Relatório de Projeto

Ana Raquel Neves Vidal

Engenharia Informática

jul | 2023

GUARDA
POLI
TÉCNICO

POLI TÉCNICO GUARDA

Escola Superior de Tecnologia e Gestão

PROTEÇÕES E LIGAÇÕES DE TERRA

PROJETO
PARA OBTENÇÃO DO GRAU DE LICENCIADO(A) EM ENGENHARIA INFORMÁTICA

Professor(a) Orientador(a): Doutor António Martins

Ana Raquel Neves Vidal
Julho / 2023

POLI TÉCNICO GUARDA

Escola Superior de Tecnologia e Gestão

PROTEÇÕES E LIGAÇÕES DE TERRA

PROJETO
PARA OBTENÇÃO DO GRAU DE LICENCIADO EM ENGENHARIA INFORMÁTICA

Professor(a) Orientador(a): Doutor António Martins

Ana Raquel Neves Vidal

Julho / 2023

POLI TÉCNICO GUARDA

Agradecimentos

Em primeiro gostaria de agradecer ao Instituto Politécnico da Guarda e aos professores de Engenharia Informática por toda a ajuda e conhecimento que me transmitiram ao longo destes 3 anos.

Quero deixar um agradecimento ao meu Orientador do Projeto , o professor António Martins, por me guiar ao longo do projeto, e por toda a sua disponibilidade não só durante a realização do projeto, mas também ao longo de todo o meu percurso académico.

Por fim, não podia deixar de agradecer à minha família e amigos pelo suporte e motivação durante o meu percurso no Instituto Politécnico da Guarda.

POLI TÉCNICO GUARDA

Ficha de Identificação

Aluno

Nome: Ana Raquel Neves Vidal

Nº de Aluno: 1705182

Licenciatura: Engenharia Informática

Estabelecimento de Ensino

Instituto Politécnico da Guarda(IPG)

Escola Superior de Tecnologia e Gestão(ESTG)

Docente Orientador de Estágio

Nome: António Martins

Grau Académico: Doutoramento

POLI TÉCNICO GUARDA

Resumo

O presente Relatório descreve o projeto realizado no âmbito da unidade curricular Projeto de Informática, da Licenciatura de Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda.

O objetivo do projeto consiste no desenvolvimento de uma página web, baseada na micro-framework Flask em Python, para auxiliar no planeamento de Sistemas de Terras. O website deve oferecer as seguintes funcionalidades:

- Analisar o tipo de solo e classificá-lo como homogéneo ou não homogéneo;
- Calcular a resistividade do solo utilizando as fórmulas de Tagg e Dwight;
- Fornecer várias opções de geometria, dependendo da resistividade do solo;
- Calcular o comprimento necessário para o elétrodo escolhido.

A aplicação web possui um menu inicial onde o utilizador poderá selecionar o processo que se ajusta ao problema em questão. Assim, poderá classificar o solo introduzindo as medições do local ou, alternativamente, conhecer a estratificação do solo através de uma tabela fornecida para obter a resistividade do solo. Após reunir todos os dados necessários, o programa apresentará vários tipos de geometrias a implementar e realizará os cálculos necessários utilizando o método de Newton-Raphson para obter a informação necessária para o planeamento do sistema de terras.

Palavras-chave: Tagg, Dwight, Resistência Terra, Solo homogéneo, Solo não homogéneo.

POLI TÉCNICO GUARDA

Abstract

The present Report describes the project carried out under the curricular unit Informatics Project, part of the bachelor's degree in computer engineering at the School of Technology and Management of the Polytechnic Institute of Guarda.

The project's objective is the development of a web page, based on the Flask micro-framework in Python, to assist in the planning of Grounding Systems.

The website must:

- Analyse the type of soil and classify it as homogeneous or non-homogeneous;
- Calculate the soil resistivity using the Tagg and Dwight formulas;
- Provide various geometry options based on the soil resistivity;
- Calculate the necessary length for the chosen electrode.

The web application contains an initial menu where the user can choose the process that fits the specific problem. By doing so, the user can classify the soil by inputting measurements from the location or, alternatively, by knowing the soil stratification through a provided table to retrieve the soil resistivity. After gathering all the data, the program will present various geometries to implement and perform the necessary calculations using the Newton-Raphson method to obtain the information required for the grounding system planning.

Keywords: Tagg, Dwight, Resistência Terra, Solo homogéneo, Solo não homogéneo.

POLI TÉCNICO GUARDA

Índice

Índice de Figuras

Índice de Tabelas

Lista de Siglas e Acrónimos

HTML	HyperText Markup Language
SQL	Structured Query Language
CSS	Cascading Style Sheets
ODBC	Open Database Connectivity
XML	Extensible Markup Language
SSMS	SQL Server Management Studio

POLI TÉCNICO GUARDA

1. Introdução

O presente relatório descreve o projeto desenvolvido pela aluna Ana Raquel Neves Vidal, no âmbito da unidade curricular de Projeto de Informática, pertence ao 3º ano da Licenciatura em Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda.

1.1. Motivação | Enquadramento

A eletricidade tornou-se um essencial imprescindível no nosso dia a dia tanto para carregar um telemóvel como para fazer a nossa comida. Mas todo o cuidado é pouco, para garantir o bom funcionamento dos eletrodomésticos, e a sua segurança muitas vezes é um desafio. A energia elétrica está tão presente no nosso cotidiano, que é comum esquecermos de que realmente a usamos para tudo e a falha desta pode resultar em problemas. Incêndios, choques, curto-circuito, fuga de energia, queima de um eletrodoméstico, estes são alguns exemplos do que pode acontecer caso exista um problema na instalação elétrica.

Deste modo, uma das opções mais seguras de proteger as instalações elétricas, é a ligação de terra. E como a própria palavra indica, tudo tem a ver com a terra. Este tem como objetivo 'transportar a energia elétrica' para um local de descarga segura, isto é, eliminar essa enorme quantidade de energia através da ligação à terra.

Assim, o sistema de terras tem algumas funções importantes:

- Segurança de pessoas: evitando potenciais perigos de toque e de passo, através de terras de baixa impedância e ligações à terra de equipamentos que permitam contactos diretos que possam resultar em tensões perigosas, originadas por defeitos elétricos ou descargas atmosféricas;
- Proteção do equipamento e do edifício: por ligações diretas à terra, de baixa impedância, dos equipamentos elétricos e dos dispositivos de proteção contra sobretensões, de modo a permitir que as correntes originadas por defeitos ou descargas atmosféricas sejam rapidamente dissipadas e não resultem em tensões perigosas. (Portugal)

1.2. Descrição do problema

O problema que foi colocado, pelo orientador de estágio, foi o desenvolvimento de uma aplicação *web* de um sistema de terra onde o principal desafio é detetar com os dados necessários se o solo era homogéneo, ou não e de seguida calcular a espessura do extrato superior. O programa calcula, então, o comprimento necessário consoante a geometria desejada.

1.3. Objetivos

O objetivo do projeto foi, através do *Flask*, desenvolver uma aplicação *web* com os seguintes requisitos:

- Visualizar uma tabela com várias resistividades dos solos, onde estas são fixas;
- Classificar o solo através de medições locais ou usando uma tabela em função do tipo de solo;
- Definir o tipo de geometria a implementar.

Para cumprir com os requisitos foi necessário dividir o trabalho nas seguintes fases:

- Ligação da tabela de dados ao PYTHON;
- Realização do Template em HTML e CSS;
- Estratificando o solo como homogéneo ou não homogéneo;
- Calcular o comprimento necessário tendo em conta a geometria selecionada pelo utilizador.

1.4. Estrutura do documento

Este relatório encontra-se dividido em 7 capítulos.

O primeiro contém a introdução juntamente com os objetivos pretendidos para o projeto. No segundo capítulo está descrita a metodologia adotada ao longo da realização do projeto. No terceiro capítulo é feita a análise de requisitos, em que se identificam as necessidades do software. No quarto capítulo são abordadas as tecnologias que se teve acesso e as ferramentas utilizadas para a construção da aplicação *web*. O quinto capítulo descreve a implementação e o desenvolvimento do sistema de terras. O capítulo sexto define a realização de testes feitos ao longo do projeto. Por fim, o último capítulo, são abordadas reflexões sobre o projeto realizado.

2. Metodologia

A metodologia ágil de *software* usa o desenvolvimento iterativo, isto é cada nova funcionalidade implementada é dada a conhecer ao orientador de estágio para futura validação.

Das variadas metodologias ágeis, para a elaboração deste projeto, a escolhida foi a metodologia Scrum, mas adaptada para um trabalhador individual e para a produtividade pessoal. (Clara, Engenharia de software 1, 2020)

O Scrum é uma abordagem dos princípios ágeis. Criou-se uma lista de tarefas priorizadas (*backlog*) e planearam-se iterações curtas, definindo metas e prazos. A cada iteração, realiza-se um trabalho concentrado nas tarefas escolhidas e também uma revisão junto com o orientador para refletir sobre o progresso e ajustar se necessário.

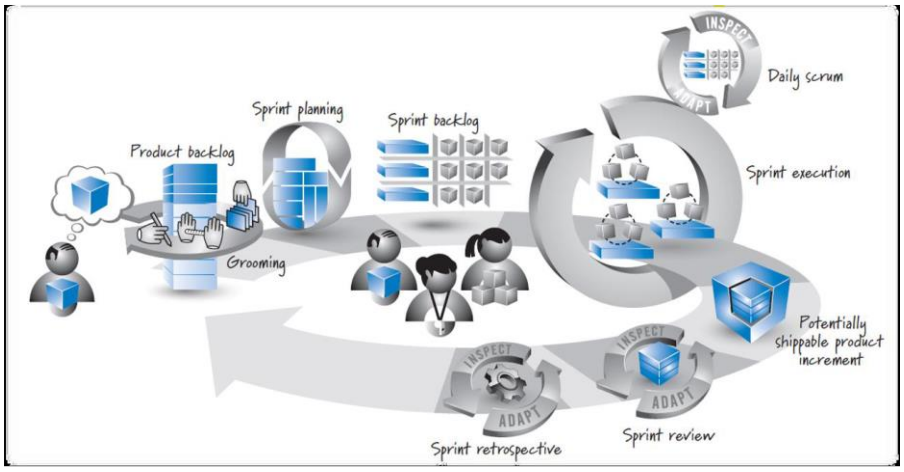


Figura 1-Diagrama de funcionamento Scrum (Rubin, 2012)

De acordo com a figura acima, obtêm se os seguintes papeis no desenvolvimento do projeto.

Product Owner – É quem determina as funcionalidades que devem ser desenvolvidas ao longo do projeto. Neste projeto foi o meu orientador de projeto, professor António Martins.

SCRUM Master – Orientador que assegura que eu siga as práticas e valores do Scrum. No projeto foi novamente o orientador do projeto.

SCRUM Team – Grupo de pessoas que desenvolveu o projeto, onde trabalharam para atingir o objetivo final de cada *Sprint.* Neste projeto, foi apenas a autora.

As funcionalidades a serem implementadas no projeto são mantidas numa lista conhecida como *Product Backlog*. No início de cada Sprint, faz-se um *Sprint Planning Meeting*, ou seja, uma reunião com o intuito de planear, onde o *Product Owner* prioriza as tarefas do *Product Backlog* e a equipe, seleciona as atividades que será capaz de implementar durante o Sprint que se inicia. As tarefas adicionadas ao Sprint são transferidas do *Product Backlog* para o *Sprint Backlog*.

A cada Sprint, realiza se uma breve reunião, chamada *Daily Scrum*. O objetivo é falar sobre o que foi feito, identificar dificuldades e priorizar o trabalho dos dias que se seguem.

3. Análise de Requisitos

A análise de requisitos é fundamental no desenvolvimento de qualquer processo de desenvolvimento de um sistema. Sendo então, nesta fase onde são definidos os requisitos que definem claramente o que a aplicação *web* terá de realizar para satisfazer as necessidades da organização e dos utilizadores.

3.1. Descrição da aplicação

A aplicação *web* a desenvolver é para o planeamento do sistema de terra.

Não existe qualquer impedimento para usufruir da aplicação, sendo apenas necessário aceder ao *link* da página. Deste modo, ao entrar de imediato observa-se a página Home, onde podemos aceder ao menu lateral e escolher três opções, verificar Solo, Solo Homogéneo e visualizar a Tabela, que consta no Anexo 3. Se o utilizador selecionar as primeiras duas opções este terá de preencher um tipo de formulário para a escolha pretendida. Depois da inserção dos dados pedidos o sistema irá apresentar a conclusão dos cálculos efetuados com as informações fornecidas, o utilizador poderá depois escolher o tipo de geometria pretendida ou até mesmo alterar os dados fornecidos inicialmente. Depois de concluído o utilizador será reencaminhado para a página Home.

3.2. Atores e Respetivos Casos de Uso

A tabela 1 contêm o ator da aplicação e os respetivos casos de uso.

Tabela 1-Atores e respetivos Casos de Uso

Atores	Casos de Uso
Utilizador	• Visualizar a tabela das resistividades do solo; • Inserir medições; • Verificar solo; • Selecionar o tipo de solo; • Limpar formulário; • Visualizar num gráfico as medidas inseridas; • Alterar o ponto de visualização do gráfico; • Escolher o tipo de geometria a aplicar.

3.3. Diagrama Casos de Uso

O diagrama de casos de uso descreve a funcionalidade proposta para criar a aplicação *web.* Este possibilita a visualização do papel do ator bem como todos os casos de uso que irão ser alvo de análise.

Na Figura 2, observar-se o diagrama de casos de uso com o ator como utilizador e todos os casos de uso associados a ele. A fronteira delimita os casos de uso que irão ser desenvolvidos no projeto.

