo de Usuário)	Eu preciso (Fazer alguma tarefa)	Para que eu possa (Objetivo)
Usuário do sistema	Avaliar a confiabilidade de uma empresa	Saber por que empresa procurar para um melhor serviço
Usuário do sistema	Buscar informações de empresas por CNPJ	Obter detalhes relevantes como localidade, proprietário e data de criação
Administrador do sistema	Gerenciar contas de usuário	Manter a segurança do sistema
Desenvolvedor do sistema	Dockerizar o aplicativo	Simplificar a implantação e gerenciamento
Desenvolvedor do sistema	Criar um sistema de indexação eficiente para os dados empresariais	Indexação escalável, atualização do índice automatizada, busca rápida e eficiente das informações empresariais
Usuário do sistema	Comparar duas empresas	Melhorar a qualidade da minha análise
Usuário do sistema	Criar uma conta e fazer login	Acessar recursos adicionais
Administrador do sistema	Fazer backup do banco de dados	Aumentar a redundância dos dados armazenados
Usuário do sistema	Exportar os resultados da pesquisa	Realizar uma análise adicional
Usuário do sistema	Acessar meu histórico de pesquisas	Revisitar empresas e análises que consultei anteriormente

2.2 Casos de uso

Abaixo, ao invés da forma tradicional de tabelas, exibimos os diagramas de casos de uso.

Pesquisa por Empresas

- **Atores:** Usuário
- **Premissas:** O sistema possui um banco de dados atualizado com informações sobre empresas.
- **Fluxo de Eventos:**
  1. O usuário acessa a página inicial do aplicativo.
  2. O usuário insere o CNPJ básico ou razão social da empresa.
  3. O usuário clica no botão de pesquisa
  4. O usuário é redirecionado para a página com informações da empresa.
- **Frequência de Uso:** Alta
- **Pre-condições:** Nenhuma
- **Pós-condições:** O sistema exibe os resultados da pesquisa com base nos critérios fornecidos.
- **Problemas:** A busca pode não retornar resultados precisos se o nome ou CNPJ inserido pelo usuário estiver incorreto ou não estiver presente na base de dados. Além disso, a lentidão na busca pode impactar a experiência do usuário.

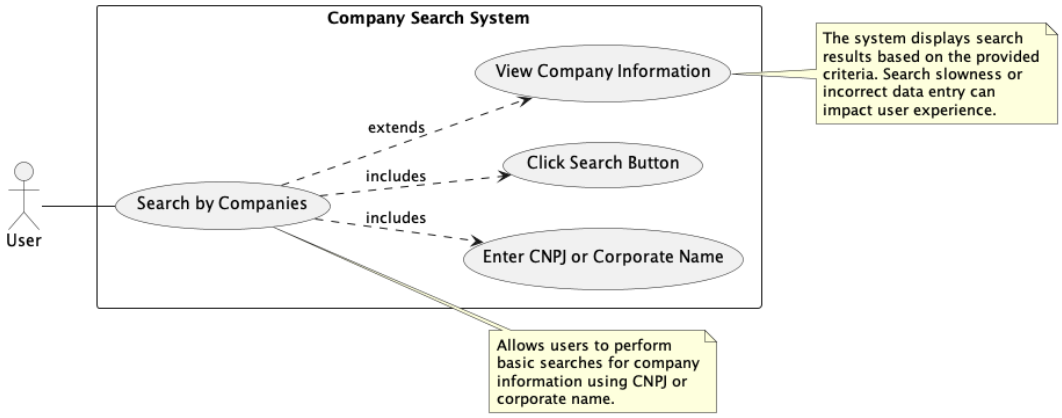

Figura 1: Pesquisa por empresas

Visualização de Detalhes da Empresa

- **Atores:** Usuário
- **Premissas:** O sistema possui informações detalhadas sobre as empresas.
- **Fluxo de Eventos:**
  1. O usuário realiza uma pesquisa utilizando critérios como CNPJ básico e razão social
  2. O sistema exibe uma lista de resultados correspondentes à pesquisa.
  3. O usuário clica em uma empresa específica na lista para visualizar mais detalhes.
  4. O usuário seleciona a opção de "Comparar com outra empresa"
  5. O usuário seleciona a segunda empresa.
  6. O sistema apresenta as informações de empresas lado a lado.
- **Frequência de Uso:** Alta
- **Pre-condições:** A pesquisa retorna resultados correspondentes à empresa desejada.
- **Pós-condições:** O usuário visualiza detalhes específicos da empresa selecionada.
- **Problemas:** Possíveis lentidões em períodos de alta demanda; informações incompletas ou indisponíveis para algumas empresas.

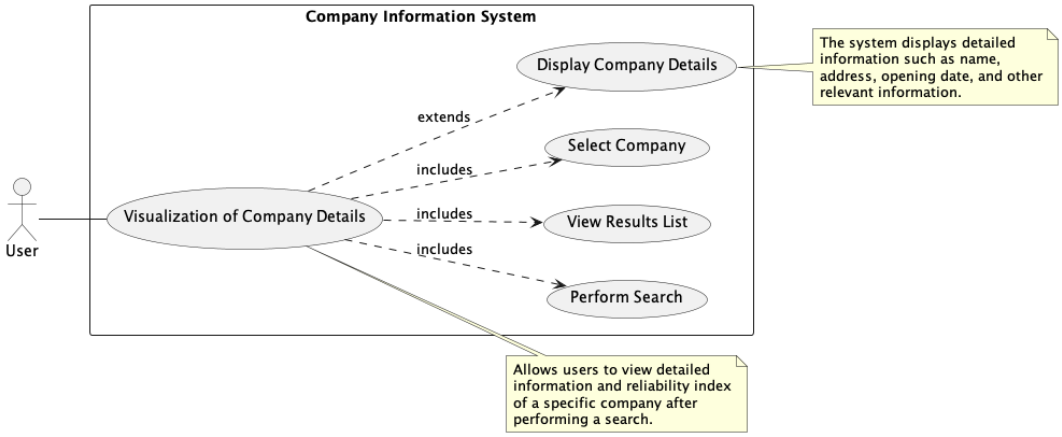

Figura 2: Visualização de detalhes da empresa.

Exportar Página

- **Atores:** Usuário
- **Premissas:** O sistema exibe informações detalhadas sobre a empresa em uma página formatada.
- **Fluxo de Eventos:**
    1. Com base nos dados coletados, o sistema realiza uma análise detalhada, utilizando algoritmos ou métodos específicos.
    2. O sistema calcula um índice ou grau de confiabilidade com base nos resultados da análise de dados. Esse índice pode ser representado numericamente ou em uma escala, indicando o nível de confiança na empresa.
    3. Com o índice de confiabilidade calculado, o sistema atribui um grau específico à empresa, classificando-a, por exemplo, como "Alta", "Média"ou "Baixa"confiabilidade.
    4. O grau de confiabilidade calculado é exibido junto aos demais detalhes da empresa.
- **Frequência de Uso:** Moderada
- **Pre-condições:** O usuário está na página de informações detalhadas de uma empresa.
- **Pós-condições:** O usuário obtém um arquivo exportado com as informações da empresa.
- **Problemas:** Possíveis problemas de formatação no arquivo exportado, dependendo do formato escolhido; recursos de exportação podem não estar disponíveis para todas as empresas.

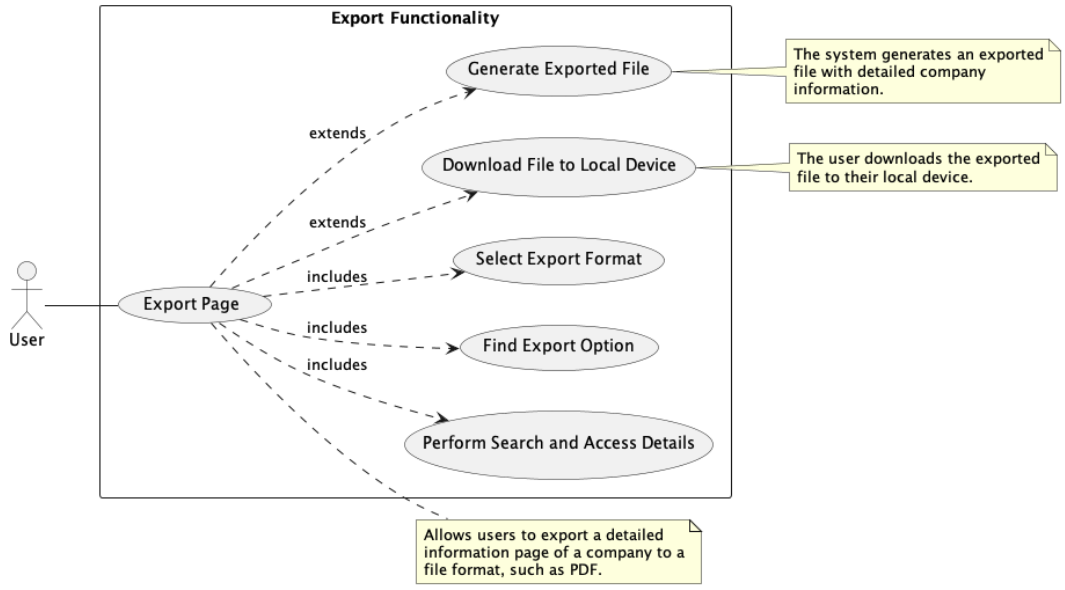

Figura 3: Exportar Página.

