



Documentação

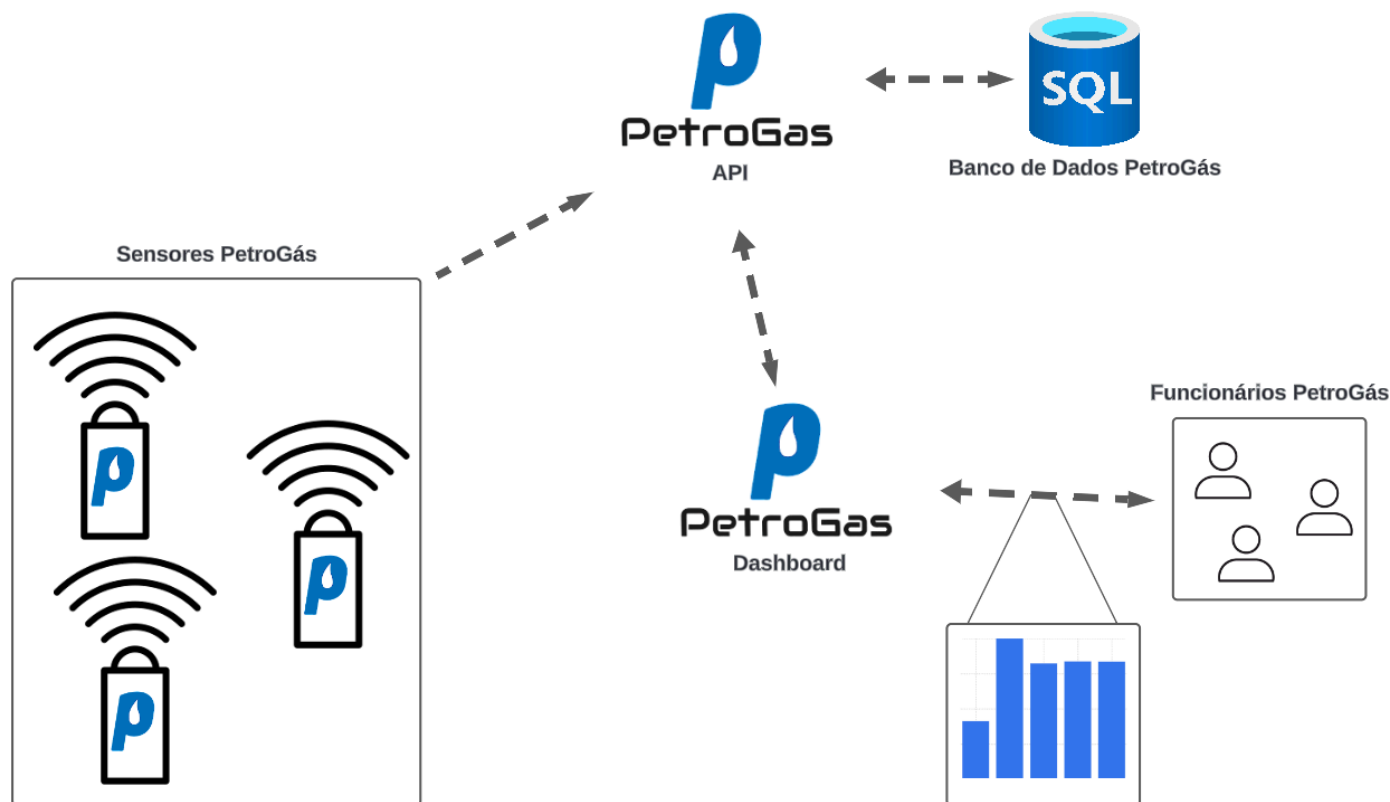
Descrição do Projeto

Sumário

- Sumário..... 2**
- Descrição do Projeto.....3**
- Tecnologias utilizadas.....4**
 - Servidor (API):..... 4
 - Dashboard Web:.....4
 - Banco de Dados:..... 5
 - Geral:..... 5
- Instruções de uso..... 6**
- Primeira Execução.....7**
- Artefatos de Documentação..... 8**
- Testes Unitários - BackEnd..... 8**
- Testes de Integração - FrontEnd..... 11**

Descrição do Projeto

A empresa **PetroGás** é uma empresa **FICTÍCIA** sem nenhum tipo de filiação ou associação com **empresas reais**, que atua no setor de óleo e gás. O projeto desenvolvido e documentado neste artefato é uma plataforma para o controle, registro e visualização dos dados obtidos através de sensores da Petrogás, fornecendo também o controle de acesso à plataforma.