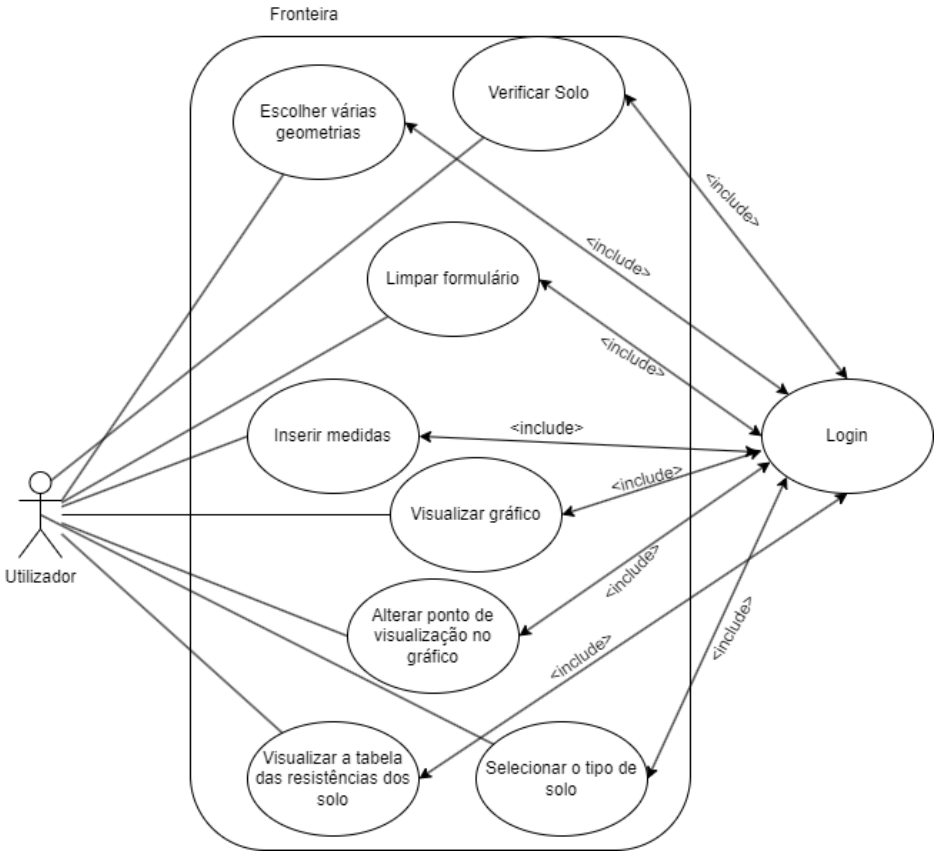


Figura 2-Diagrama Casos de Uso

- O ator, representa a pessoa que implementa os casos de uso;
- Os casos de uso têm um formato oval na horizontal e representam os diferentes casos de uso que o utilizador pode ter;

- Os relacionamentos são representados por uma linha contínua que constitui uma interação entre o ator e o caso de uso.

O relacionamento do tipo “include”, está situado fora da fronteira pois é umas das funcionalidades que será implementada mais tarde.

Como se vê na Figura 2, o ator “utilizador” irá poder verificar a homogeneidade do solo em casos de dúvidas e depois prosseguir para a escolha da geometria, se souber a homogeneidade, simplesmente escolhe o tipo de solo e procede para a escolha da geometria a implementar.

3.4. Descrição Casos de Uso

De forma a entender melhor de como o utilizador irá interagir com a aplicação é crucial fazer uma descrição dos casos de uso. Sendo uma aplicação *web,* esta está previamente estruturada, portanto existe pouca liberdade para realizar novos caminhos a não ser os já definidos, mas ainda assim é importante realizar esta descrição, pois estas são bastante importantes para detetar falhas.

3.4.1. Verificar Solo

Este caso de uso é provocado quando o utilizador quer aplicar o sistema de terra, mas não sabe o tipo de solo, isto é a sua resistividade e se o solo é constituído por um estrato ou por dois ou mais. Neste projeto apenas se vai verificar até um número máximo de dois diferentes. Deste modo, o utilizador seleciona a opção “verificar solo”. A Tabela 2 descreve este caso de uso:

Tabela 2-Descrição Caso de Uso "Verificar Solo"

Nome	Verificar Solo
Descrição	Este caso de uso tem como objetivo classificar o solo.
Pré-Condição	Inserir as 4 medições pedidas
Caminho Principal	1. O ator clica na opção “Verificar Solo”. 2. O sistema apresenta uma página para introduzir as medições necessárias. 3. O ator introduz os dados necessários.

	4. O sistema apresenta um gráfico com as medições introduzidas, mostrando os pontos de inflexão calculados. 5. O ator analisa e verifica se vai ao encontro do que introduziu. 6. O Sistema retorna a classificação do solo, apresentado as conclusões do solo.
Caminho Alternativo	3. a)O sistema não apresenta o gráfico pois as medidas não são válidas. 4. a)O ator verifica que introduziu mal uma medida e insere novamente as medições.

3.4.2. Tipo Geometria

Este caso de uso acontece devido ao facto de o utilizador saber o tipo de solo onde irá realizar o sistema de terras e seleciona a opção ”Solo Homogéneo”. A Tabela 3 descreve este caso de uso:

Tabela 3-Descrição Caso de Uso "Tipo Geometria"

Nome	Tipo Geometria
Descrição	Este caso de uso tem como objetivo calcular o comprimento necessário dependo do tipo e geometria.
Pré-Condição	Inserir as 4 medições pedidas
Caminho Principal	1. O ator clica na opção “Tipo Geometria”. 2. O sistema apresenta uma página para introduzir os dados necessárias. 3. O ator introduz os dados necessários. 4. O sistema apresenta um resultado de acordo com as medições introduzidas, isto é, o comprimento necessário para a geometria selecionada. 5. O ator analisa e verifica se vai ao encontro do que introduziu.
Caminho Alternativo	3. a)O ator introduz dados que não sejam validos.

4. Tecnologias

Neste capítulo serão apresentadas as tecnologias usadas para o desenvolvimento da aplicação.

4.1. HyperText Markup Language

HyperText Markup Language, mais conhecido pela sigla HTML, é uma linguagem de marcação padrão que define a estrutura e o conteúdo de uma página web. Para isso utiliza "*tags*" que são interpretadas pelo *browser* e exibidas ao utilizador.

4.2. Cascading Style Sheet

Cascading Style Sheet, conhecido pela sigla CSS, é uma linguagem de folhas de estilo utilizada para personalizar a parte visual de um documento escrito em linguagem de marcação, como por exemplo o HTML ou XML. Normalmente a aplicação de estilos é realizada num documento à parte, devido a uma melhor organização, e depois faz-se referência ao ficheiro CSS no ficheiro HTML, sendo também possível incluir os estilos diretamente no ficheiro HTML.22. A framework "boostrap", permite aplicar estilos e adicionar funcionalidades aos elementos que constituem uma página web de forma rápida e simples. Também permite que a página se torne responsável e adaptável a qualquer dispositivo. Por estes motivos no desenvolvimento da aplicação *web* foi utilizada esta *framework*.

4.3. Microsoft SQL Server Management Studio

O SQL Server Management Studio (SSMS) é um software para gestão de qualquer infraestrutura SQL. Este fornece recursos para configurar, monitorizar e administrar os componentes do SQL Server e bancos de dados . Durante o desenvolvimento do projeto a tabela foi criada utilizando este software.

4.4. Python

Python é uma linguagem de programação de alto nível, utlizada para desenvolver aplicações de todos os tipos. Diferenciada do *Java* e do *.Net* , trata-se de uma linguagem interpretada, ou seja, tende a ser mais flexível, pois contem um interpretador, que faz a conversão do código linha a linha, em vez de ser compilada, de uma só vez.

4.5. Flask

O *Flask* é uma *micro-framework* destinada principalmente para aplicações mais pequenas e com requisitos mais simples e assim mais rápido.
A sua simplicidade, por possuir apenas o necessário para o desenvolvimento de uma aplicação, torna-a mais leve comparada com outras *frameworks*.

4.6. Zoom

O *Zoom* é uma das principais aplicações de *software* de videoconferência. Permite interagir virtualmente com a pessoa que necessita de contactar, sendo assim útil pois permitiu no desenvolvimento do projeto realizar várias reuniões online para esclarecimento de dúvidas e atualizações do projeto.

4.7. Outlook

Microsoft Outlook é um *software* que permite enviar e receber e-mails. Este foi bastante usado para a partilha de documentos e comunicações entre os intervenientes no desenvolvimento do projeto.

4.8. JavaScript

O JavaScript é uma linguagem leve, com funções de primeira classe, e esta é mais conhecido como a linguagem de script para páginas *Web,* sendo também muito dinâmica e o seu principal uso no projeto foi na implementação de validações, facilitando a rotas de validações que são um bocado limitativas usando a framework Flask.

5. Implementação

Neste capítulo, é feita a descrição das etapas realizadas durante o desenvolvimento do projeto Sistema de Terras.

5.1. Arquitetura do sistema

Na Figura 3, mostra a arquitetura da aplicação, dividida em três partes, *frontend*, *backend* e a base de dados SQL Server Management.

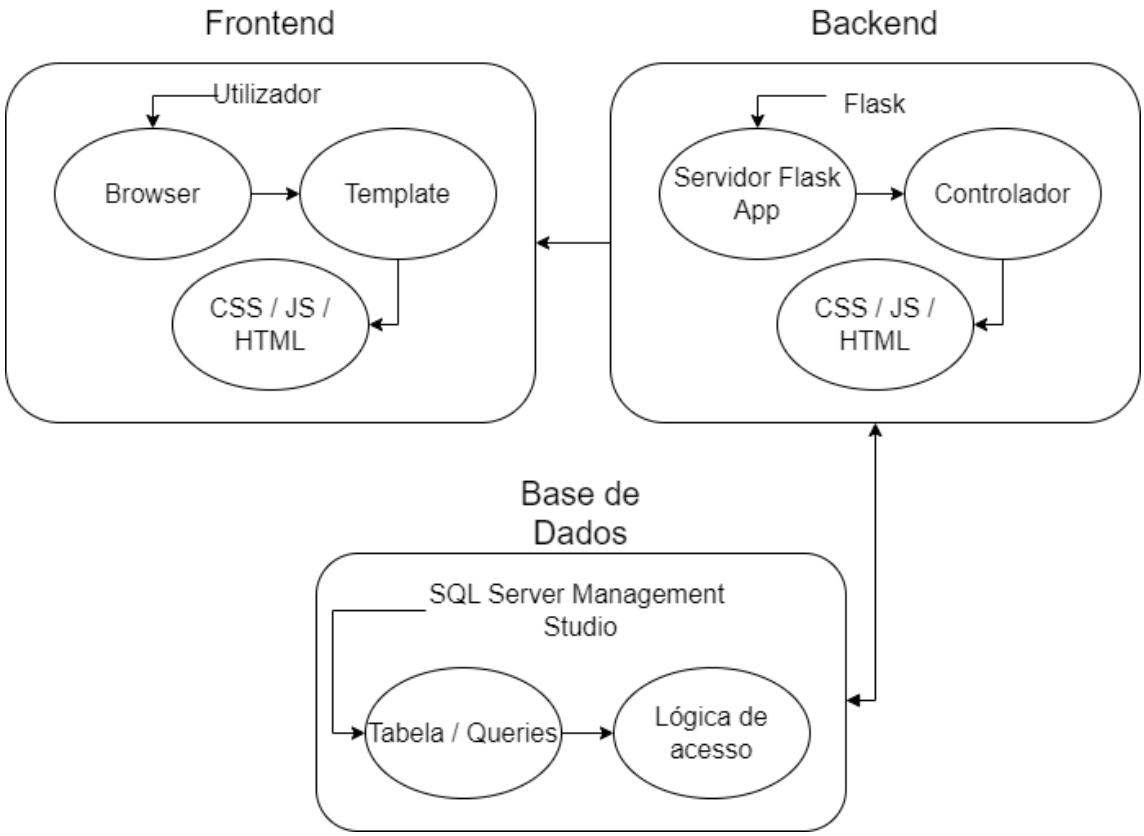


Figura 3-Arquitetura do Sistema

O *Python* utiliza a biblioteca pyodbc, Anexo 5, esta é um modulo que fornece uma interface para estabelecer a ligação à base de dados usando o ODBC(*Open Database Connectivity*), Anexo 4.1.

O Utilizador envia os dados do formulário preenchido na página para o *backend* através do método "POST" definida com uma dada rota através do comando **app.route()**, e o backend obtém os dados usando o método **request.form**.

A estrutura de cada parte é discutida nos seguintes subtópicos.

5.2. Estrutura Frontend

Para desenvolver o projeto em *Flask*, Anexo *6,* foi necessário realizar os seguintes passos:

- Executar o seguinte código no terminal, *pip install Flask*;
- Importar as seguintes bibliotecas, *Flask, render_template;*
- Criar uma pasta Template, Figura 4, dentro desta pasta, um ficheiro chamado *home.html*;
- E por fim executar o código '*python app.py'* no terminal, para depois conseguir aceder à página pelo *browser*.

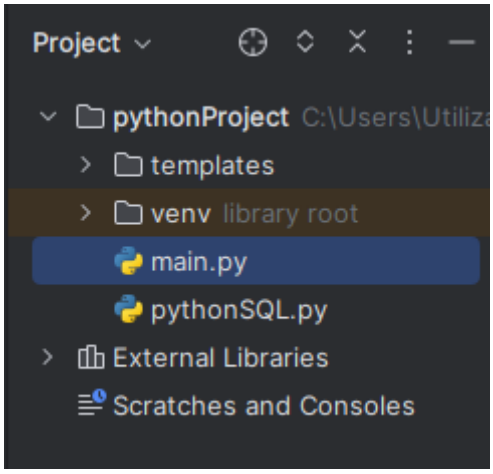


Figura 4-Estrutura da aplicação Flask

Todas as interfaces necessárias estarão localizadas dentro da pasta "templates", como se observa na Figura 5.