Comparar Empresas

- **Descrição:** Permite que os usuários comparem informações entre duas empresas para facilitar a tomada de decisões.
- **Atores:** Usuario
- **Premissas:** O sistema possui informações detalhadas sobre as empresas disponíveis para comparação.
- **Fluxo de Eventos:**
    1. O usuário realiza uma pesquisa utilizando critérios como CNPJ básico e razão social.
    2. O sistema exibe uma lista de resultados correspondentes à pesquisa.

3. O usuário clica em uma empresa específica na lista para visualizar mais detalhes.
4. O usuário seleciona a opção de ”Comparar com outra empresa”.
5. O usuário seleciona a segunda empresa.
6. O sistema apresenta as informações das empresas lado a lado.

- **Frequência de Uso:** Moderada
- **Pre-condições:** O usuário realizou uma pesquisa e selecionou duas empresas para comparação.
- **Pós-condições:** O usuário obtém uma visão comparativa das informações das empresas selecionadas.
- **Problemas:** Necessidade de garantir que as métricas comparadas sejam relevantes e comparáveis entre as empresas; possíveis desafios na apresentação visual dos dados.

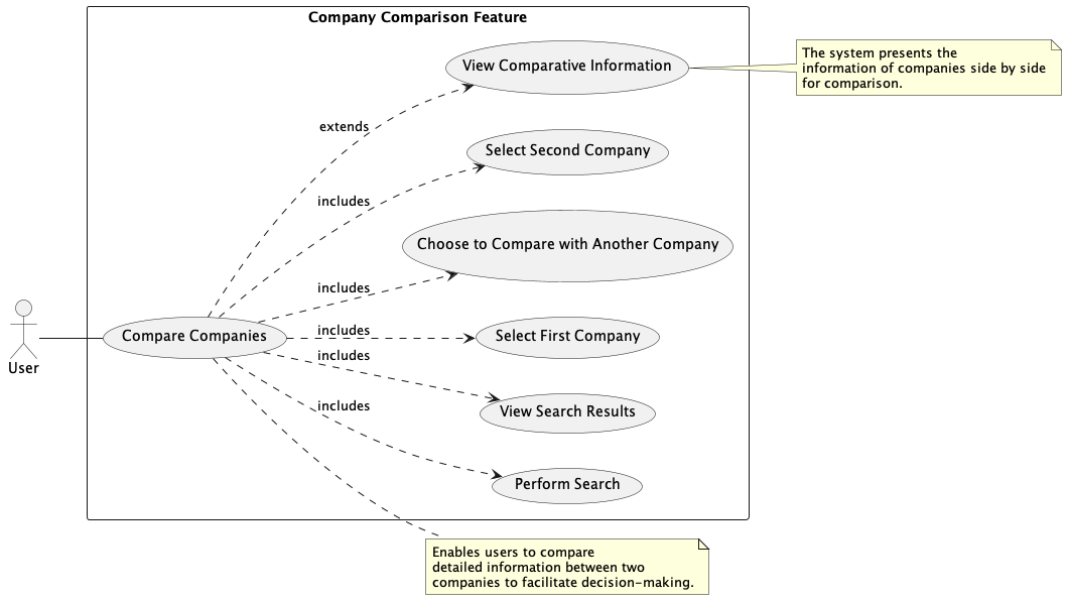

Figura 4: Comparar Empresas.

Gerar um grau de confiabilidade

- **Descrição:** Gerar um grau de confiabilidade para cada empresa com suporte nos dados disponíveis na base de dados.
- **Atores:** Sistema.
- **Premissas:** O sistema possui dados minimamente suficientes para calcular o índice de confiabilidade.
- **Fluxo de Eventos:**

1. Com base nos dados coletados, o sistema realiza uma análise detalhada, utilizando algoritmos ou métodos específicos.
2. O sistema calcula um índice ou grau de confiabilidade com base nos resultados da análise de dados. Esse índice pode ser representado numericamente ou em uma escala, indicando o nível de confiança na empresa.
3. Com o índice de confiabilidade calculado, o sistema atribui um grau específico à empresa, classificando-a, por exemplo, como ”Alta”, ”Média”ou ”Baixa”confiabilidade.
4. O grau de confiabilidade calculado é exibido junto aos demais detalhes da empresa.

- **Frequência de Uso:** Alta.

- **Pre-condições:** Os algoritmos ou métodos utilizados na análise de dados estão implementados e são capazes de processar as informações disponíveis.
- **Pós-condições:** Um grau de confiabilidade é atribuído a cada empresa analisada.
- **Problemas:** Possíveis indisponibilidades de dados históricos para algumas empresas; cálculos podem variar dependendo da disponibilidade e precisão dos dados históricos.

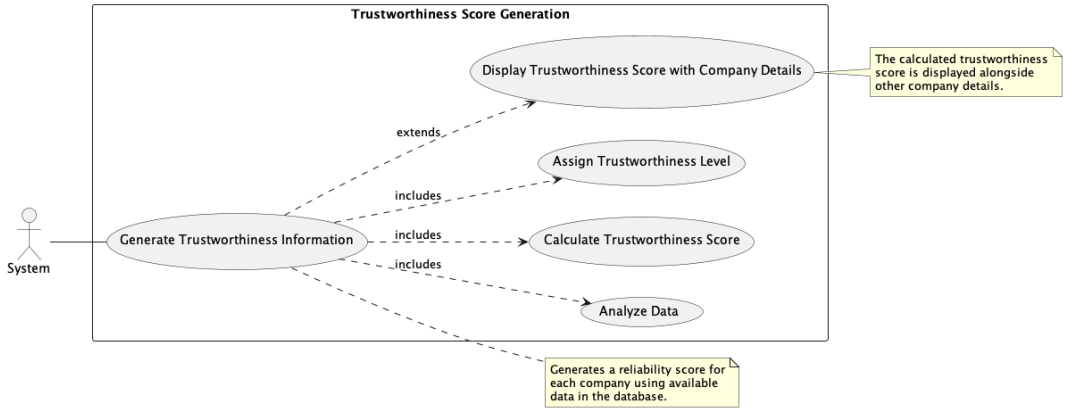

Figura 5: Gerar um grau de confiabilidade.

Observações e gerenciamento de dados e metadados dos modelos

- **Descrição:** Observações e gerenciamento de dados e metadados dos modelos.
- **Atores:** Administrador do Sistema.
- **Premissas:** O administrador possui credenciais de acesso ao sistema.
- **Fluxo de Eventos:**
  1. O Administrador do Sistema faz login na Plataforma de Metadados e Gerenciamento Interno.
  2. Na interface da plataforma, o administrador acessa os modelos disponíveis.
  3. O administrador pode visualizar informações relevantes sobre os dados armazenados.
  4. A plataforma permite que o administrador modifique ou adicione informações aos dados e metadados.
  5. O administrador salva quaisquer mudanças feitas.
- **Frequência de Uso:** Baixa.
- **Pre-condições:** Uma plataforma para análise, abstração de ideias e gerenciamento.
- **Pós-condições:** As alterações feitas pelo administrador refletem no sistema.
- **Problemas:** Possíveis desafios técnicos na escolha da tecnologia de indexação e otimização do desempenho; necessidade de atualizações periódicas à medida que o volume de dados aumenta.

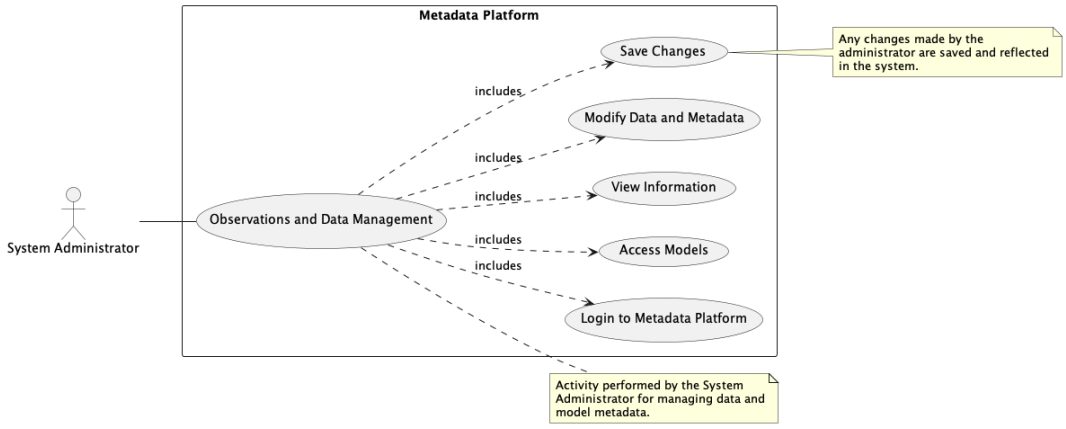

Figura 6: Observações e gerenciamento de dados e metadados dos modelos.