A aplicação foi desenvolvida pelo desenvolvedor João Batista de Freitas Filho ([LinkedIn](#) | [GitHub](#)). Toda a aplicação foi desenvolvida em inglês, visando a utilização dessa plataforma internacionalmente.

A aplicação foi desenvolvida pensando em fornecer uma solução simples, porém elegante e escalável para suprir as necessidades da empresa, organizando os dados dos seus mais de 2000 sensores, funcionando 24h diariamente espalhados no Brasil e no mundo, e fornecendo seus dados em visualizações inteligentes e interativas para fornecer insights de milhares de dados com baixo esforço.

O sistema **Petrogás** descrito neste documento consiste em:

- Um **Servidor** com **API** para controle dos dados e requisições
- Um **Dashboard Web**, com visualização de dados e controle de acesso
- Um **Banco de Dados** relacional, responsável por armazenar os dados das leituras dos sensores, além dos dados dos usuários do dashboard.

Tecnologias utilizadas

O sistema Petrogás foi desenvolvido utilizando as seguintes tecnologias:

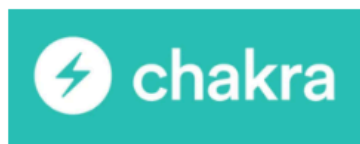
Servidor (API):

- **Python** como linguagem de implementação
- **FastAPI** para o desenvolvimento e gerenciamento da API
- **SQLModel** e **Pydantic** para as interações SQL com o banco de dados
- **Pytest** para testes unitários
- **Swagger** para documentação de API



Dashboard Web:

- **HTML**, **CSS** e **Typescript** como linguagem de implementação
- **ReactJs** para a composição dos componentes web
 - **Vite** para gerenciar e simplificar o desenvolvimento e deploy
 - **Chakra** para os componentes visuais
 - **Playwright** para testes de integração



Banco de Dados:

- **PostgreSQL** como o banco de dados relacional
- **Alembic** para gerenciamento e controle de migração



Geral:

- **Docker** para containerização do ambiente



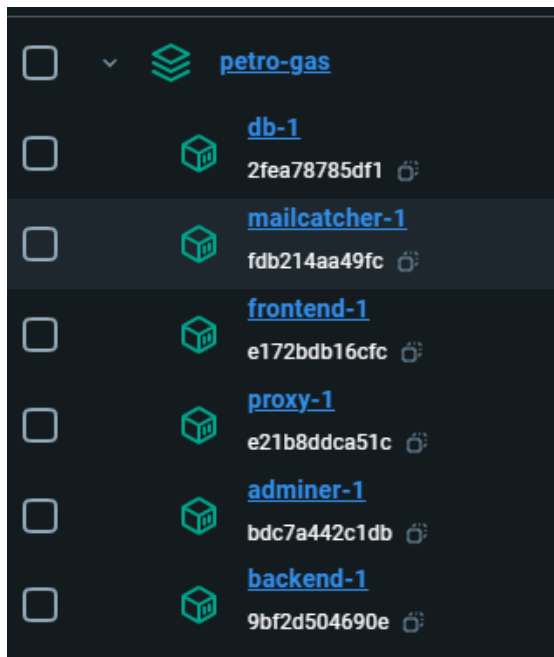
Instruções de uso

A aplicação está completamente containerizada em docker, para uma fácil instalação e deploy. Para executá-la, basta seguir as instruções:

- Clone o projeto no [Github](#) e coloque-o em uma pasta na sua máquina
- Altere todas as ocorrências dos textos abaixo para os dados do sistema que irá rodar na sua m
 - **INSERT_SECRET_KEY_HERE**: Secret Key para criptografia
 - **INSERT_SUPERUSER_PASSWORD_HERE**: Senha do administrador padrão
 - **INSERT_POSTGRES_PASSWORD_HERE**: Senha do PostgreSQL
- Instale o [Docker Desktop](#) e inicie-o
- Usando um console (powershell se estiver no Windows) execute os comandos:
 - ***docker-compose build***
 - ***docker compose up -d***

OBS: Talvez você precise instalar e baixar algumas imagens Docker, e outras tecnologias necessárias (Node, Npm, PostgreSQL, etc.). Fique atento à saída dos comandos e siga as instruções que aparecerem.