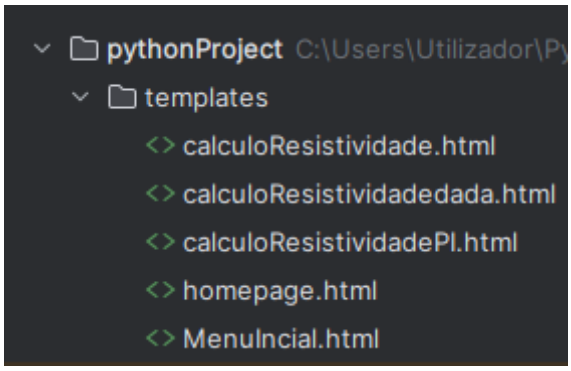


Figura 5-Pasta Template detalhada

Para estas serem visualizadas foram criadas funções, como ilustrado na Figura 6, que estão associadas a um Template HTML que define a *página* da aplicação.

```
@app.route("/homepage", methods=['POST'])
def calculaSoloHomogenio():
    global r, h
    ...
```

Figura 6-Função associado a um Template HTML

Todas as *páginas* irão ter o menu de barra lateral igual, sendo que a única diferença é o formulário nas diferentes páginas, como está representado na Figura 7.

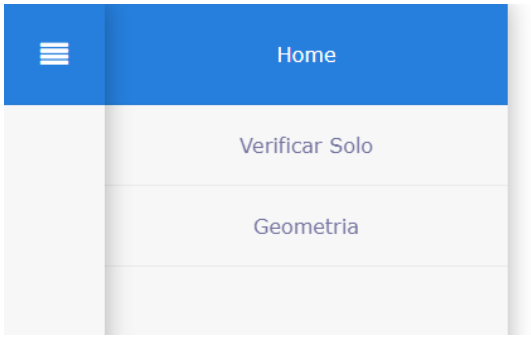


Figura 7-Barra Lateral

5.3. Criação do Backend

Nesta parte abordam-se a criação do *backend* e a sua estrutura, que consta na Figura 8.
O backend foi desenvolvido em *Python*, e usando o *Flask* que é uma micro-framework usada em *python* que fornece uma maior flexibilidade e acessibilidade na realização da aplicação *web.*

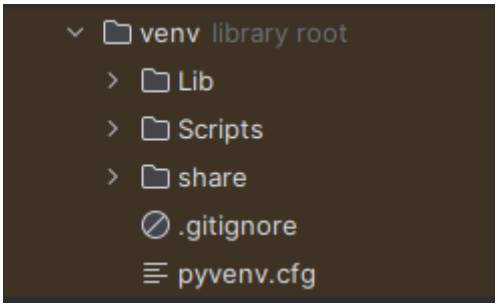


Figura 8-Estrutura backend

Observando a Figura 8, pode-se ver que a estrutura do backend está dividida em três pastas, lib, Scripts e share.

Lib – Nesta pasta contém todas as bibliotecas/pacotes do *python* instaladas, usando o comando "*pip install*", estes são automaticamente descarregados para dentro desta pasta.

Scripts – Contém scripts executáveis para ativar o ambiente virtual e instalar pacotes. Estes são específicos para cada sistema operativo.

Share – Nesta pasta pode conter metadados ou outros recursos relacionados com os pacotes instalados na venv.

5.4. Criação das páginas em HTML e CSS

Este subcapítulo refere a construção das páginas em HTML e CSS. Sendo uma linguagem de marcação e de folha de estilo, respetivamente, foi uma escolha obvia para a construção da aplicação.

Na utilização destas duas linguagens consegui facilmente criar uma página *web*, bem estruturada e funcional, sendo que devido à sua vasta comunidade de desenvolvedores facilmente solucionei problemas e apliquei componentes sem grandes dificuldades.

Na sua construção usaram-se alguns componentes como ***forms*, *submit buttons*, *radio buttons e tables***.

5.5. Validações do *frontend*

Antes de enviar os dados preenchidos pelo utilizador foi necessário realizar uma validação de forma a perceber se todos os campos foram preenchidos corretamente.

O linguagem utilizado para fazer essas validações foi o *JavaScript*. Uma das principais vantagens de a usar para a validação de dados num formulário é que esta pode ser feita no lado do *browser,* isto é, a validação torna-se mais rápida dando um feedback instantâneo, sem precisar de esperar pela resposta do servidor. Esta validação encontra-se ilustrada na Figura 9.

```
<!-- Uso o javascript de forma a tornar mais fácil as validações na aplicação web -->
<script>
  function validarform(){
    var input = document.querySelectorAll('input[type="text"]');
    for(var i = 0; i < input.length; i++){
      if(input[i].value === "") {
        alert("Por favor preencha todos campos");
        return false;
      }
    }

    var inputpermitido = /^[0-9]+(\.[0-9]+)?$/;
    for(var j = 0; j < input.length; j++){
      if(!inputpermitido.test(input[j].value)){
        alert("Introduza somente números");
        return false;
      }
    }
    return true;
  }
</script>
```

Figura 9-Código validação forms

Por vezes foi necessário atualizar as validações dependendo do input do utilizador, um exemplo dessa situação é representado na Figura 10.

O utilizador ao escolher a opção um, irá aparecer um campo, chamado comprimento e este vai ser necessário para realizar os cálculos necessários do comprimento necessário para a geometria da vara, mantendo a resistividade da mesma com um limite máximo de 20 ohms, de acordo com a legislação.

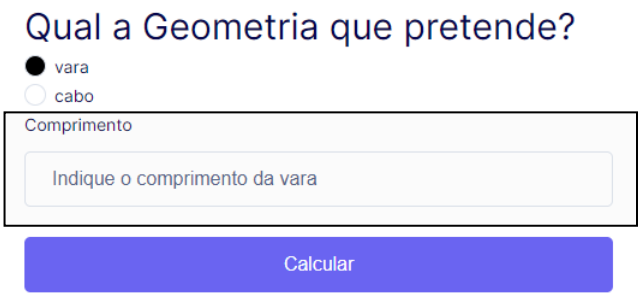


Figura 10-Interface Verificar Solo

Ilustra-se o programa da função **validarForm()**.

```
<script>
  function validarForm() {
    var radioButtons = document.getElementsByName("tipo");
    var radioButtonSelecionado = false;
    var input =
document.querySelectorAll('input[type="text"]');
    //realizo o ciclo para verificar todos os inputs
    //mas apenas quero verificar se um está selecionado
    for (var i = 0; i < radioButtons.length; i++) {
      if (radioButtons[i].checked) {
        radioButtonSelecionado = true;
        break;
      }
    }
    // condição boleana se este for falso
    if (!radioButtonSelecionado) {
      alert("Por favor, selecione uma das opções.");
      return false;
    }
    // Verificar qual é opção selecionada antes de fazer a
validação
    //vara selecionada
    if (radioButtons[0].checked) {
      // verificar se a opção vara está selecionada e se o
campo distancia está preenchido
      var distanciaVaras =
document.getElementById("distancia").value;
      // se não estiver preenchido
      if (!distanciaVaras) {
        alert("Por favor, preencha o comprimento da
vara.");
        return false;
      }
    //cabo selecionada
    } else if (radioButtons[1].checked) {

      //verificar os campos dessa opção se estão
preenchidos
      var raioInterno =
document.getElementById("RaioInterno").value;
```

```
      var raioExterno =
document.getElementById("RaioExterno").value;
      var resistividadecabo =
document.getElementById("resistividadecabo").value;
      //Se não estiverem preenchidos
      if (!raioInterno || !raioExterno ||
!resistividadecabo) {
        alert("Por favor, preencha todos os campos para a
opção cabo.");
        return false;
      }
    }
    return true;
  }
</script>
```

A função "**validarForm()**" é chamada sempre que se clica no botão "Calcular" e, essencialmente, verifica qual a opção selecionada e se todos os campos estão devidamente preenchidos. No caso dos botões de seleção, apenas um pode ser escolhido de cada vez. Depois de determinar se está selecionada a opção "vara" ou "cabo", verifica também se os campos associados estão preenchidos corretamente. Caso algum campo obrigatório esteja em branco ou preenchido incorretamente, é exibida uma mensagem de alerta ao utilizador, informando o que precisa ser corrigido, conforme demonstrado nas Figuras 11 e 12.

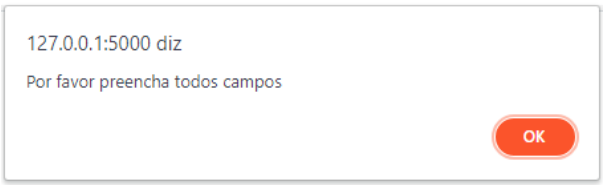


Figura 11-Validação 1

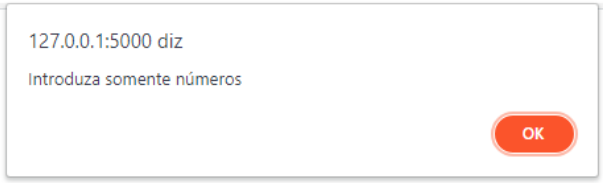


Figura 12-Validação 2

5.6. Tipo de Solo

Neste subcapítulo, é explicado o desenvolvimento realizado na verificação do Solo. O utilizador tem a opção de verificar se o solo é homogéneo ou não homogéneo, caso tenha dúvidas, uma vez que diferentes recursos serão aplicados com base na sua classificação. A escolha da geometria depende da estratificação do solo.

Se for homogéneo, os principais sistemas de terra, são:

- Uma vara cravada no solo;
- Duas varas alinhadas;
- três varas em triângulo;
- Quatro varas em quadrado;
- Varas em círculo;
- Cabos enterrados no solo em cruz, estrela, e em quadrados.

Para o cálculo das geometrias usa-se as fórmulas de Dwight (Dwight, December 1936).

O sistema de terra a adotar depende do tipo de sistema elétrico do local e do custo.

Se não homogéneo, as principais geometrias para os sistemas de terras são:

- Vara cravada no solo;
- Cabo enterrado no solo;

Para o cálculo destas geometrias usa-se as fórmulas de Tagg (Tagg, 1964).

5.6.1. Solo não homogéneo

5.6.1.1. Frontend

Primeiramente, na análise do tipo de solo, o utilizador deverá realizar quatro medidas e preenchê-las no formulário da página, como é ilustrado na Figura 13. Essas medidas representam a resistividade do solo em diferentes profundidades, sendo o primeiro relativamente á profundidade e o segundo relativo á resistividade dessa mesma profundidade.

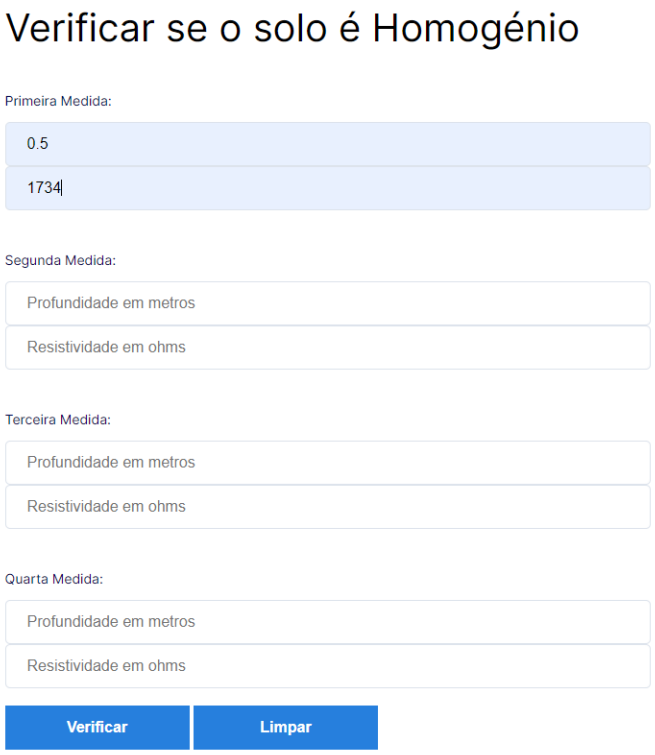


Figura 13-Interface verificar solo

Depois de preenchido o formulário a aplicação irá apresentar um gráfico, Figura 14, de forma a dar uma perspetiva visual ao utilizador, dos dados que inseriu.

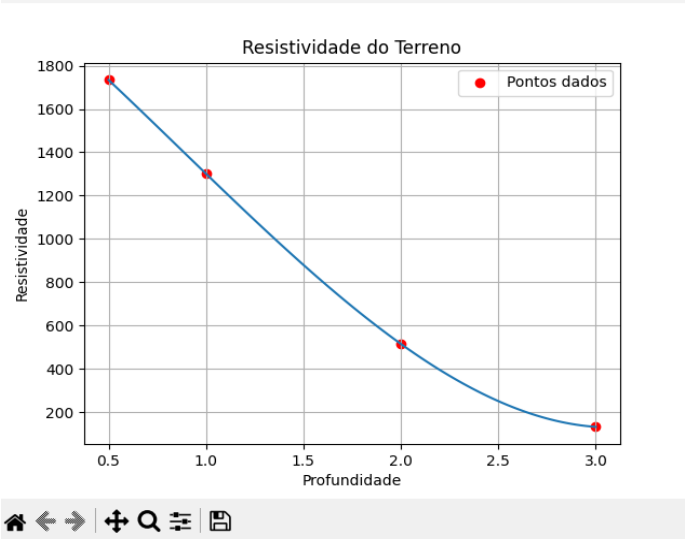


Figura 14-Gráfico da resistividade do Terreno

Sendo o solo classificado como não homogéneo, é perguntado ao utilizador qual a geometria que pretende usar, sendo lhe dado duas opções "vara" ou "cabo".