Realizar Backup de Dados Periodicamente

- **Descrição:** Permite que os administradores de sistema realizem backups regulares dos dados do sistema para garantir a segurança e a recuperação de dados em caso de falhas.
- **Atores:** Administrador de Sistema (ou sistema, se for automático).
- **Premissas:** O administrador possui permissões adequadas para realizar backups; o sistema possui funcionalidades de backup.
- **Fluxo de Eventos:**
  1. O Administrador de Sistema acessa a funcionalidade de backup no sistema.
  2. O Administrador inicia o processo de backup, selecionando opções como a periodicidade dos backups e o local de armazenamento.
  3. O sistema realiza uma cópia dos dados do sistema para um local seguro.
- **Frequência de Uso:** Mensal.
- **Pre-condições:** O administrador tem permissões de backup; o sistema está operacional.
- **Pós-condições:** Um novo backup dos dados é criado e armazenado em um local seguro.
- **Problemas:** Possíveis falhas no processo de backup devido a problemas de conectividade, falta de espaço de armazenamento ou outros problemas técnicos.

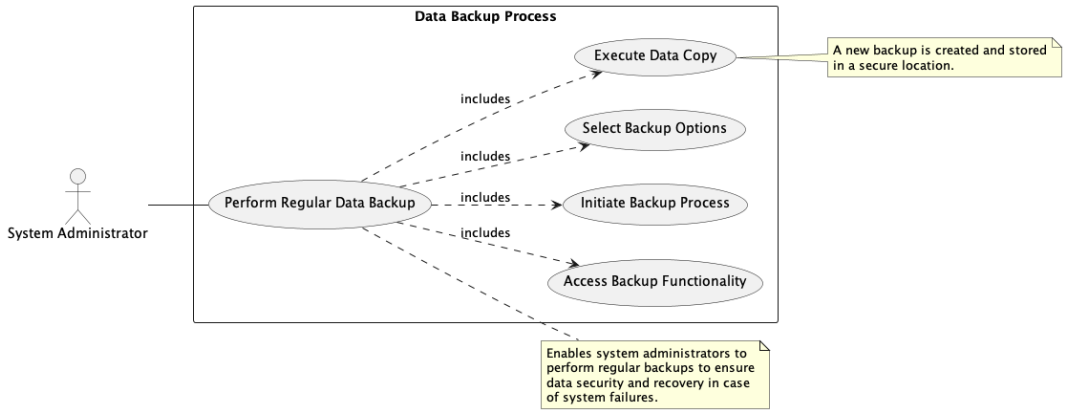

Figura 7: Realizar Backup de Dados Periodicamente.

Expansão da Base de Dados com o Econodata

- **Descrição:** Scrapping em tempo real para suprir a quantidade de dados disponível na base de dados do projeto.
- **Atores:** Desenvolvedor de Sistema.
- **Premissas:** O Econodata possui dados já tratados e organizados com suporte nas bases do governo e outras fontes.
- **Fluxo de Eventos:**
  1. O Desenvolvedor implementa um scraper capaz de realizar consultas em tempo real à base de dados do Econodata.
  2. O scraper é testado em ambiente controlado para garantir que está recuperando os dados corretamente.
  3. Os dados extraídos pelo scraper são integrados à plataforma existente no projeto.
  4. Os dados são exibidos complementando os dados originais.
- **Frequência de Uso:** Única.
- **Pre-condições:** A base de dados do projeto possui brechas que podem ser supridas com novos dados.
- **Pós-condições:** O sistema está documentado, incluindo detalhes sobre a integração com o Econodata e o scraper desenvolvido.
- **Problemas:** Possíveis interrupções temporárias durante a atualização ou falta de disponibilidade de dados atualizados.

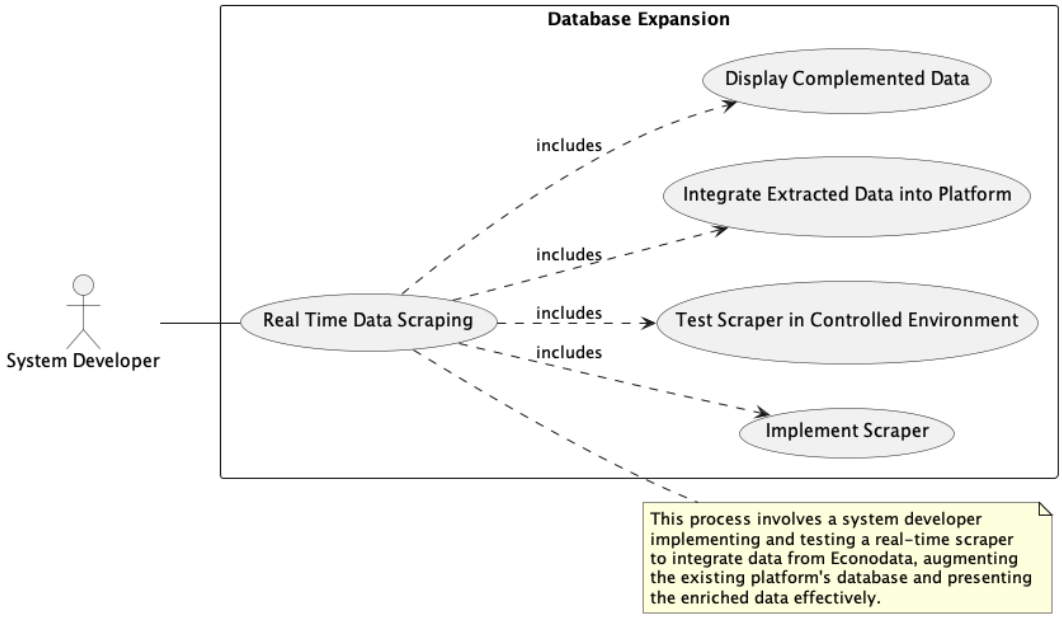

Figura 8: Expansão da Base de Dados com o Econodata.

Histórico de Pesquisa

- **Descrição:** Permite que o sistema mantenha um histórico das consultas de pesquisa durante a sessão do usuário.
- **Atores:** Sistema.
- **Premissas:** O usuário está em uma sessão ativa no sistema.
- **Fluxo de Eventos:**

1. Durante a sessão do usuário, o sistema registra automaticamente as consultas de pesquisa realizadas.
2. O sistema armazena as consultas de pesquisa em uma estrutura de dados dedicada associada à sessão do usuário.
3. O usuário pode visualizar seu histórico de pesquisa através da interface do usuário durante a sessão ativa.

- **Frequência de Uso:** Alta.
- **Pre-condições:** O usuário está em uma sessão ativa no sistema.
- **Pós-condições:** Os usuários podem revisitar consultas de pesquisa sem precisar digitá-las novamente.
- **Problemas:** Possíveis desafios na implementação técnica para armazenar e acessar os dados; questões de privacidade e regulamentações devem ser consideradas e comunicadas aos usuários.

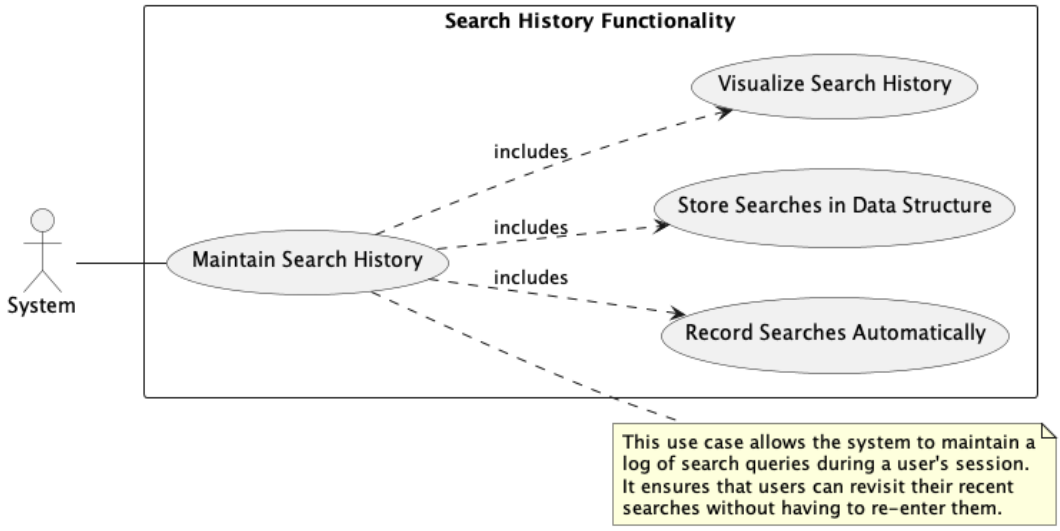

Figura 9: Histórico de Pesquisa

Dockerização e Implantação.