- Uma vez executado os comandos, você poderá acessar os contêineres dentro do Docker Desktop



- Você pode então acessar a documentação via Swagger em <http://localhost/docs#/>
- A aplicação poderá ser acessada em <http://localhost/login>
- Você pode usar uma aplicação como o [DBeaver](#) para acessar o banco de dados.

OBS: Para fazer o deploy da aplicação, basta seguir o processo, alterando os campos de domínio da aplicação e subir a aplicação containerizada.

Primeira Execução

Na primeira execução do projeto, o sistema irá realizar algumas ações para a devida utilização do sistema, sendo elas:

- Inicialização do banco de dados, juntamente com a aplicação das migrações
 - Todas as modificações no banco de dados deverão ser feitas usando o [Alembic](#)
 - Na criação do contêiner, o banco é criado e todas as suas migrações são devidamente aplicadas, criando todas as tabelas do banco de dados, juntamente com os Stored Procedures, Functions, Views, Constraints, Triggers, etc.
- Criação do super usuário padrão
 - **Email:** admin@petrogas.com
 - **Senha:** Mesma senha configurada no arquivo `.env` do projeto (o valor inicial é ***INSERT_SUPERUSER_PASSWORD_HERE***)

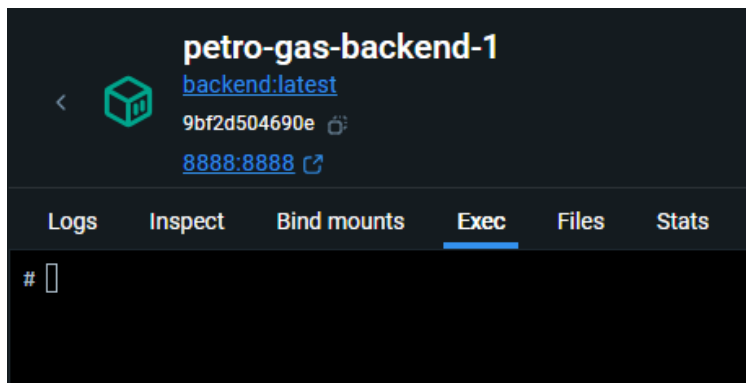
Artefatos de Documentação

Você pode encontrar este, e todos os outros artefatos de documentação para esse projeto dentro da pasta **docs** do repositório.

Testes Unitários - BackEnd

O servidor da aplicação possui testes unitários escritos usando **PyTest**. Para rodar os testes, basta seguir os passos:

- Inicie a aplicação (veja o tutorial anterior)
- Localize o contêiner **petro-gas-backend** no Docker Desktop (também pode aparecer como petro-gas-backend-1)
- Acesse contêiner, e vá para a aba **Exec**



- Execute o comando **bash /app/tests-start.sh**
- Os testes unitários serão executados, você poderá ver os resultados uma vez que eles tenham finalizado

```
+ bash ./scripts/test.sh
+ coverage run --source=app -m pytest
===== test session starts =====
platform linux -- Python 3.10.14, pytest-7.4.4, pluggy-1.5.0
rootdir: /app
plugins: anyio-4.4.0
collected 54 items

app/tests/api/routes/test_login.py .....
app/tests/api/routes/test_sensor_data.py .....
app/tests/api/routes/test_users.py .....
app/tests/crud/test_user.py .....
app/tests/scripts/test_backend_pre_start.py .
app/tests/scripts/test_test_pre_start.py .
```


- Você também verá a cobertura (“**coverage**”) de todos os arquivos relevantes do backend em relação aos testes. Isso pode e deve ser tomado como base para melhorar a qualidade do sistema.

```
===== 54 passed, 2 warnings in 21.64s =====
```

```
+ coverage report --show-missing
```

Name	Stmts	Miss	Cover	Missing
app/__init__.py	0	0	100%	
app/api/__init__.py	0	0	100%	
app/api/deps.py	34	4	88%	39-40, 47, 52
app/api/main.py	7	0	100%	
app/api/routes/__init__.py	0	0	100%	
app/api/routes/auth.py	52	7	87%	107, 127-139
app/api/routes/sensor_data.py	107	40	63%	111-127, 140-198, 212, 237-272
app/api/routes/users.py	98	0	100%	
app/api/routes/utils.py	11	3	73%	20-26
app/backend_pre_start.py	23	7	70%	25-27, 31-33, 37
app/core/__init__.py	0	0	100%	
app/core/config.py	72	8	89%	21-23, 43, 100-107
app/core/db.py	10	2	80%	18-23
app/core/security.py	16	0	100%	
app/crud.py	47	0	100%	
app/initial_data.py	14	14	0%	1-23
app/main.py	14	1	93%	17
app/models.py	122	11	91%	97-99, 119, 140-141, 170-172, 185-186
app/sql_functions.py	2	2	0%	1-4