Se o utilizador optar pela vara a aplicação gera um campo para a inserção do comprimento da vara, Figura 15.

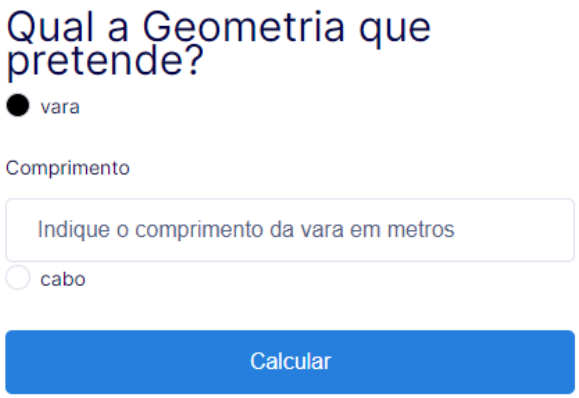


Figura 15-Geometria vara selecionada

Se o utilizador optar pelo cabo uma vez mais a aplicação irá gerar um campo para a inserção do raio do cabo, com a resistividade do mesmo, Figura 16.

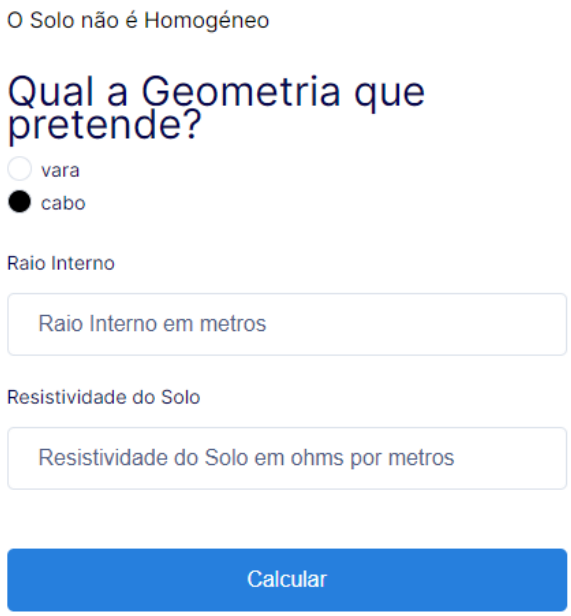


Figura 16-Geometria cabo selecionada

Por fim, o programa irá apresentar o resultado consoante os dados fornecidos pelo utilizador.

No caso da vara, Figura 17:

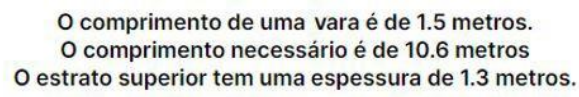


Figura 17-Output vara

No caso do cabo, Figura 18:

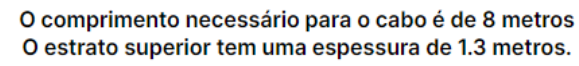


Figura 18-Output cabo

5.6.1.2. Backend

Numa primeira fase da análise do tipo de solo, o utilizador terá de fazer quatro medidas e preencher no *forms* da página. Esta análise irá consistir em dois "outputs" homogéneo ou não homogéneo, para tal o programa faz as seguintes análises através das medidas dadas:

- Contém pontos de inflexão?
- Apresenta alguma forma de concavidade?
- A resistividade é inferior a 30% da resistividade média?

Isto foram pontos cruciais para a verificação do solo. Desta forma, explica-se a implementação do código segundo os pontos mostrados acima.

Na função calculasoloHomogeneo() obtêm-se os dados preenchidos pelo utilizador e realiza os seguintes passos:

- Recolha e análise dos valores introduzidos pelo utilizador;
- Aumentos e diminuição de declive acima dos 20%;
- Encontrar os pontos de inflexão;
- Desenhar o gráfico.

Na recolha e análise dos valores, através do método "*post"* recolhe-se os dados com o seguinte comando, **request.form.get(),** de seguida estes pontos irão ser adicionados a um array pela instrução **array.append(),** numa segunda fase calcula-se a resistividade média, através de fórmulas convencionais.

Deste modo, analisa-se se existe entre os pontos dados um aumento ou diminuição significativa nos seus declives, isto é, através da biblioteca *numpy* que permite calcular a derivada discreta ao longo de um ou mais eixos de um array. Então, com o comando **np.gradient(r, p)** calcula o declive do array r (resistividades) em relação aos valores do array p (profundidade).

Através de um ciclo, são identificadas diferenças significativas entre os pontos fornecidos pelo utilizador, ou seja, aquelas que estão acima de 30% e abaixo de 90%. Em seguida, as abcissas onde o declive ultrapassa esses valores são armazenadas. Essas percentagens foram selecionadas com base em experimentações realizadas em medidas anteriores e análises efetuadas.

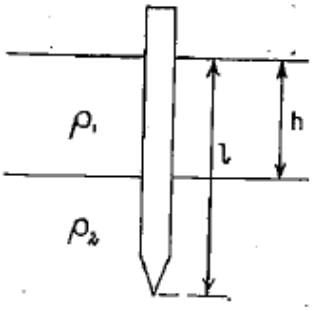
Ao nível dos pontos de inflexão, usando uma vez mais a função **np.gradient(r, p),** foi-se calcular a derivada discreta de segunda ordem, para detetar a mudança de concavidade, verificando então uma mudança de sinal e guardar os pontos num array.

De seguida, é removido os pontos duplicados nas abcissas e ordenadas e são atualizadas de acordo com os índices obtidos, assim realiza-se uma interpolação cúbica dos pontos usando a função **interp1d()**, para obter uma curva suave a partir dos pontos.

O cálculo dos pontos de inflexão é realizado duas vezes usando a derivada, com o objetivo de garantir uma maior precisão dos pontos.

Deste modo, a primeira parte é responsável por encontrar os pontos de inflexão na curva original dos dados fornecidos pelo utilizador, através da derivada. Depois, os dados são interpolados para obter uma curva suave, em seguida a segunda derivada é calculada **df_interp** e assim os pontos de inflexão são detetados na curva suave. A segunda parte serve para verificar se os pontos de inflexão encontrados estão próximos um dos outros. A variável tolerância define uma distância limite entre os pontos, para determinar se são considerados próximos, e se a distância entre os dois pontos for menor que a tolerância estabelecida, apenas o ponto mais significativo, isto é, o ponto mais afastado dos outros é mantido, e os restantes são descartados. Usa-se esta abordagem para eliminar pontos de inflexão repetidos e próximos um dos outros, uma vez que quando queria verificar se o solo era homogéneo apenas diferencia entre os pontos resultava num solo não homogéneo.
Com isto, a aplicação conforme os dados obtidos classifica o solo como homogéneo, se não houver pontos de inflexão ou grandes variações no declive, ou não homogéneo, se estas duas não se verificarem.
Tendo sido classificado o solo como não homogéneo, recorreu-se às seguintes formulas de Tagg, para o cálculo do comprimento da vara.

Para a vara:



$$R = \frac{\rho_1}{2\pi l}\frac{1+k}{1-k+2k\frac{h}{l}}\left[\ln\left(\frac{2l}{a}\right)+\sum_{n=1}^{\infty}k^n\ln\left(\frac{2nh+l}{(2n-2)h+l}\right)\right]$$

- R é a resistividade da vara;
- ρ1 é a resistividade do estrato superior do solo;
- l é o comprimento da vara;
- k, coeficiente de reflexão da tensão $k = \frac{\rho 2 - \rho 1}{\rho 2 + \rho 1}$;
- h é a espessura do estrato superior do solo;
- a é a distância da vara;
- ∑(n=1, ∞) representa a soma infinita dos termos subsequentes.

Obtém-se o valor do k através da função **supinf(),** anexo 1.2, para obter a resistividade superior e inferior, com a condição, se os pontos adjacentes ao mesmo tiverem um aumento ou diminuição de 10%, ele calcula a media dos mesmos, obtendo assim a resistividade superior e inferior**.**

Para calcular o ciclo infinito, a fórmula de Tagg é dividida em três partes, realizando simplificações como $\ln(2 * l) = \ln(2) + \ln(l)$ e também $\ln\frac{2}{l} = \ln(2) - \ln(l)$. Isso é necessário devido a limitações da biblioteca sympy, especificamente da função sy.lambdify(), que não consegue lidar com a multiplicação de um valor numérico por uma variável simbólica. Portanto, a última parte da fórmula de Tagg representa a soma infinita dos termos, com o comprimento da vara (L). Para calcular essa soma infinita, utilizamos o ciclo "while" para calcular a soma até que a diferença entre dois termos consecutivos seja menor que um por cento da soma acumulada, conforme ilustrado na função **calcularResistividadeVara().**

Função **calcularResistividadeVara()**.

```
# Parte 1 da equação de Tagg
parte1 = (rs / (2 * math.pi * l)) * ((1 + k) / (1 - k + 2 *
k * (h / l)))

parte2 = (np.log(2) + sy.log(l) - np.log(a))

# Parte 3 da equação de Tagg (soma finita)
parte3 = (k ** n) * (np.log(2) + np.log(n) + np.log(1 + 1 /
(2 * n * h))) / ((2 * n - 2) * h + 1)
print("parte3: ", parte3)

termo = (k ** n) * (sy.log(2) + sy.log(n) + sy.log(1 + 1 /
(2 * n * h))) / ((2 * n - 2) * h + 1)

tolerancia_soma = parte3 * 0.01
print("Passou aqui")

#Soma infinita
while (abs(termo) > abs(tolerancia_soma)):
    parte3 += termo
    n += 1
    termo = (k ** n) * (np.log(2) + np.log(n) + np.log(1 +
1 / (2 * n * h))) / ((2 * n - 2) * h + 1)
    print(n)
    tolerancia_soma = parte3 * 0.01
```

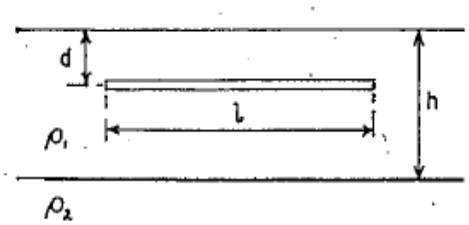
Para encontrar o comprimento da vara utiliza-se o método iterativo de Newton-Raphson, este método foi uma aplicação igual ao que foi aprendido nas aulas de Métodos Numéricos, Figura 19. Inicialmente configura-se o método de Newton e estabelece-se variáveis como a tolerância, máximo de iterações e o ponto inicial.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Figura 19-Método Newton-Raphson (Valente, 2018)

De seguida, convertem-se as expressões para funções numéricas através da função **sy.lambdify(),** e com um loop 'for' realizam-se as iterações i = 1 até i = 3, encontrando então a raiz da equação.

Para o cabo:



$$R = R_{\text{hom}} + \frac{2\rho_1}{\pi l}\sum_{n=1}^{\infty} k^n \left[\ln \frac{1+\sqrt{\left(\frac{2nh}{l}\right)+1}}{2n\frac{h}{l}} + \frac{2nh}{l} - \sqrt{\left(\frac{2nh}{l}\right)+1} \right]$$

- R é a resistividade da vara;
- ρ1 é a resistividade do estrato superior do solo;
- l é o comprimento da vara;
- k é a razão entre as resistividades do estrato inferior e superior do solo;
- h é a espessura do estrato superior do solo;
- a é a distância da vara;
- ∑(n=1, ∞) representa a soma infinita dos termos subsequentes.

Para determinar o valor de "k", mais uma vez, utilizou-se a função supinf() para obter a resistividade superior e inferior. Nesse caso, a condição estabelecida foi que, se os valores adjacentes tiverem um aumento ou diminuição de 10%, a média desses valores seria calculada, resultando na resistividade superior e inferior.

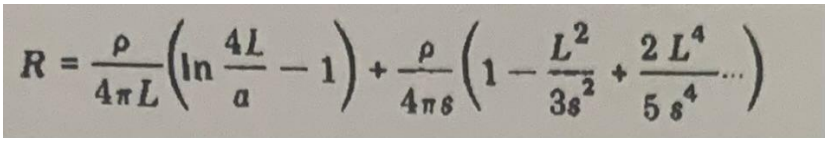
Para calcular o ciclo infinito, seguiu-se o mesmo processo utilizado para a vara, o qual pode ser consultado no Anexo 1.3. O mesmo procedimento foi adotado para o método de Newton.

5.6.2. Solo homogéneo

5.6.2.1. Frontend

Nesta parte, mais uma vez o utilizador introduz as medidas, gerando um gráfico para uma melhor visualização dos dados, e retorna o "output" dizendo ao utilizador que o solo é homogéneo.

Deste modo, o utilizador indicar a geometria que quer, esta deve ser sempre escolhida consoante o problema em questão. Depois de selecionada o programa calcula utilizando as fórmulas de Dwight, Figura 20.



$$R = \frac{\rho}{4\pi L}\left(\ln\frac{4L}{a} - 1\right) + \frac{\rho}{4\pi s}\left(1 - \frac{L^2}{3s^2} + \frac{2L^4}{5s^4} \cdots\right)$$

Figura 20-formula para calcular a resistência de dua varas (Dwight, December 1936)

5.6.2.2. Backend

Observa-se o seguinte código implementado segundo a Figura 20.

```
parte1 = (rs / (4 * math.pi * v))
print("parte1 ", parte1)
parte2 = (sy.log((4 * v) / d) - 1) + (rs / (4 * math.pi *
d_varas))
print("parte2 ", parte2)
parte4 = (1 - (v ** 2 / (3 * (d_varas ** 2))) + ((2 * (v **
4)) / (5 * (d_varas ** 4)))) - r
print("parte4 ", parte4)

equacao = parte1 * parte2 * parte4

# derivada da função
derivada = sy.diff(equacao, v)
print("derivada : ", derivada)

# método de Newton-Raphson
tolerancia = 1e-6
max_iteracoes = 4
l_aproximado = xn
```

```
# Converter expressões para funções numéricas
equacao_numerica = sy.lambdify(v, equacao)
derivada_numerica = sy.lambdify(v, derivada)

for _ in range(max_iteracoes):
    l_anterior = l_aproximado
    l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
    if abs(l_aproximado - l_anterior) < tolerancia:
        break

valor_l = l_aproximado
```

5.6.3. Solo homogéneo conhecida a Resistividade do Solo

5.6.3.1. frontend

Inicialmente se o utilizador, souber o solo onde vai aplicar o sistema de solo, escolhe a opção indicada ao solo em questão, Figura 2Figura 211.