- **Descrição:** Este caso de uso descreve o processo de empacotamento do aplicativo em contêineres Docker e sua implantação em servidores para garantir uma implantação fácil, consistente e escalável do sistema.
- **Atores:** Administrador de Sistema, Sistema.
- **Premissas:** O aplicativo e todos os seus componentes estão configurados e prontos para implantação.
- **Fluxo de Eventos:**
  1. O administrador de sistema inicia o processo de dockerização.
  2. O sistema empacota o aplicativo, suas dependências e configurações em contêineres Docker, garantindo a portabilidade e isolamento.
  3. Os contêineres Docker são construídos localmente ou em um ambiente de compilação.
  4. Após a construção bem-sucedida, os contêineres são armazenados em um registro de contêineres, como o Docker Hub ou um registro privado.
  5. O sistema implanta os contêineres em servidores, como instâncias da AWS, usando uma orquestração de contêineres, como o Docker Compose ou Kubernetes.
  6. O sistema inicia o aplicativo nos contêineres, tornando-o disponível para os usuários.
- **Frequência de Uso:** O processo de dockerização é executado quando uma nova versão do sistema é lançada ou quando as configurações do ambiente de implantação mudam. A implantação pode ser contínua ou agendada, dependendo das necessidades do sistema.

- **Pre-condições:** O aplicativo e todos os seus componentes estão configurados e prontos para serem empacotados em contêineres enquanto a infraestrutura de implantação, como servidores e ambiente de orquestração, está configurada e pronta para receber os contêineres.
- **Pós-condições:** O sistema está implantado e em execução em contêineres Docker.
- **Problemas:** Possíveis problemas incluem erros durante o processo de dockerização, falhas na construção de contêineres, conflitos de versões de dependências e problemas de implantação que podem afetar a disponibilidade do sistema. É importante documentar e testar o processo de dockerização e implantação para minimizar esses problemas.

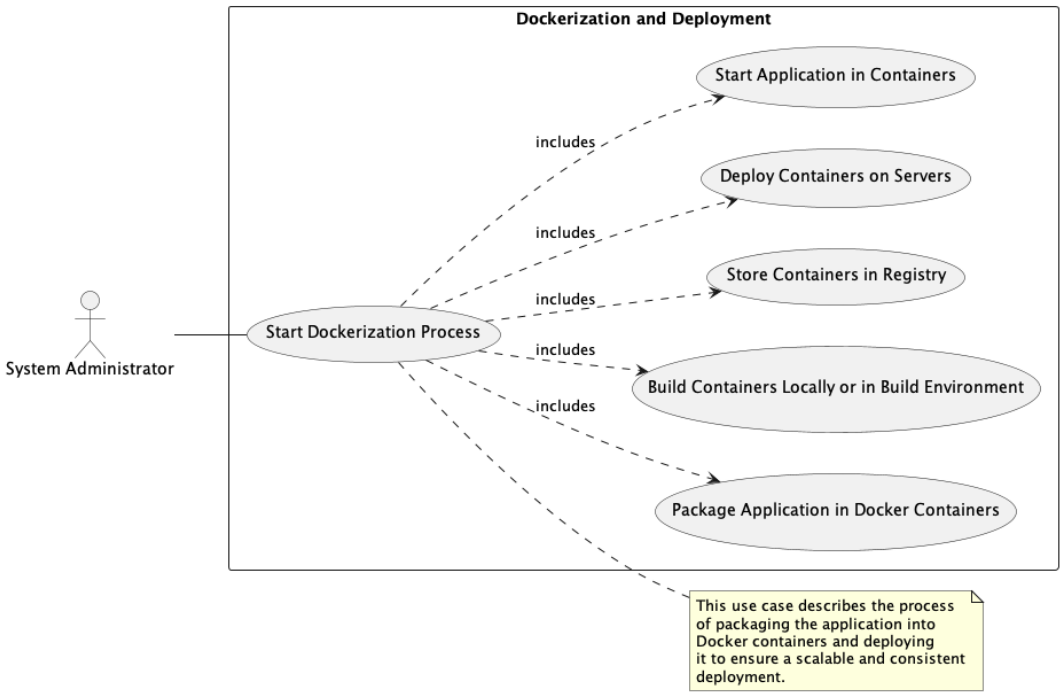

Figura 10: Dockerização e Implantação.