- Uma versão em html do relatório de cobertura também está disponível uma vez que os testes tenham finalizado. Para acessá-la, basta acessar o arquivo contido em **backend/htmlcov/index.html**

coverage: 88%

Files Functions Classes

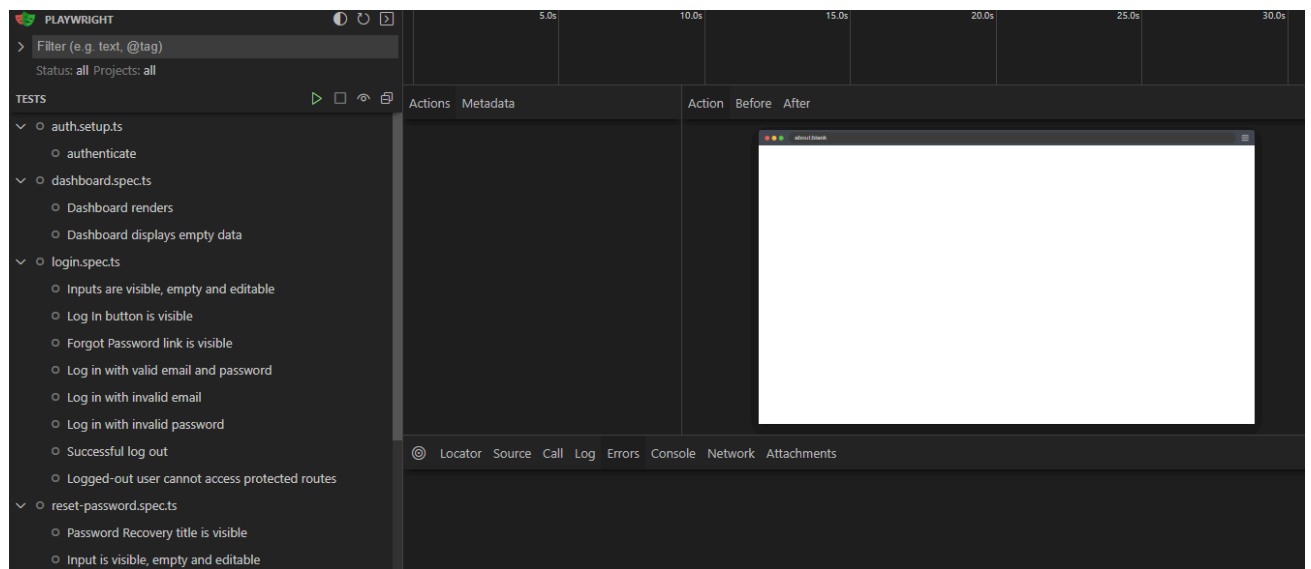
coverage.py v7.6.1, created at 2024-09-16 20:10 +0000

File ▲	statements	missing	excluded	coverage
app/__init__.py	0	0	0	100%
app/api/__init__.py	0	0	0	100%
app/api/deps.py	34	4	0	88%
app/api/main.py	7	0	0	100%
app/api/routes/__init__.py	0	0	0	100%
app/api/routes/auth.py	52	7	0	87%
app/api/routes/sensor_data.py	107	40	0	63%
app/api/routes/users.py	98	0	0	100%
app/api/routes/utils.py	11	3	0	73%
app/backend_pre_start.py	23	7	0	70%
app/core/__init__.py	0	0	0	100%
app/core/config.py	72	8	0	89%
app/core/db.py	10	2	0	80%
app/core/security.py	16	0	0	100%