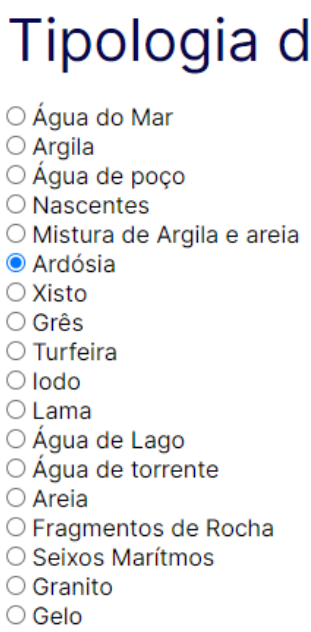


Figura 21-Tipologia do solo

De seguida, seleciona-se a geometria que se quer aplicar consoante o problema em questão, como mostra na Figura 22.

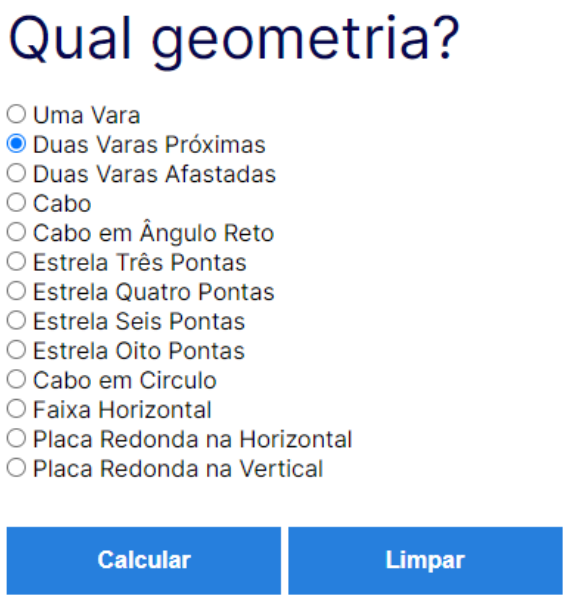


Figura 22-Geometria duas varas

Depois da escolha da geometria é redirecionado para a página onde terá de preencher o formulário para o cálculo do comprimento necessário, Figura 23.

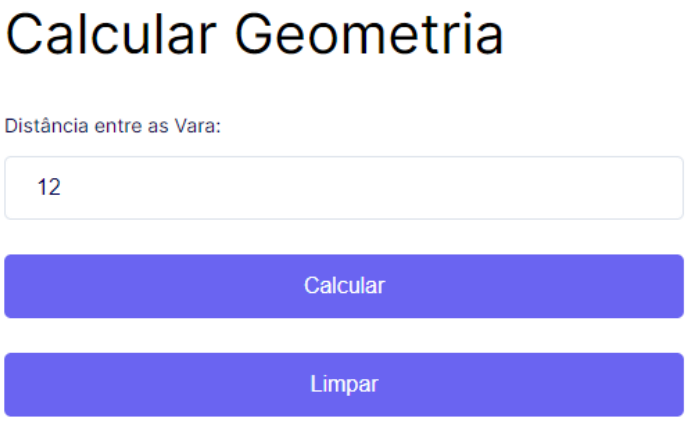


Figura 23-Calcular geometria

Deste modo, ao clicar no botão “Calcular” a aplicação irá redirecionar o utilizador para a página “solo homogéneo” apresentando o “output”, neste caso com um comprimento necessário de 4,1 metros.

5.6.3.2. Backend

Ao nível do backend o método que se usa para obter os dados é o método “post”, que é utilizado para obter os dados enviados pelo utilizador quando este preenche e clica no botão para enviar. Depois, através biblioteca *sympy* para a expressão da equação que será posteriormente resolvida pelo método de Newton-Raphson.

O método de Newton-Raphson é um algoritmo de aproximação numérica, neste caso para calcular o comprimento necessário para a vara, através de um ciclo para realizar as iterações. Este continua até que a diferença entre o valor anterior(l_aproximado) seja menor que a tolerância definida.

As expressões “equação” e “derivada” são convertidas em funções numéricas usando a função sy.lambdify(), deste modo permitir o cálculo da expressão numérica, Anexo 1.11.
Uma vez terminado o ciclo, o valor do comprimento necessário é armazenado no variável “valor_l”.

6. Verificação e Validação

O teste de software é um processo para avaliar a funcionalidade da aplicação com a intenção de descobrir se o projeto está a funcionar como esperado ou não, garantindo assim que o produto final é um produto de qualidade.

Os testes foram conduzidos pelo autor do projeto, utilizando medições e um documento em Excel que fornecia os parâmetros necessários para obter os resultados corretos. Esses parâmetros foram disponibilizados pelo orientador do projeto. Além disso, foram utilizadas medições obtidas em livros fornecidos pelo mesmo orientador.

6.1. Verificar solo

Na Figura 24 é apresentado um gráfico da resistividade do solo com a profundidade. Devido á mudança de concavidade facilmente detetada, consegue-se classificar o solo como não homogéneo.

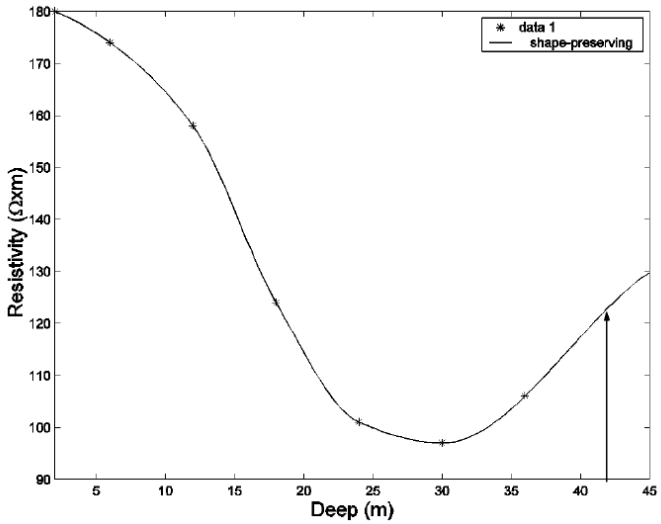


Figura 24-Resistividade do solo com a profundidade (Martins, 2020)

De modo a validar a aplicação foram introduzidos os mesmos pontos apresentados na Figura 24 e com isto obtemos o seguinte gráfico.

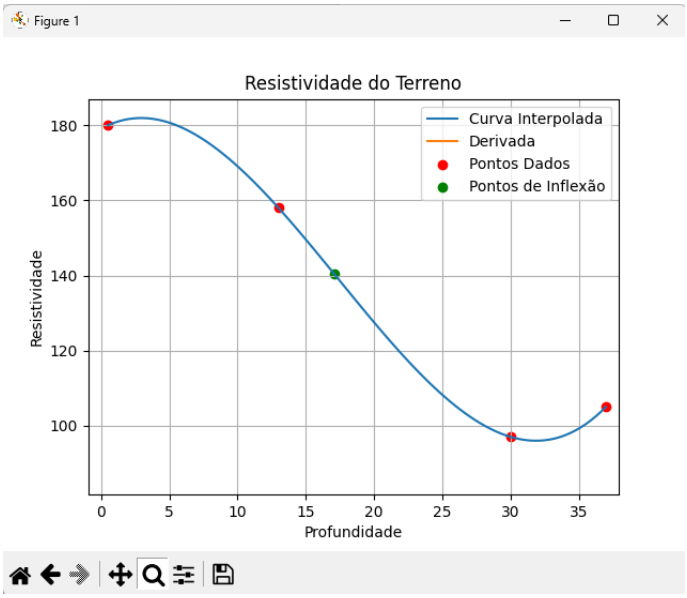


Figura 25-Resistividade do solo com a profundidade referente á aplicação web

Uma vez representado o gráfico, Figura 25, observa-se que é idêntico ao da Figura 24, portanto o programa deteta os pontos de inflexão classificando o solo como não homogéneo, como se apresenta ilustrado no Figura 26.

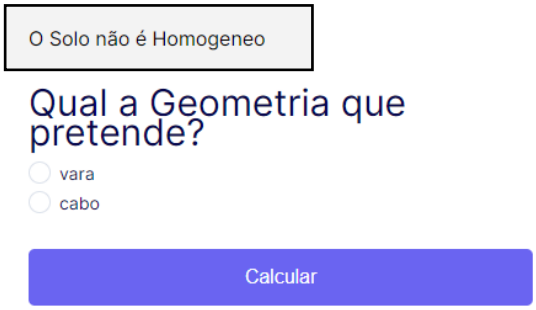


Figura 26-Classificação do solo

6.2. Cálculo da resistência da vara para um solo não homogéneo

Como ilustrado na Figura 27, selecionou-se a vara e definiu-se um comprimento para esta de 1,5 metros. Quando o utilizador clicar no botão "calcular" a aplicação através das fórmulas de Tagg irá calcular o comprimento necessário para que a resistência da vara seja no máximo de 20 ohms.

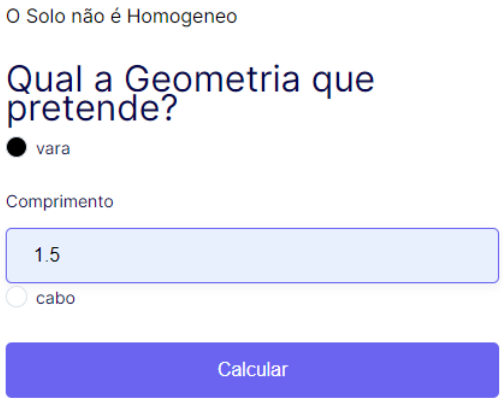


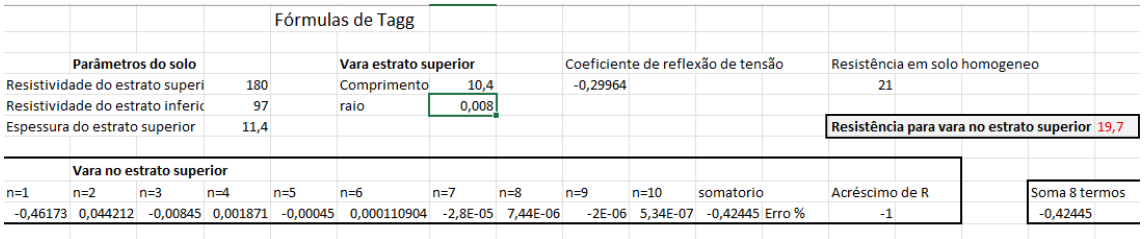
Figura 27-Cálculo do comprimento necessário da vara

Deste modo, a aplicação calculou um comprimento de 10,4 metros, Figura 28.

O comprimento de uma Vara é de 1.5 metros.
O comprimento necessário é de 16.3 metros
O estrato superior tem uma espessura de 11.4 metros.

Figura 28-"Output" do comprimento da vara

Para verificar se a aplicação realmente calculou o comprimento certo, usou-se o documento em Excel para garantir um resultado correto. Como se observa na Figura 29, para uma resistividade superior igual a 180 Ωm e uma resistividade inferior de 97 Ωm, obtemos o coeficiente de reflexão de tensão k= -0,3, e a espessura do estrato superior igual a 11,4, para a resistência da vara ser 20 ohms o comprimento da vara será de 10,4 metros.



Fórmulas de Tagg													
	Parâmetros do solo			**Vara estrato superior**			Coeficiente de reflexão de tensão		Resistência em solo homogeneo				
Resistividade do estrato superi	180			Comprimento	10,4		-0,29964		21				
Resistividade do estrato inferic	97			raio	0,008								
Espessura do estrato superior	11,4								**Resistência para vara no estrato superior**	19,7			
	Vara no estrato superior												
n=1	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10	somatorio	Acréscimo de R		Soma 8 termos
-0,46173	0,044212	-0,00845	0,001871	-0,00045	0,000110904	-2,8E-05	7,44E-06	-2E-06	5,34E-07	-0,42445	Erro %	-1	-0,42445

Figura 29-Cálculo do comprimento da vara em Excel

Ao comparar os dados obtidos na aplicação com os do Excel, observa-se um erro percentual de 1,5%, ilustrado no Tabela 4.

Tabela 4-Erro relativo vara no estrato superior

Comprimento vara(m)	**Resistência Excel(Ω)**	**Resistência Aplicação(Ω)**	**Erro(%)**
10,4	19,7	20	1,5

6.3. Cálculo da resistência de varas aproximadas para um solo homogéneo

Sabendo que o solo é homogéneo, o utilizador pode escolher o terreno ao qual vai aplicar o sistema de terra, desta forma escolher a tipologia do solo, como se mostra na Figura 30.

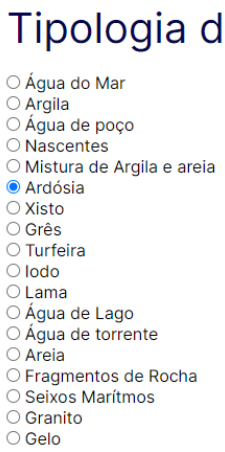


Figura 30-Escolha da tipologia do solo

Agora o utilizador tem de escolher a geometria a implementar, neste caso varas afastadas, com o espaçamento entre elas tem de ser superior ao comprimento das mesmas, Figura 31.