2.3 Diagrama de classes

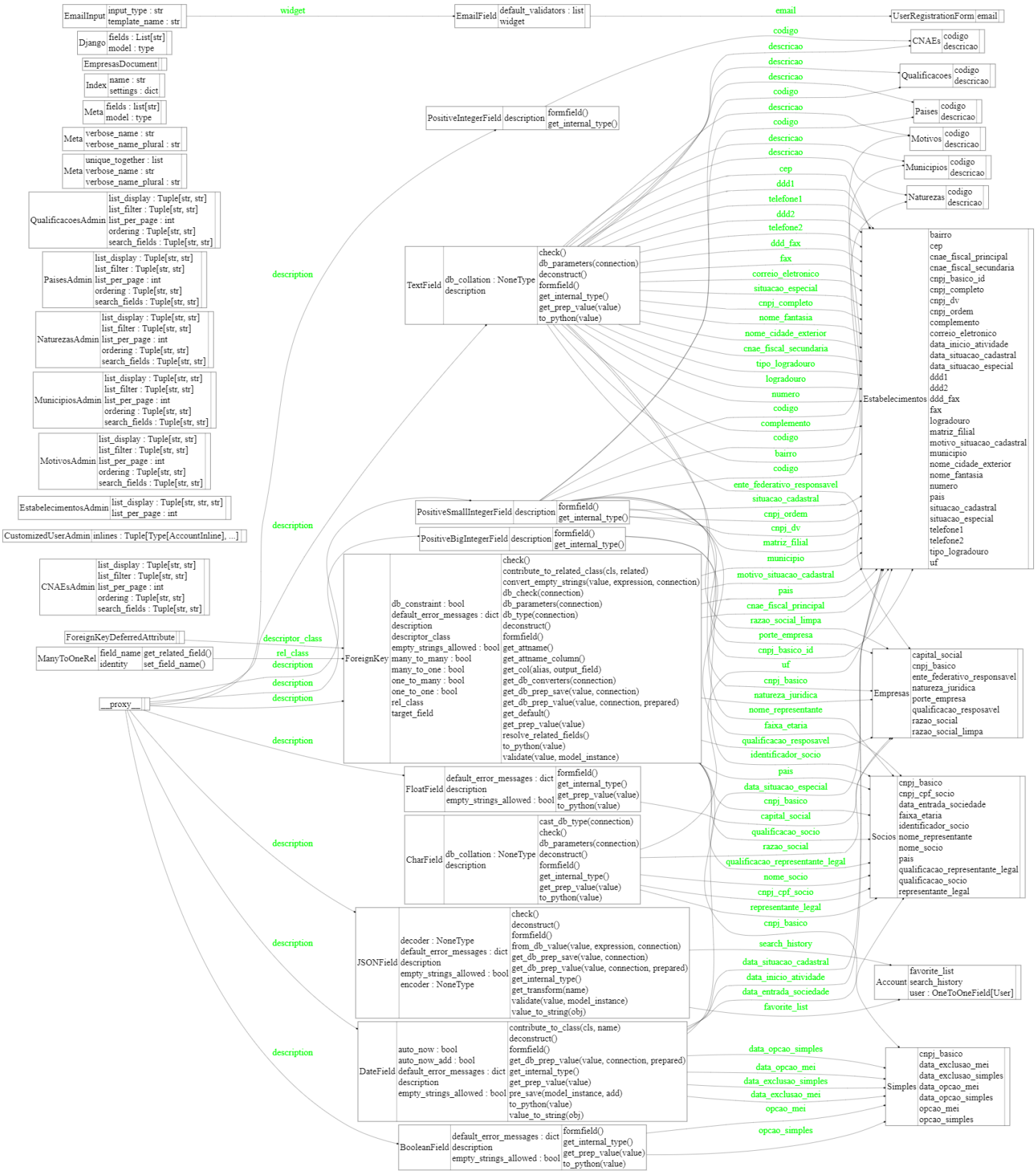

Figura 11: Diagrama de classes

2.4 Diagrama de componentes

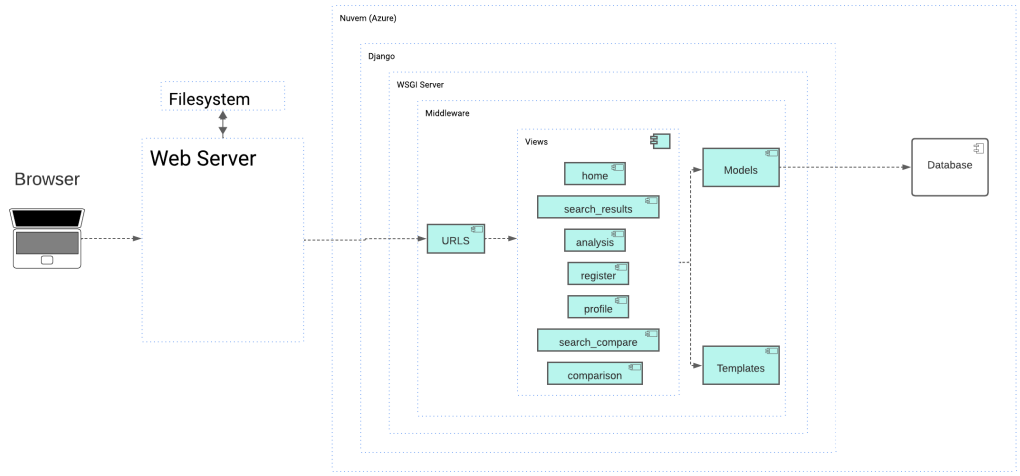

Figura 12: Diagrama de componentes

2.5 Diagrama de pacote

Este diagrama tem o propósito de representar a organização de pacotes e/ou módulos do projeto.

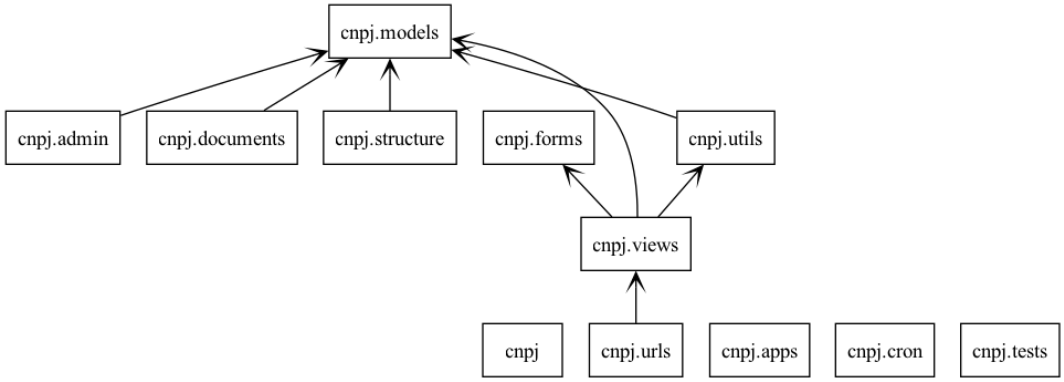

Figura 13: Diagrama de pacote

cnpj.models

Os modelos refletem a estrutura de dados do sistema, com foco em informações empresariais e cadastrais.

- `Paises` - Armazena informações sobre países.
- `Municipios` - Modelo para municípios brasileiros. Campos: código, descrição.
- `Qualificacoes` - Armazena qualificações de sócios ou responsáveis. Campos: código, descrição.
- `Naturezas` - Representa as naturezas jurídicas das empresas. Campos: código, descrição.
- `CNAEs` - Modelo para a Classificação Nacional de Atividades Econômicas. Campos: código, descrição.
- `Motivos` - Armazena motivos de situações cadastrais. Campos: código, descrição.
- `Empresas` - Modelo central para empresas, contendo informações básicas e jurídicas. Campos: CNPJ, razão social, natureza jurídica, entre outros.

- `Estabelecimentos` - Detalha informações sobre estabelecimentos de empresas. Campos incluem CNPJ, nome fantasia, endereço, entre outros.
- `Simples` - Relacionado ao regime tributário Simples Nacional, com detalhes de opção e exclusão. Campos: CNPJ, opção pelo Simples, datas de opção e exclusão, entre outros.
- `Socios` - Modelo para sócios das empresas, com informações pessoais e de participação. Campos: nome do sócio, qualificação, dados de representação legal, entre outros.
- `Account` - Modelo para contas de usuário, com armazenamento flexível em campos JSON.

Funções:

- Métodos `__str__` para modelos - Representações em string para instâncias de modelos.
- `create_user_account` - Função relacionada à criação de conta de usuário.

cnpj.admin

Administração do Django para os modelos. Estabelece como os modelos são representados e gerenciados na interface administrativa do Django.

Classes:

- `PaisesAdmin` - Administração do Django para modelos relacionados a países.
- `MunicipiosAdmin` - Administração do Django para modelos de municípios.
- `QualificacoesAdmin` - Administração do Django para modelos de qualificações.
- `NaturezasAdmin` - Administração do Django para modelos de naturezas.
- `CNAEsAdmin` - Administração do Django para modelos relacionados a CNAE.
- `EstabelecimentosAdmin` - Administração do Django para modelos de estabelecimentos.
- `MotivosAdmin` - Administração do Django para modelos de motivos.
- `AccountInline` - Representação inline de informações de contas.
- `CustomizedUserAdmin` - Classe de administração de usuários personalizada.

cnpj.documents

Gerenciamento de documentos relacionados às empresas, utilizando tecnologias de busca e indexação de dados.

Classes:

- `EmpresasDocument` - Modelo de documento para dados de empresas.

cnpj.structure

Contém funções auxiliares para conversão de tipos de dados e manipulação de modelos, especialmente em relação ao tratamento de chaves estrangeiras.

Funções:

- `integer(value: Any, _: bool = False) -> Optional[int]` - Converte valores para inteiro.

- `float_(value: Any, _: bool = False) -> Optional[float]` - Converte valores para float.
- `text(value: Any, _: bool = False) -> Optional[str]` - Converte valores para texto.
- `char(value: Any, _: bool = False) -> Optional[str]` - Converte valores para caracteres.
- `boolean(value: Any, _: bool = False) -> Optional[bool]` - Converte valores para booleano.
- `date(value: Any, _: bool = False) -> Optional[str]` - Converte valores para data.
- `foreign_key(model: Type[Model], key: str)` - Manipula chaves estrangeiras em modelos.

Comentários:

- Funções auxiliares para conversão de dados.
- Obter ou criar uma instância de modelo usando a chave fornecida.

cnpj.forms

Define os formulários usados na aplicação, principalmente para a interação do usuário com os dados do sistema.

Classes:

- `UserRegistrationForm` - Formulário para registro de usuários.

cnpj.utils

Inclui funções utilitárias para operações específicas como cálculos, extração de dados e avaliações.

Funções:

- `calcular_dv_cnpj(cnpj_base)` - Calcula dígitos verificadores do CNPJ.
- `calculate_dv_cnpj(cnpj_base: int) -> str` - Calcula dígitos verificadores do CNPJ.
- `econodata_scrapping(cnpj_base: int) -> dict` - Extrai dados da Econodata.
- `confiability_score(cnpj_base: int) -> str` - Calcula o índice de confiabilidade.

cnpj.views

Gerencia a apresentação de dados e interação com o usuário, controlando o que é exibido nas páginas da aplicação.

Funções:

- `home(request)` - Renderiza a página inicial.
- `search_results(request)` - Exibe resultados de pesquisa.
- `analysis(request, pk)` - Realiza análise de dados.
- `register(response)` - Registra usuários.
- `profile(response)` - Gerencia o perfil do usuário.

cnpj.urls

Define as rotas da aplicação, mapeando URLs para as respectivas views que devem ser chamadas.

2.6 Diagrama de sequência

Os diagramas de sequência abaixo tem por objetivo fornecer uma representação visual clara das interações e do fluxo de execução em um aplicativo, facilitando a compreensão de como o sistema funciona.