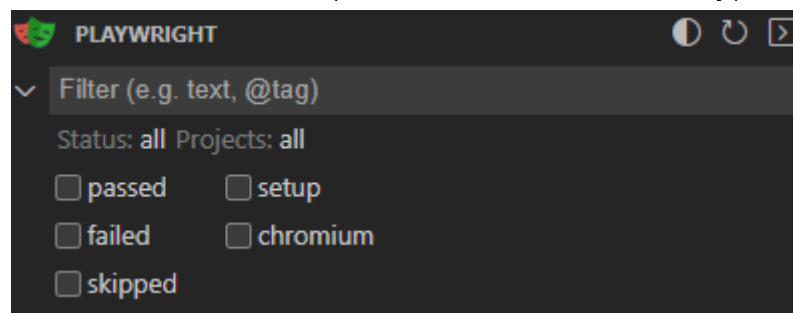
Testes de Integração - FrontEnd

O servidor da aplicação possui testes de integração escritos usando **Playwright**. Para rodar os testes, basta seguir os passos:

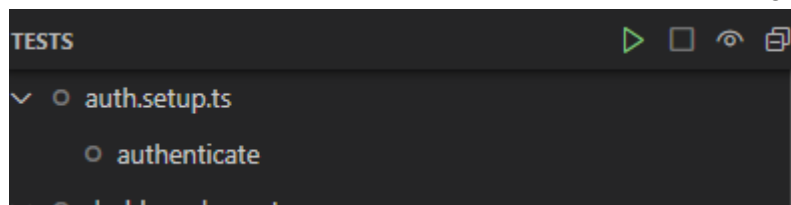
- Inicie a aplicação (veja o tutorial anterior)
- Certifique-se que sua máquina possui Npm, Nvm e Npx instalados.
- Em um prompt de comando (powershell no Windows) acesse o repositório e navegue até a pasta **frontend**
- Execute o comando ***npx playwright test***
Isso executará todos os testes de integração, e irá lhe mostrar os resultados no navegador. Essa abordagem não é preferível para debug, visto que as informações não são facilmente navegáveis.
- Ao invés disso, execute o comando ***npx playwright test --ui***
Isso fará com que uma janela visual interativa seja aberta contendo a interface de testes do Playwright



- Se você não estiver vendo todos os testes (só estiver vendo o **auth.setup**) basta ajustar os filtros:



- Basta clicar em executar  para iniciar os testes de integração

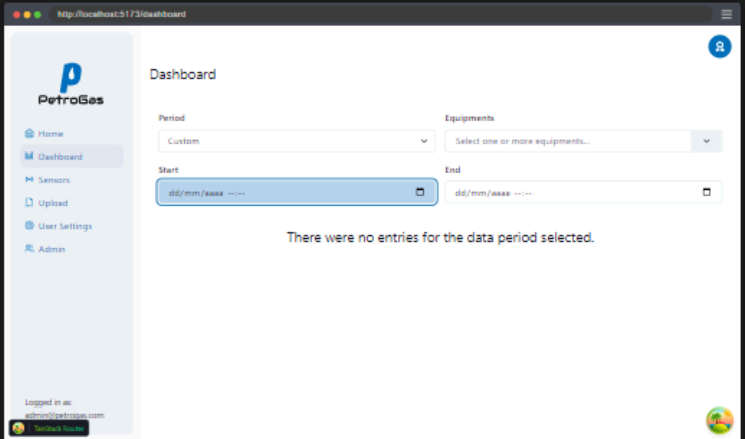


- Uma vez finalizados, os resultados dos testes serão exibidos na tela:

✓	auth.setup.ts	
✓	authenticate	6.1s
✓	dashboard.spec.ts	
✓	Dashboard renders	3.5s
✓	Dashboard displays empty data	4.1s
✓	login.spec.ts	
✓	Inputs are visible, empty and editable	10.9s
✓	Log In button is visible	12.1s
✓	Forgot Password link is visible	4.5s
✓	Log in with valid email and password	4.4s
✓	Log in with invalid email	6.5s
✓	Log in with invalid password	4.7s

- Você também pode interagir com um teste, e ver o passo a passo juntamente com uma visualização do que aconteceu na tela:

Actions	Metadata	Action	Before	After
✓ Passed	12.2s			
> Before Hooks	3.7s			
page.goto /dashboard	3.5s			
locator.selectOption getByLabel('...	2.1s	⊗2		
expect.toBeVisible locator('#begin_c...	92ms			
expect.toBeVisible locator('#end_cus...	51ms			
locator.fill locator('#begin_custom_...	128ms			
locator.fill locator('#end_custom_da...	158ms			
expect.toBeVisible getByText('There ...	53ms			
> After Hooks	2.4s			



OBS: As vezes um teste pode ser **flaky**, ou seja, falhar mesmo quando a aplicação não estiver com erro. Se um teste falhar, tente rodá-lo novamente.