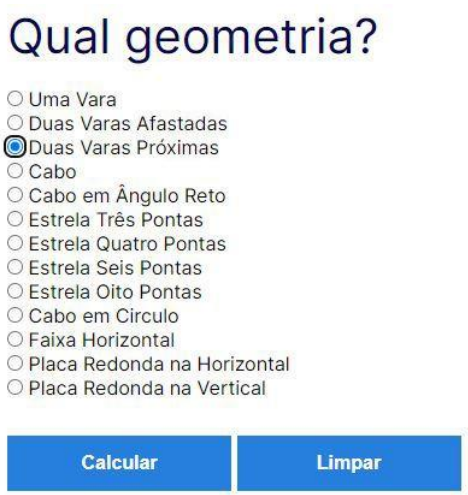


Figura 31-Geometria a implementar

Se o utilizador definir uma distância de 4 metros, por exemplo, a aplicação irá calcular o comprimento necessário das varas, obtendo então um comprimento de 4,1 metros, Figura 32.

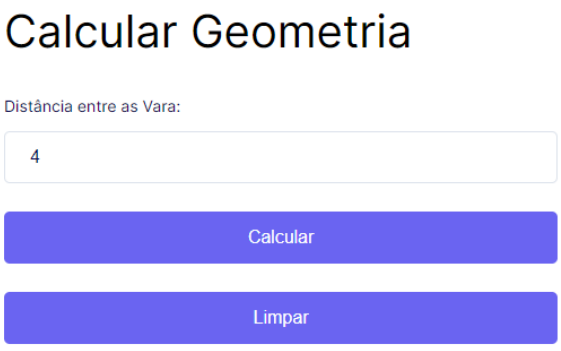
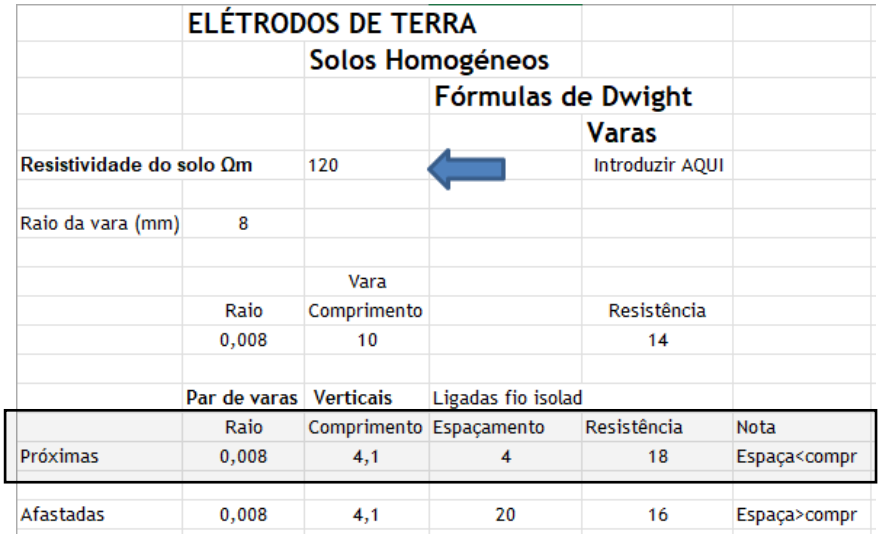


Figura 32-Cálculo do comprimento necessário

Recorrendo uma vez mais ao Excel fornecido pelo orientador, vai se conferir o resultado do comprimento, como se observa na Figura 33.



	ELÉTRODOS DE TERRA				
		Solos Homogéneos			
			Fórmulas de Dwight		
				Varas	
Resistividade do solo Ωm		120		Introduzir AQUI	
Raio da vara (mm)	8				
		Vara			
	Raio	Comprimento		Resistência	
	0,008	10		14	
	Par de varas	**Verticais**	Ligadas fio isolad		
	Raio	Comprimento	Espaçamento	Resistência	Nota
Próximas	0,008	4,1	4	18	Espaça<compr
Afastadas	0,008	4,1	20	16	Espaça>compr

Figura 33-Verificação do "Output" varas afastadas

Ao comparar os dados obtidos na aplicação com os do Excel, observa-se um erro percentual de 11,5%, ilustrado no Tabela 5.

Tabela 5-Erro relativo varas próximas

Comprimento vara(m)	**Resistência Excel(Ω)**	**Resistência Aplicação(Ω)**	**Erro(%)**
4,1	18	20	11,5

7. Conclusões

Após concluir o desenvolvimento deste projeto, fica evidente a importância de uma aplicação web que auxilia no planeamento de sistemas de terras. Através da micro-framework Flask em Python, foi possível criar uma ferramenta eficiente e de fácil utilização, proporcionando ao utilizador a capacidade de analisar, classificar e calcular informações relevantes relacionadas ao solo.

A funcionalidade de verificar a homogeneidade do solo permite ao utilizador tomar decisões mais racionais sobre os recursos a serem aplicados, com base na classificação obtida. Além disso, a escolha da geometria adequada ao solo estratificado permite uma abordagem mais precisa no planeamento de sistemas de terras.

A implementação do ciclo infinito e do método de Newton-Raphson para o cálculo da resistência da vara demonstrou ser uma abordagem eficaz, proporcionando resultados confiáveis e precisos para determinar o comprimento necessário do elétrodo escolhido.

Durante o desenvolvimento do projeto, realizaram-se testes detalhados e comparações de medições e dados provenientes de fontes fidedignas, garantindo assim a validação e confiabilidade da aplicação. A obtenção de um erros percentuais no intervalo[1,5% 11,5%] na comparação dos resultados com o Excel demonstra que o pior erro alcançada pela aplicação web é razoável.

Com base nos resultados e funcionalidades implementadas, conclui-se que o projeto foi bem-sucedido, pois alcançou o objetivo de fornecer uma ferramenta útil para o planeamento de sistemas de terras. A aplicação web apresenta uma interface intuitiva e de fácil navegação, permitindo ao utilizador interagir e obter informações relevantes de forma eficiente.

Contudo, reconhece-se que há sempre espaço para melhorias e expansão do projeto, tanto ao nível do backend como do frontend, a implementação de um sistema de login, será um exemplo de melhoria.

Em resumo, o projeto foi uma oportunidade enriquecedora para aplicar e aprender conhecimentos teóricos e práticos na área de sistemas de terras e desenvolvimento web. A aplicação desenvolvida proporciona uma solução valiosa para profissionais e entusiastas da

área, simplificando o processo de planeamento e contribuindo para um projeto mais eficiente, seguro e reduzindo custos desnecessários para a sua implementação.

Bibliografia

AppMaster. (s.d.). *As 10 melhores estruturas de back-end da Web em 2023 para desenvolvimento da Web*, pp. https://appmaster.io/pt/blog/10-melhores-estruturas-de-back-end-da-web.

César. (2020). *Análise Matemática.* Guarda: Politécnico da Guarda.

Clara, M. (2020). *Engenharia de software 1.* Guarda: Politécnico da Guarda.

Clara, M. (2020). *Engenharia de Software 2.* Guarda: Politécnico Guarda.

DelftStack. (s.d.). *Log natural em Python*, pp. https://www.delftstack.com/pt/howto/numpy/natural-log-python/.

Dwight, H. B. (December 1936). *Calculation of Resistances to Ground.* IEEE.

Filho, S. V. (2002). *Aterramento Elétrico.* São Paulo: Artliber Editora Ltda.

Foundation, P. S. (s.d.). *Funções matemáticas*, pp. https://docs.python.org/pt-br/3/library/math.html.

Frank, G. (1964). *Earth resistances.* London, George Newnes.

freeCodeCamp. (s.d.). *Python Switch Statement – Switch Case*, pp. https://www.freecodecamp.org/news/python-switch-statement-switch-case-example/.

GilcierWeb. (s.d.). *Formulário de contato com Python, Flask e envio de e-mail*, pp. https://gilcierweb.com.br/posts/formulario-de-contato-com-python-flask-e-envio-de-e-mail.

Kindermann, G. (1995). *Aterramento Elétrico.* Porto Alegre: Sagra-DC Luzzatto.

LTDA, T. T. (s.d.). *Django ou Flask*, pp. https://www.treinaweb.com.br/blog/django-ou-flask-eis-a-questao/.

Microsoft. (s.d.). *Início Rápido: criar um projeto do Python com base em um modelo no Visual Studio*, pp. https://learn.microsoft.com/pt-br/visualstudio/python/quickstart-02-python-in-visual-studio-project-from-template?view=vs-2022.

Microsoft. (s.d.). A network-related or instance-specific error occurred while establishing a connection to SQL Server. pp. https://learn.microsoft.com/en-us/troubleshoot/sql/database-engine/connect/network-related-or-instance-specific-error-occurred-while-establishing-connection.

Mozilla. (s.d.). *Document: querySelectorAll() method*, pp. https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll.

Portugal, C. e. (s.d.). *Rede de Terras.*

Prize, P. W. (2020). *International Review of Eletrical Engineering, Vol.15, N.2.*

Rubin. (2012).

Valente, F. P. (2018). *Análise Numérica.* Guarda: Politécnico da Guarda.

W3Schools. (s.d.). *HTML Tutorial*, p. https://www.w3schools.com/html/default.asp.

Anexos

1. Código

1.1. Função menu()

```
@app.route("/")
def menu():
    return render_template("MenuIncial.html")
```

1.2. Função supinf()

```
def supinf(array):
    resistividadeInf = min(array)
    resistividadeSup = max(array)
    #1.1 pois se fizermos os 10% do array[0] vamos obter o
valor extra desse aumento ou diminuição, depois somado ao
array0 vamos obter os 1.1, isto é 110%, o aumento ou
diminuição de 10%

    indice_max = array.index(resistividadeSup)
    indice_min = array.index(resistividadeInf)

    # Verificar se o ponto máximo tem aumento ou diminuição
de 10%
    if indice_max > 0 and (resistividadeSup <= 1.1 *
array[indice_max - 1] or resistividadeSup >= 0.9 *
array[indice_max - 1]):
        # Calcular a média entre o ponto máximo e seu
adjacente
        ponto_adjacente = array[indice_max - 1]
        media_maximo = (resistividadeSup + ponto_adjacente)
/ 2
    else:
        media_maximo = resistividadeSup

        # Verificar se o ponto mínimo tem aumento ou
diminuição de 10%
        if indice_min < len(array) - 1 and (
                resistividadeInf <= 1.1 * array[indice_min
+ 1] or resistividadeInf >= 0.9 * array[indice_min + 1]):
            # Calcular a média entre o ponto mínimo e seu
adjacente
            ponto_adjacente = array[indice_min + 1]
            media_minimo = (resistividadeInf +
```

```
ponto_adjacente) / 2
        else:
            media_minimo = resistividadeInf

    print("Resistividade Superior:", resistividadeSup)
    print("Resistividade Inferir:", resistividadeInf)

    return resistividadeSup, resistividadeInf
```