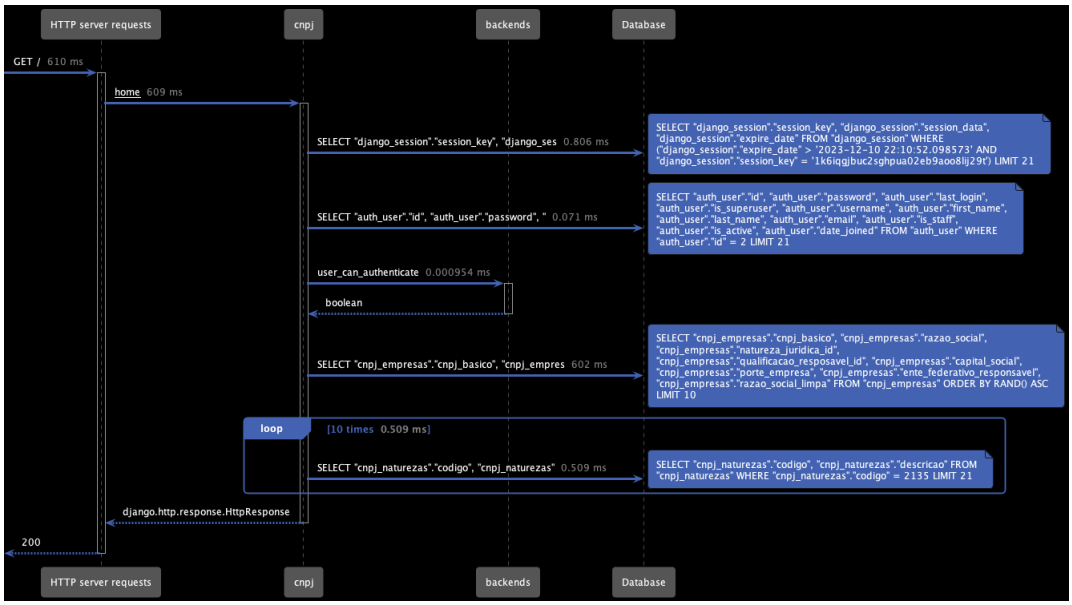

Figura 14: Diagrama de sequência da página home

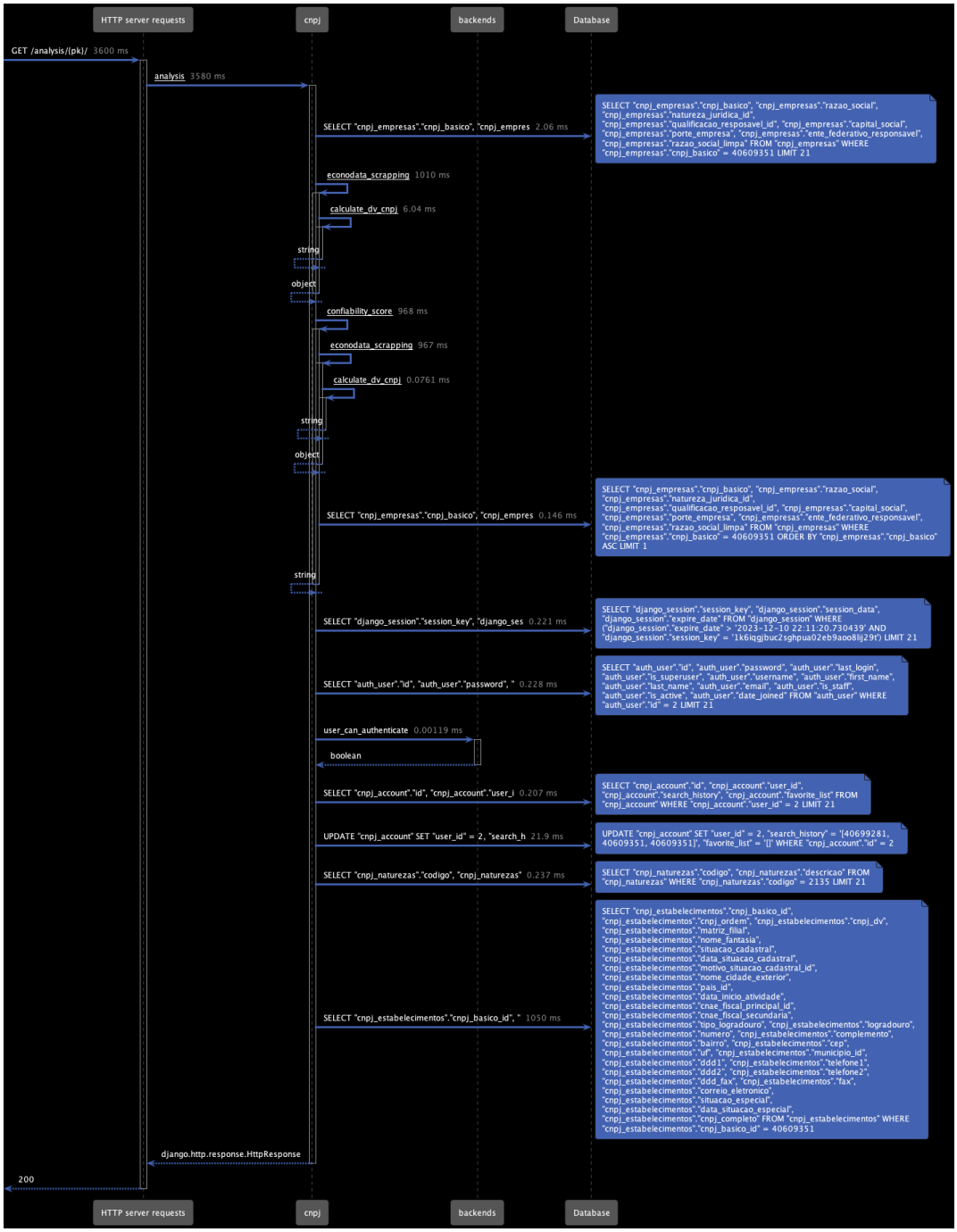

Figura 15: Diagrama de sequência da página análise

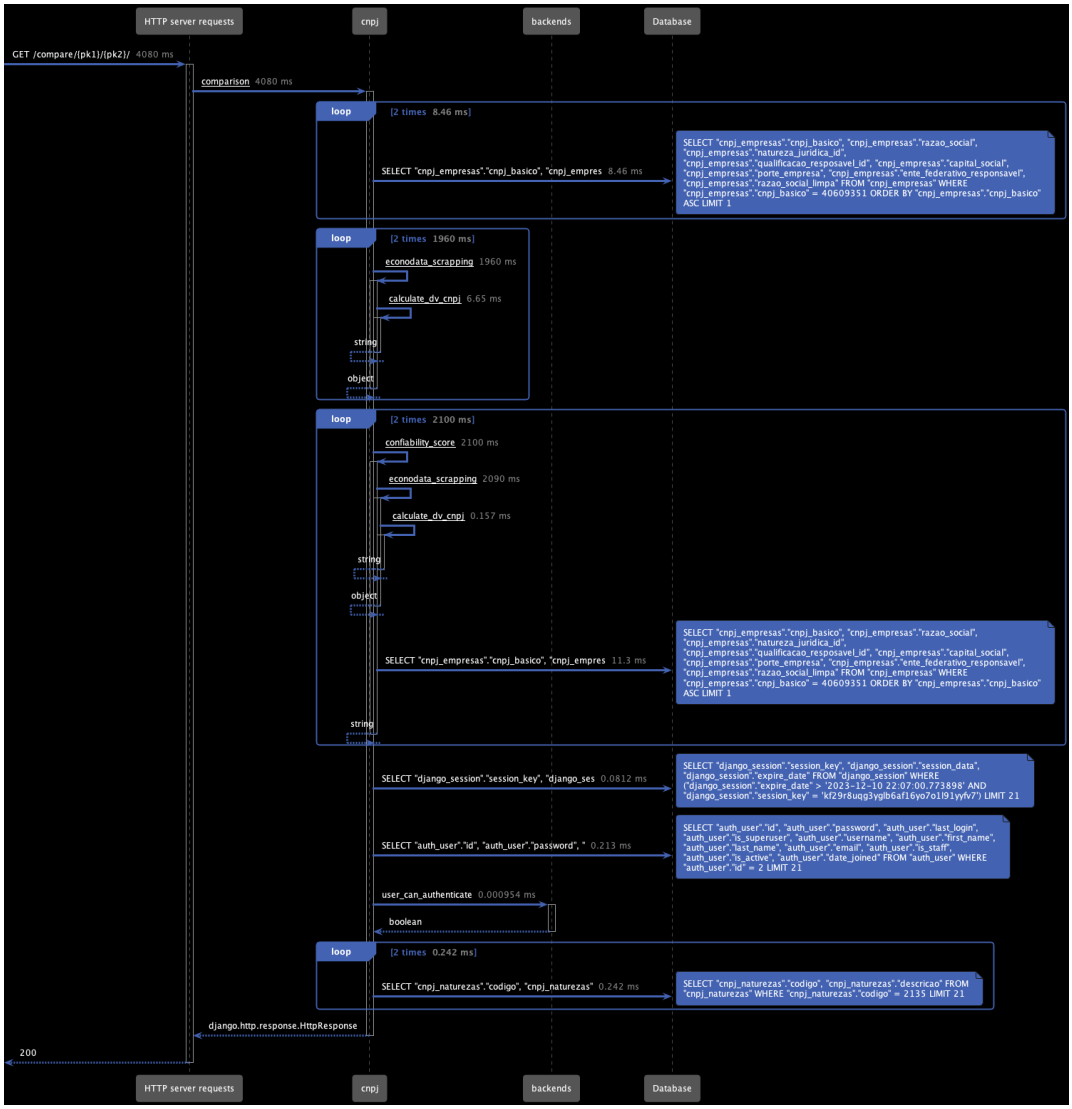

Figura 16: Diagrama de sequência da página comparação

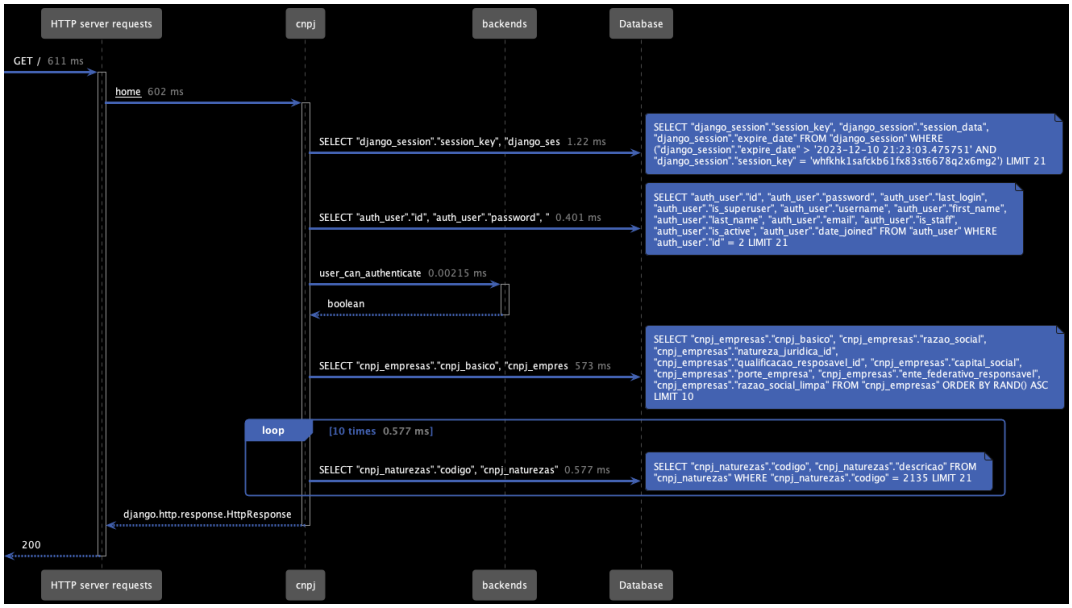

Figura 17: Diagrama de sequência de abrir uma página

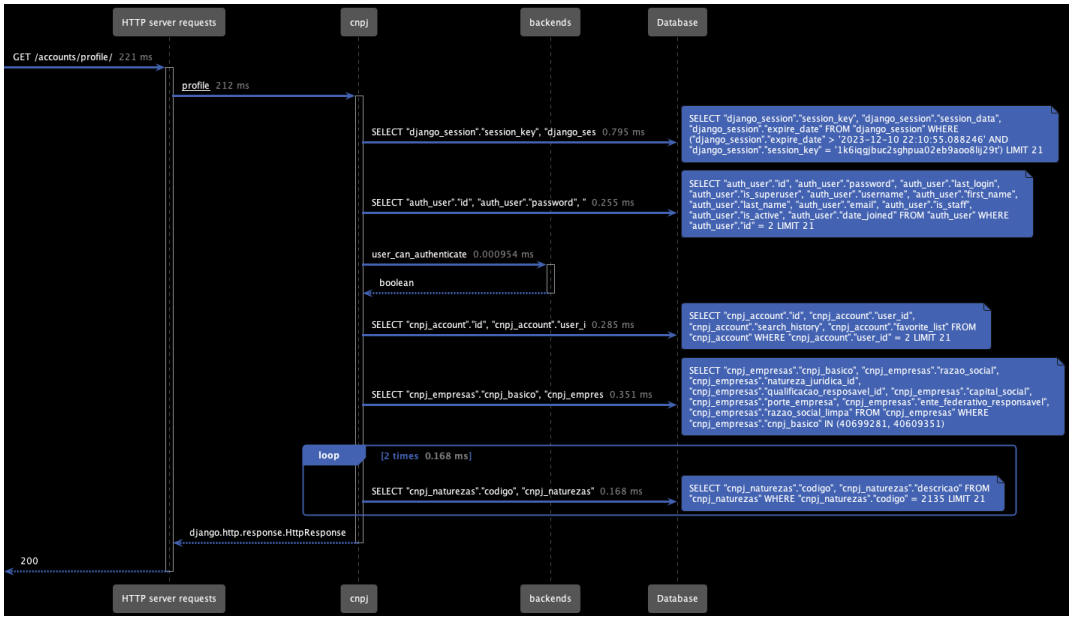

Figura 18: Diagrama de sequência da página perfil

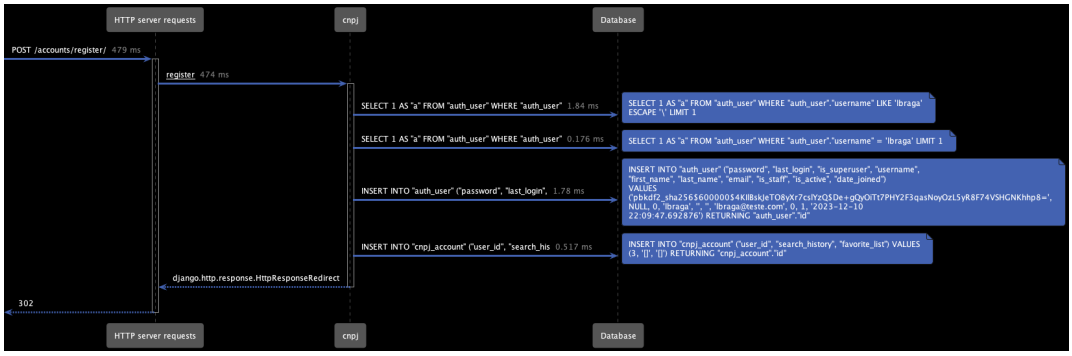

Figura 19: Diagrama de sequência do registro

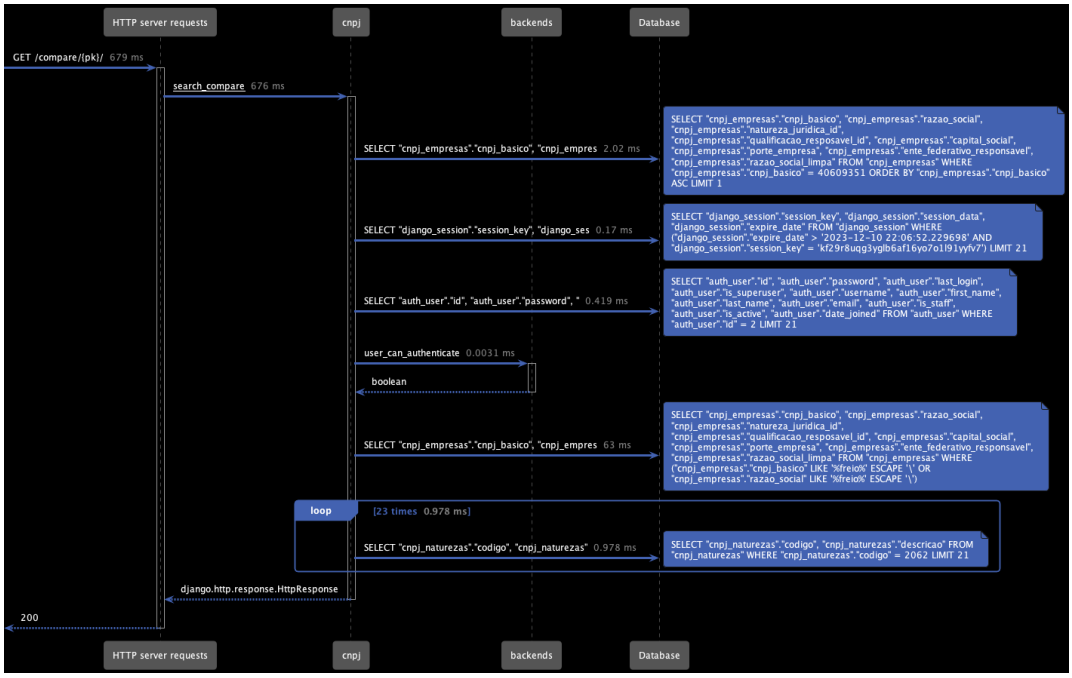

Figura 20: Diagrama de sequência da comparação de empresas

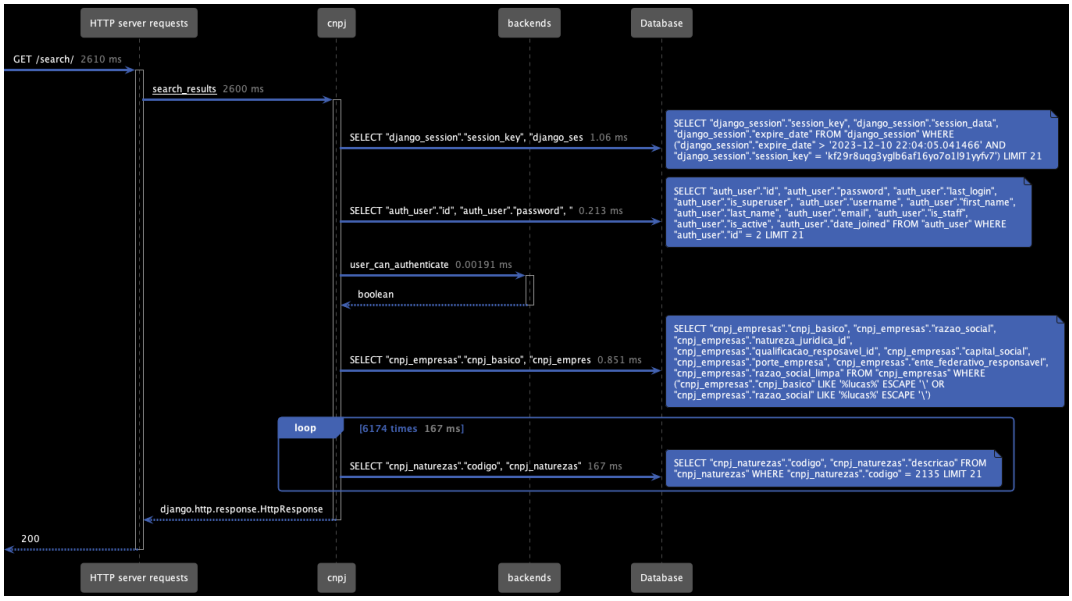

Figura 21: Diagrama de sequência da busca de empresas

3 Padrões de Projeto

3.1 Padrões de criação

Singleton em models.py

O Singleton é um padrão de design criacional que permite garantir que uma classe tenha apenas uma instância, proporcionando um ponto de acesso global a essa instância. Este padrão é particularmente útil quando apenas uma instância de um objeto é necessária para coordenar ações em todo o sistema.

3.2 Padrões estruturais

Decorator em admin.py

O padrão Decorator é implementado no arquivo `admin.py`, utilizado para registrar modelos no site de administração do Django. Este padrão é identificado pelo uso do decorador `@admin.register(Model)`, aplicado em classes que herdam de `admin.ModelAdmin`.

3.3 Padrões de comportamento

Meta Class em models.py e forms.py

O padrão Meta Class é observado nos arquivos `models.py` e `forms.py`. Nas classes de modelo, a classe Meta interna é usada para definir configurações do banco de dados e outras opções. Nas classes de formulário, ela especifica o modelo associado e os campos a serem incluídos, facilitando a configuração e entendimento do comportamento das classes.

Incremental em cnpj/urls.py

O padrão Iterator implementado no `cnpj/urls.py` onde se usa uma classe para adicionar ou modificar caminhos de URL ao longo do tempo, mantendo uma estrutura organizada e legível para a definição de URLs no seu projeto Django.

Template em models.py

O modelo Template esta sendo usado em `models.py`, onde a classe BaseModel serve como uma estrutura comum compartilhada por outros modelos, seguindo um padrão de modelagem comum em Python para evitar duplicação de código e promover a reutilização.

Strategy em cleaning.py

O modelo Strategy esta sendo usado em `cleaning.py`, onde as estratégias são encapsuladas em classes concretas, e o contexto (o migrador) é configurado para usar diferentes estratégias em momentos diferentes.

4 Coleta de dados

Os dados foram coletados através do site da Receita Federal, nesse endereço. Como haviam muitas tabelas e a conexão não era muito boa, decidimos por pegar uma tabela de cada assunto (empresas, estabelecimentos, etc.) e complementá-los com um webscrapping em tempo real com o econodata. Essa escolha se deu pela facilidade de fazer um webscrapping em página estática, ainda mais considerando que nós estamos fazendo isso em uma máquina na nuvem. Como comentaremos a seguir, por essa decisão, mantemos dois bancos de dados: um PostgreSQL e um SQLite, pois nossos créditos não eram suficiente para popular o PostgreSQL com todos os dados que tinhamos tratado e modelado no SQLite.

5 Como executar

O projeto está hospedado em uma máquina na Microsoft Azure e pode ser acessado por: `http://20.195.169.122:8000/`. Sugerimos que usufrua ao máximo da plataforma. Caso queira utilizar uma conta de administrador, deixamos uma com login: adame e senha: batata. Mas você pode criar contas normais para criar seu próprio histórico de busca.

Para executar o projeto localmente, é necessário ter instalado o `docker` e o `docker-compose`. Além disso, você deve baixar o banco de dados SQLite neste endereço e deixá-lo na pasta raiz. Apesar de termos configurado e hospedado um PostgreSQL, como gostaríamos de ter a maior quantidade de empresas em nosso projeto (500k +), inviabilizava hospedar um banco deste tamanho com os créditos disponibilizados pela FGV. Ainda assim, o projeto hospedado na nuvem está com o banco persitente instanciado por docker, como pedido pelo requisito extra. As imagens estão presentes no repositório.

Para executar o projeto, existem duas formas. Uma de desenvolvimento e uma de production. As variáveis de ambiente definidas no repositório são apenas para testes e não há perigo em serem públicas.

A versão de production basta executar o comando `docker-compose -f docker-compose.prod.yml up --build` e se torna acessível em `https://localhost:1337`.

A versão de desenvolvimento basta executar o comando `docker-compose -f docker-compose.yml up --build` e se torna acessível em `https://localhost:8000`.

Note que é recomendado manter apenas um dos contêineres levantados. Para derrubá-los, basta: `docker-compose down -v`