1.3. Função calculaSoloHomogenio()

```
@app.route("/verificarsolo", methods=['POST'])
def calculaSoloHomogeneo():
    global r, h
    ...
    #criar array para manipular mais facilmente as
coordenadas
    p = []
    r = []

    p1 = float(request.form.get('profundidade1'))#ir buscar
o ponto ao form da pagina
    p.append(p1)#adicionar o ponto ao array
    p2 = float(request.form.get('profundidade2'))
    p.append(p2)
    p3 = float(request.form.get('profundidade3'))
    p.append(p3)
    p4 = float(request.form.get('profundidade4'))
    p.append(p4)

    r1 = float(request.form.get('resistividade1'))
    r.append(r1)
    r2 = float(request.form.get('resistividade2'))
    r.append(r2)
    r3 = float(request.form.get('resistividade3'))
    r.append(r3)
    r4 = float(request.form.get('resistividade4'))
    r.append(r4)

    rmedia = (r1 + r2 + r3 + r4)/4
    print(f"Resistividade média {rmedia}")

    # calcular o declive dos pontos
    declive = np.gradient(r, p)
```

```
    # comparar para ver se realmente existe declives com um
aumento acima de 20%, que seja bastante considravel
    # inicialmente consideralas falsas
    d_acima = False
    d_abaixo = False
    ponto_diferenca = []  # guardar os pontos(abcissa) num
array para depois descobrir os pontos com esse aumento ou
diminuição

    # calcular e comparar
    for i in range(len(declive) - 1):
        diferenca = (declive[i + 1] - declive[i]) /
declive[i]
        # print("diferença: ", diferenca)
        if (diferenca > 0.33) and (diferenca < 0.9) or
(diferenca > 1):
            d_acima = True
            ponto_acima = (p[i])  # ir buscar o
ponto(abcissa) onde registou o aumento
            ponto_diferenca.append(ponto_acima)  # Guardar
esses pontos no array
            print("ponto para calcular o h:", ponto_acima)
            # passar esse ponto para a folha html
depois!!!!!!!
        elif (diferenca < -0.33) and (diferenca > -0.9) or
(diferenca < -1):
            d_abaixo = True
            ponto_acima = (p[i])  # ir buscar o
ponto(abcissa) onde registou o aumento
            ponto_diferenca.append(ponto_acima)  # Guardar
esses pontos no array
            print("ponto para calcular o h:", ponto_acima)


    # Encontrar os pontos de inflexão
    ponto_inflexao = []
    if (d_abaixo==True) or (d_acima==True):
        for i in range(1, len(p) - 1):
            if (declive[i] > 0 and declive[i + 1] < 0) or
(declive[i] < 0 and declive[i + 1] > 0):
                y = p[i]
                x = r[i]
                ponto_inflexao.append((x, y))

        coordenadasX = p.copy()
        coordenadasY = r.copy()
```

```
        # Remover pontos duplicados
        coordenadasX, indices = np.unique(coordenadasX,
return_index=True)
        coordenadasY = np.array(coordenadasY)[indices]

        # Interpolar os pontos para obter uma curva suave
        f = interp1d(coordenadasX, coordenadasY,
kind='cubic')
        p_interp = np.linspace(min(coordenadasX),
max(coordenadasX), 100)
        r_interp = f(p_interp)

        # Cálculo dos pontos de inflexão usando a segunda
derivada
        df_interp = np.gradient(np.gradient(r_interp,
p_interp), p_interp)

        tolerancia = 0.9
        # Cálculo das coordenadas dos pontos de inflexão
        pontos_inflexao = []
        for i in range(1, len(p_interp) - 1):
            if (df_interp[i] > 0 and df_interp[i + 1] < 0)
or (df_interp[i] < 0 and df_interp[i + 1] > 0):
                x = p_interp[i]
                y = f(x)
                # Verificar proximidade com outros pontos
de inflexão
                adicionar_ponto = True
                for ponto in pontos_inflexao:
                    if abs(x - ponto[0]) < tolerancia and
abs(y - ponto[1]) < tolerancia:
                        adicionar_ponto = False
                        break

                if adicionar_ponto:
                    pontos_inflexao.append((x, y))
        #print("Pontos inflexão:", pontos_inflexao)

    if (r1 <= 0.15*rmedia) and len(ponto_inflexao) == 0 and
d_acima==False and d_abaixo==False:
        print(r1)
        print(rmedia)
        print("O solo é Homogénio")
        # Interpolar os pontos para obter uma curva suave
```

```
        #modar a interpolação!!!!!!!!!!!!!
        f = interp1d(p, r, kind='cubic')
        p_interp = np.linspace(min(p), max(p), 100)
        r_interp = f(p_interp)

        # Plotar a curva suave e os pontos de inflexão
        plt.plot(p_interp, r_interp, '-')
        plt.plot(p, r, 'ro')
        plt.xlabel('profundidade')
        plt.ylabel('resistência')
        plt.title('Gráfico')
        plt.grid(True)
        plt.show()

        rmedio_s = str(rmedia)
        texto = "O Solo é Homogéneo"
        texto2 = f"A resistividade do Solo é de {rmedio_s}
ohm"

        return
render_template("SoverificadoHomogeneo.html", texto=texto,
texto2=texto2)  # Não há ponto de inflexão

    elif (len(ponto_inflexao) > 0) & (r1 >= 0.15*rmedia):

        # Imprimir os pontos de inflexão
        print("Pontos de inflexão:")
        for ponto in pontos_inflexao:
            y, x = ponto
            print(f"Coordenadas: ({y}, {x}")

        # calcular h - expessura do extrato superior
        h = ((2 / 3) * y)  # Confirmar coordenada!!!!------
-------
        print(f"h é igual {h}")

        supinf(r)

        # Plotar a curva interpolada e a sua derivada
        plt.plot(p_interp, r_interp, label='Curva
Interpolada')
        plt.plot(p_interp, df_interp, label='Derivada')
        plt.scatter(coordenadasX, coordenadasY,
color='red', label='Pontos Dados')
        plt.scatter(y, x, color='green', label='Pontos de
Inflexão')
```

```
        plt.xlabel('Profundidade')
        plt.ylabel('Resistividade')
        plt.title('Resistividade do Terreno')
        plt.legend()
        plt.grid(True)
        plt.show()

        texto = "O Solo não é Homogéneo"

        return render_template("ComprimentovaraCabo.html",
texto=texto)

    elif (len(ponto_inflexao) == 0) and (r1 > 0.15 *
rmedia) or (d_acima == True or d_abaixo == True):
        print("O solo não é homogénio")
        supinf(r)
        # ir buscar o ponto onde o declive aumentou ou
diminuio
        h = (2/3) * ponto_diferenca[0]
        print("valor h = ", h)
        coordenadasX = p.copy()
        coordenadasY = r.copy()

        # Remover pontos duplicados
        coordenadasX, indices = np.unique(coordenadasX,
return_index=True)
        coordenadasY = np.array(coordenadasY)[indices]

        # Interpolar os pontos para obter uma curva suave
        f = interp1d(coordenadasX, coordenadasY,
kind='cubic')
        p_interp = np.linspace(min(coordenadasX),
max(coordenadasX), 100)
        r_interp = f(p_interp)

        # Plotar a curva interpolada e a sua derivada
        plt.plot(p_interp, r_interp, '-')
        plt.scatter(coordenadasX, coordenadasY,
color='red', label='Pontos dados')
        plt.xlabel('Profundidade')
        plt.ylabel('Resistividade')
        plt.title('Resistividade do Terreno')
        plt.legend()
        plt.grid(True)
        plt.show()
```

```
        texto = "O Solo não é Homogéneo"
        return render_template("ComprimentovaraCabo.html",
texto=texto)  # Não há ponto de inflexão
```

1.4. Função cabo()

```
@app.route("/calculocabo", methods= ['GET' ,'POST'])
def cabo():
    global r, h
    if request.method == 'POST':
        raioExterno = request.form.get('raioExterno')
        raioExternoF = float(raioExterno)
        print("raio externo", raioExternoF)

        raioInterno = request.form.get('raioInterno')
        raioInternoF = float(raioInterno)
        print("raio Interno", raioInternoF)

        resistividadeCondutor =
request.form.get('resistividadecabo')
        resistividadeCondutorF =
float(resistividadeCondutor)
    print("Passou Aqui 2")
    a = math.pi * (raioExternoF**2 - raioInternoF**2)
    print("a: ", a)
    rhom =
(resistividadeCondutorF/a)*(1/np.log(raioExternoF/raioInter
noF))
    print("rhom: ", rhom)
    valor_l = calculoResistividadeCabo(rhom)
    return valor_l, h
```

1.5. Função calculoResistividadeCabo()

```
def calculoResistividadeCabo(rhom):
    print("Passei aqui 3 cabo")
    # Chamar a função supinf() para obter resistividade
superior e inferior
    rs, ri = supinf(r)
    # resistencia da vara para dois extratos diferentes
    xn = 2.0  # comprimento minimo da vara
    # declara a variável simbólica
    l = sy.Symbol('l')
```

```
    k = round((ri - rs) / (ri + rs), 1)
    print("K = ", k)
    n = 1
    # resistencia da vara
    rv = 20
    # Parte 1 da equação de Tagg
    parte1 = rhom
    parte2 = ((2 * rs) / (math.pi * l))
    # Parte 2 da equação de Tagg (soma finita)
    parte3 = k**n * np.log((1 + (math.sqrt(((2 * n *
h)/(1)) + 1)))/(2*n*(h/1))) + ((2*n*h)/(1)) -
math.sqrt(((2*n*h)/(1)) + 1)
    print("parte3: ", parte3)

    termo = k**n * np.log((1 + (math.sqrt(((2 * n * h)/(1))
+ 1)))/(2*n*(h/1))) + ((2*n*h)/(1)) -
math.sqrt(((2*n*h)/(1)) + 1)
    tolerancia_soma = parte3 * 0.05
    print("Passou aqui cabo")

    while (n < 9):
        parte3 += termo
        n += 1
        termo = k**n * np.log((1 + (math.sqrt(((2 * n *
h)/(1)) + 1)))/(2*n*(h/1))) + ((2*n*h)/(1)) -
math.sqrt(((2*n*h)/(1)) + 1)
        print(n)
        tolerancia_soma = parte3 * 0.01

    # Equação de Tagg
    equacao = (parte1 + parte2 * parte3) - rv
    print("Equação : ", equacao)

    # Derivada da função em relação a l
    derivada = sy.diff(equacao, l)
    print(derivada)

    # Configuração do método de Newton-Raphson
    tolerancia = 1e-6
    max_iteracoes = 7
    l_aproximado = 2.0  # Valor inicial de l

    # Converter expressões para funções numéricas
    equacao_numerica = sy.lambdify(l, equacao)
    derivada_numerica = sy.lambdify(l, derivada)
```

```
    for _ in range(1, max_iteracoes):
        l_anterior = l_aproximado
        l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
        print("L_aproximado", l_aproximado)
        if abs(l_aproximado - l_anterior) < tolerancia:
            break
    valor_l = round(l_aproximado)
    print("O comprimento necessário para o cabo é de",
round(valor_l))
    return valor_l
```

1.6. Função calculoResistividade1()

```
@app.route("/calculo1", methods=['GET', 'POST'])
def calculoResistividade1():
    print("Passou Aqui 1")
    global r, h
    print(h)
    print(r)
    escolha = request.form['tipo']
    print(escolha)
    if escolha == 'vara':
        valor_l, h, comprimentoF =
calculaResistividadeVara(r, h)
        valor_l_s = str(round(valor_l, 1)) #é necessátio
converte-los para string para depois redirecionar o texto
para a minha página
        h_s = str(round(h, 1))
        cumprimentoF_s = str(comprimentoF)
        texto = f"O comprimento de uma Vara é de
{cumprimentoF_s} metros."
        texto2 = f"O comprimento necessário é de
{valor_l_s} metros"
        texto3 = f"O estrato superior tem uma espessura de
{h_s} metros."
        return render_template("VerificarSolo.html",
texto=texto, texto2=texto2, texto3=texto3)
    elif escolha == 'cabo':
        valor_l, h = cabo()
        valor_l_s = str(valor_l)
        h_s = str(round(h, 1))
        texto2 = f"O comprimento necessário para o cabo é
de {valor_l_s} metros"
```

```
        texto3 = f"O estrato superior tem uma espessura de
{h_s} metros."
        return render_template("VerificarSolo.html",
texto2=texto2, texto3=texto3)
```

1.7. Função calculoResistividadeVara()

```
def calculaResistividadeVara(r, h):
    print("Passei aqui 3")
    if request.method == 'POST':
        comprimento = request.form.get('distanciaVaras')
        print(comprimento)
        comprimentoF = float(comprimento)
    print("Distancia F:", comprimentoF)
    # Chamar a função supinf() para obter resistividade
superior e inferior
    rs, ri = supinf(r)
    #resistencia da vara para dois extratos diferentes
    xn = 2.0 #comprimento minimo da vara
    # declara a variável simbólica
    l = sy.Symbol('l')
    k = round((ri-rs)/(ri+rs), 1)
    print("K = ", k)
    a = comprimentoF
    n = 1
    # resistencia da vara
    rv = 20

    # Parte 1 da equação de Tagg
    parte1 = (rs / (2 * math.pi * l)) * ((1 + k) / (1 - k +
2 * k * (h / l)))
    parte2 = (np.log(2) + sy.log(l) - np.log(a))
    # Parte 3 da equação de Tagg (soma finita)
    parte3 = (k ** n) * (np.log(2) + np.log(n) + np.log(1 +
1 / (2 * n * h))) / ((2 * n - 2) * h + 1)
    print("parte3: ", parte3)

    termo = (k ** n) * (sy.log(2) + sy.log(n) + sy.log(1 +
1 / (2 * n * h))) / ((2 * n - 2) * h + 1)

    tolerancia_soma = parte3 * 0.01
    print("Passou aqui")
    #Soma infinita
    while (n < 9):
        parte3 += termo
```

```
        n += 1
        termo = ((k ** n)/2) * (np.log(n * h + 1) -
np.log(n * h - 1))
        print(n)
        tolerancia_soma = parte3 * 0.01

    #Equação de Tagg
    equacao = parte1 * (parte2 + parte3) - rv
    print("Equação : ", equacao)

    #Derivada da função em relação a l
    derivada = sy.diff(equacao, l)
    print(derivada)

    #Configuração do método de Newton-Raphson
    tolerancia = 1e-6
    max_iteracoes = 3
    l_aproximado = 2.0  # Valor inicial de l

    #Converter expressões para funções numéricas
    equacao_numerica = sy.lambdify(l, equacao)
    derivada_numerica = sy.lambdify(l, derivada)

    for _ in range(1, max_iteracoes):
        l_anterior = l_aproximado
        l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
abs(derivada_numerica(l_anterior)))
        print("L_aproximado", l_aproximado)
        if abs(l_aproximado - l_anterior) < tolerancia:
            break
    valor_l = l_aproximado
    print("O comprimento necessário para a vara é de",
round(valor_l, 1))
    return valor_l, h, comprimentoF
```

1.8. Função homogénio()

```
@app.route("/homogénio")
def homogenio():
    return render_template("SoloHomogeneo.html",
valor_l_s=valor_l_s)
```

1.9. Função GeometriaVara2()

```
@app.route("/GeometriaVara2", methods=['POST'])
def GeometriaVara2():
    if request.method == 'POST':
        d_varas = float(request.form.get('distancia'))

    global valor_l_s, rs
    r_s = str(rs)
    print("resistividade do solo: ", r_s)
    d = 0.08
    xn = 1.5
    # resistência do solo
    r = 20
    # verificar
    v = sy.Symbol('v')
    parte1 = (rs / (4 * math.pi * v))
    print("parte1 ", parte1)
    parte2 = sy.log((4 * v) / d) + sy.log((4 * v) / d) - 2
+ (d / (2 * v)) - (d ** 2 / 16 * v ** 2) + (
                d ** 2 / 512 * v ** 4)
    print("parte2 ", parte2)

    equacao = parte1 * parte2 - r

    # derivada da função
    derivada = sy.diff(equacao, v)
    print("derivada : ", derivada)

    # método de Newton-Raphson
    tolerancia = 1e-6
    max_iteracoes = 5
    l_aproximado = xn

    # Converter expressões para funções numéricas
    equacao_numerica = sy.lambdify(v, equacao)
    derivada_numerica = sy.lambdify(v, derivada)

    for _ in range(max_iteracoes):
        l_anterior = l_aproximado
        l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
        if abs(l_aproximado - l_anterior) < tolerancia:
            break
```

```
    valor_l = l_aproximado
    print("Comprimento necessário:", valor_l)
    valor_l_s = str(round(valor_l, 1))
    print("Valor em string: ", valor_l)
    texto = f"O comprimento necessário é de {valor_l_s}
metros."

    return render_template("SoloHomogeneo.html",
texto=texto)
```