6 Cobertura de testes

É possível conferir detalhadamente um relatório de cobertura, usando o módulo `coverage` em: `https://github.com/wllsena/cnpj_insight/blob/main/coverage_report.pdf`, entretanto, o resumo impresso no terminal é:

```
    Name                          Stmts   Miss  Cover   Missing
-------------------------------------------------------------
cnpj/__init__.py                      0      0   100%
cnpj/admin.py                        60      0   100%
cnpj/apps.py                          4      0   100%
cnpj/cron.py                          6      6     0%   1-9
cnpj/documents.py                    15      0   100%
cnpj/forms.py                         8      0   100%
cnpj/models.py                      124     12    90%   36, 45, 54, 63, 74, 83, 104, 149, 164, 203, 230-231
cnpj/structure.py                    41     41     0%   1-142
cnpj/tests/__init__.py                0      0   100%
cnpj/tests/test_forms.py              0      0   100%
cnpj/tests/test_models.py             0      0   100%
cnpj/tests/test_urls.py              25      0   100%
cnpj/tests/test_views.py             69      0   100%
cnpj/urls.py                          3      0   100%
```

```
cnpj/utils.py                  83     71    14%   61-90, 117-197, 212-286
cnpj/views.py                  72     23    68%   124-147, 173-180, 216-222, 282-291, 337-349
cnpj_insight/__init__.py        0      0   100%
cnpj_insight/asgi.py            9      9     0%   10-34
cnpj_insight/settings.py       29      0   100%
cnpj_insight/urls.py            4      0   100%
cnpj_insight/wsgi.py            4      4     0%   10-14
manage.py                      12      2    83%   12-13
media/__init__.py               0      0   100%
static/__init__.py              0      0   100%
staticfiles/__init__.py         0      0   100%
templates/__init__.py           0      0   100%
-------------------------------------------------------------
TOTAL                         568    168    70%
```

O que é uma boa cobertura, ao considerar quais módulos são essenciais para o funcionamento da aplicação e os que não são.

Caso quiser executar os testes, basta: `docker-compose -f docker-compose.yml run --rm web python manage.py test`.

7 Documentação

A documentação está disponível em `https://wllsena.github.io/cnpj_insight/`, foi gerada utilizando `mkdocs` e está hospedado utilizando o `github-pages`.