1.10. Função GeometriaVaras()

```
@app.route("/GeometriaVaras", methods=['POST'])
def GeometriaVaras():
    if request.method == 'POST':
        d_varas = float(request.form.get('distancia'))

    global valor_l_s, rs
    r_s = str(rs)
    print("resistividade do solo: ", r_s)
    d = 0.08
    xn = 1.5
    # resistencia do solo
    r = 20
    # verificar
    v = sy.Symbol('v')
    parte1 = (rs/ (4 * math.pi * v))
    print("parte1 ", parte1)
    parte2 = (sy.log((4 * v) / d) - 1) + (rs / (4 * math.pi
* d_varas))
    print("parte2 ", parte2)
    parte4 = (1 - (v ** 2 / (3 * (d_varas ** 2))) + ((2 *
(v ** 4)) / (5 * (d_varas ** 4)))) - r
    print("parte4 ", parte4)

    equacao = parte1 * parte2 * parte4

    # derivada da função
    derivada = sy.diff(equacao, v)
    print("derivada : ", derivada)

    # método de Newton-Raphson
    tolerancia = 1e-6
    max_iteracoes = 4
    l_aproximado = xn
```

```
    # Converter expressões para funções numéricas
    equacao_numerica = sy.lambdify(v, equacao)
    derivada_numerica = sy.lambdify(v, derivada)

    for _ in range(max_iteracoes):
        l_anterior = l_aproximado
        l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
        if abs(l_aproximado - l_anterior) < tolerancia:
            break

    valor_l = l_aproximado
    print("Comprimento necessário:", valor_l)
    valor_l_s = str(round(valor_l, 1))
    print("Valor em string: ", valor_l)
    texto = f"O comprimento necessário é de {valor_l_s}
metros."

    return render_template("SoloHomogeneo.html",
texto=texto)
```

1.11. Função GeometriaCirculo()

```
@app.route("/GeometriaCirculo", methods=['POST'])
def GeometriaCirculo():
    if request.method == 'POST':
        diametroCabo =
float(request.form.get('diametroCabo'))

    global valor_l_s, rs
    rmedio_s = str(rs)
    print("resistividade do solo: ", rmedio_s)
    d = 0.08
    xn = 1.5
    # resistencia do solo
    r = 20
    diametroCirculo = sy.Symbol('diametroCirculo')
    parte1 = (rs / ((2 * math.pi) ** 2 * diametroCirculo))
    print("parte1 ", parte1)
    parte2 = np.log(8) + sy.log(diametroCirculo) -
np.log(diametroCabo) + np.log(4) + sy.log(diametroCirculo)
- np.log(
        d)
```

```
    print("parte2 ", parte2)

    equacao = (parte1 * parte2) - r
    print("Equação", equacao)
    # derivada da função
    derivada = sy.diff(equacao, diametroCirculo)
    print("derivada : ", derivada)

    # método de Newton-Raphson
    tolerancia = 1e-6
    max_iteracoes = 3
    l_aproximado = xn

    # Converter expressões para funções numéricas
    equacao_numerica = sy.lambdify(diametroCirculo,
equacao)
    derivada_numerica = sy.lambdify(diametroCirculo,
derivada)

    for _ in range(max_iteracoes):
        l_anterior = l_aproximado
        l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
        if abs(l_aproximado - l_anterior) < tolerancia:
            break

    valor_l = l_aproximado
    print("Comprimento necessário:", valor_l)
    valor_l_s = str(valor_l)
    print("Valor em string: ", valor_l)
    texto = f"O comprimento necessário é de {valor_l_s}
metros."

    return render_template("SoloHomogeneo.html",
texto=texto)
```

1.12. Função calculoResistividadeSoloHomogenio()

```
@app.route("/homogénio", methods=['GET', 'POST'])
def calculoResistividadeSoloHomogenio():
    global  valor_l_s, rs
print("passou aqui 1")
d = 0.08
xn = 1.5
```

```
#resistencia do solo
r = 20

#verificar
v = sy.Symbol('v')
tipoSolo = request.form['solo']
cursor.execute(f"select ResistividadeTipica from Solo where
TipoSolo like (?)", tipoSolo)
print(tipoSolo)
rs = cursor.fetchone()[0]  # ir buscar á BD  a
resistividade do Solo selecionado
print(rs)
rs_s = str(rs)
formula = request.form['formula']
# distancia entre as varas

#verificar o tipo de geometria escolhida pelo utilizador
if formula == 'vara':
    equacao = (rs / (4 * math.pi * v)) * sy.log((2 * v) /
d) - r

    # derivada da função
    derivada = sy.diff(equacao, v)
    print("derivada : ", derivada)

    # método de Newton-Raphson
    tolerancia = 1e-6
    max_iteracoes = 4
    l_aproximado = xn

    # Converter expressões para funções numéricas
    equacao_numerica = sy.lambdify(v, equacao)
    derivada_numerica = sy.lambdify(v, derivada)

    for _ in range(max_iteracoes):
        l_anterior = l_aproximado
        l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
        if abs(l_aproximado - l_anterior) < tolerancia:
            break

    valor_l = l_aproximado
```

```
    print("Comprimento necessário:", valor_l)
    valor_l_s = str(valor_l)
    print("Valor em string: ", valor_l)
    return valor_l_s

elif formula == 'varas':
    return render_template("calculoGeometriaVaras.html")

elif formula == 'varas2':
    return render_template("calculoGeometriaVaras2.html")

elif formula == 'cabo':

    parte1 = (rs/(4 * math.pi * v))
    print("parte1 ", parte1)
    parte2 = sy.log((4 * v)/d) + sy.log((4 * v)/d) - 2 +
(d/(2 * v)) - (d**2/16 * v**2) + (d**2/512 * v**4)
    print("parte2 ", parte2)

    equacao = parte1 * parte2 - r

    # derivada da função
    derivada = sy.diff(equacao, v)
    print("derivada : ", derivada)

    # método de Newton-Raphson
    tolerancia = 1e-6
    max_iteracoes = 5
    l_aproximado = xn

    # Converter expressões para funções numéricas
    equacao_numerica = sy.lambdify(v, equacao)
    derivada_numerica = sy.lambdify(v, derivada)

    for _ in range(max_iteracoes):
        l_anterior = l_aproximado
        l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
        if abs(l_aproximado - l_anterior) < tolerancia:
            break

    valor_l = l_aproximado
```

```
    print("Cumprimento necessário:", valor_l)
    valor_l_s = str(valor_l)
    print("Valor em string: ", valor_l)
    return valor_l_s

elif formula == 'caboReto':

    parte1 = (rs/(4 * math.pi * v))
    print("parte1 ", parte1)
    parte2 = sy.log((2 * v)/d) + sy.log((2 * v)/d) - 0.2373
+ 0.2146 * (d/(v)) + 0.1035 * (d**2/v**2) - 0.0424 *
(d**4/v**4)
    print("parte2 ", parte2)

    equacao = parte1 * parte2 - r

    # derivada da função
    derivada = sy.diff(equacao, v)
    print("derivada : ", derivada)

    # método de Newton-Raphson
    tolerancia = 1e-6
    max_iteracoes = 3
    l_aproximado = xn

    # Converter expressões para funções numéricas
    equacao_numerica = sy.lambdify(v, equacao)
    derivada_numerica = sy.lambdify(v, derivada)

    for _ in range(max_iteracoes):
        l_anterior = l_aproximado
        l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
        if abs(l_aproximado - l_anterior) < tolerancia:
            break

    valor_l = l_aproximado
    print("Comprimento necessário:", valor_l)
    valor_l_s = str(valor_l)
    print("Valor em string: ", valor_l)
    return valor_l_s
```

```
elif formula == 'tres_pontas':

    parte1 = (rs/(6 * math.pi * v))
    print("parte1 ", parte1)
    parte2 = sy.log((2 * v)/d) + sy.log((2 * v)/d) + 1.071
- 0.2009 * (d/(v)) + 0.238 * (d**2/v**2) - 0.054 *
(d**4/v**4)
    print("parte2 ", parte2)

    equacao = parte1 * parte2 - r

    # derivada da função
    derivada = sy.diff(equacao, v)
    print("derivada : ", derivada)

    # método de Newton-Raphson
    tolerancia = 1e-6
    max_iteracoes = 3
    l_aproximado = xn

    # Converter expressões para funções numéricas
    equacao_numerica = sy.lambdify(v, equacao)
    derivada_numerica = sy.lambdify(v, derivada)

    for _ in range(max_iteracoes):
        l_anterior = l_aproximado
        l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
        if abs(l_aproximado - l_anterior) < tolerancia:
            break

    valor_l = l_aproximado
    print("Comprimento necessário:", valor_l)
    valor_l_s = str(valor_l)
    print("Valor em string: ", valor_l)
    return valor_l_s

elif formula == '4_pontas':

    parte1 = (rs/(8 * math.pi * v))
    print("parte1 ", parte1)
    parte2 = sy.log((2 * v)/d) + sy.log((2 * v)/d) + 2.912
```

```
- 1.071 * (d/(v)) + 0.645 * (d**2/v**2) - 0.145 *
(d**4/v**4)
    print("parte2 ", parte2)

    equacao = parte1 * parte2 - r

    # derivada da função
    derivada = sy.diff(equacao, v)
    print("derivada : ", derivada)

    # método de Newton-Raphson
    tolerancia = 1e-6
    max_iteracoes = 3
    l_aproximado = xn

    # Converter expressões para funções numéricas
    equacao_numerica = sy.lambdify(v, equacao)
    derivada_numerica = sy.lambdify(v, derivada)

    for _ in range(max_iteracoes):
        l_anterior = l_aproximado
        l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
        if abs(l_aproximado - l_anterior) < tolerancia:
            break

    valor_l = l_aproximado
    print("Comprimento necessário:", valor_l)
    valor_l_s = str(valor_l)
    print("Valor em string: ", valor_l)
    return valor_l_s

elif formula == '6_pontas':

    parte1 = (rs/(12 * math.pi * v))
    print("parte1 ", parte1)
    parte2 = sy.log((2 * v)/d) + sy.log((2 * v)/d) + 6.851
- 3.128 * (d/(v)) + 1.758 * (d**2/v**2) - 0.490 *
(d**4/v**4)
    print("parte2 ", parte2)

    equacao = parte1 * parte2 - r
```

```
    # derivada da função
    derivada = sy.diff(equacao, v)
    print("derivada : ", derivada)

    # método de Newton-Raphson
    tolerancia = 1e-6
    max_iteracoes = 3
    l_aproximado = xn

    # Converter expressões para funções numéricas
    equacao_numerica = sy.lambdify(v, equacao)
    derivada_numerica = sy.lambdify(v, derivada)

    for _ in range(max_iteracoes):
        l_anterior = l_aproximado
        l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
        if abs(l_aproximado - l_anterior) < tolerancia:
            break

    valor_l = l_aproximado
    print("Comprimento necessário:", valor_l)
    valor_l_s = str(valor_l)
    print("Valor em string: ", valor_l)
    return valor_l_s

elif formula == '8_pontas':

    parte1 = (rs/(16 * math.pi * v))
    print("parte1 ", parte1)
    parte2 = sy.log((2 * v)/d) + sy.log((2 * v)/d) + 10.98
- 5.51 * (d/(v)) + 3.26 * (d**2/v**2) - 1.17 * (d**4/v**4)
    print("parte2 ", parte2)

    equacao = parte1 * parte2 - r

    # derivada da função
    derivada = sy.diff(equacao, v)
    print("derivada : ", derivada)

    # método de Newton-Raphson
```

```
    tolerancia = 1e-6
    max_iteracoes = 3
    l_aproximado = xn

    # Converter expressões para funções numéricas
    equacao_numerica = sy.lambdify(v, equacao)
    derivada_numerica = sy.lambdify(v, derivada)

    for _ in range(max_iteracoes):
        l_anterior = l_aproximado
        l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
        if abs(l_aproximado - l_anterior) < tolerancia:
            break

    valor_l = l_aproximado
    print("Cumprimento necessário:", valor_l)
    valor_l_s = str(valor_l)
    print("Valor em string: ", valor_l)
    return valor_l_s

elif formula == 'circulo':
    return  render_template("calculoGeometriaCirculo.html")

texto = f"O comprimento necessário para a {formula} é de
{valor_l_s} metros"
texto2 = f"A resistividade do {tipoSolo} é de {rs} ohm"

return render_template("calculoGeometriaVaras2.html",
texto=texto, texto2=texto2, valor_l_s=valor_l_s)
```

1.13. Função soloHomogeneo()

```
@app.route("/SoloHomogeneo", methods=['POST'])
def soloHomogenio():
    global valor_l_s, rs
    formula = request.form['formula']
    d = 0.08
    xn = 1.5
    # resistencia do solo
    r = 20
```

```
    # verificar
    v = sy.Symbol('v')

    # verificar o tipo de geometria escolhida pelo
utilizador
    if formula == 'vara':
        equacao = (rs / (4 * math.pi * v)) * sy.log((2 * v)
/ d) - r

        # derivada da função
        derivada = sy.diff(equacao, v)
        print("derivada : ", derivada)

        # método de Newton-Raphson
        tolerancia = 1e-6
        max_iteracoes = 4
        l_aproximado = xn

        # Converter expressões para funções numéricas
        equacao_numerica = sy.lambdify(v, equacao)
        derivada_numerica = sy.lambdify(v, derivada)

        for _ in range(max_iteracoes):
            l_anterior = l_aproximado
            l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
            if abs(l_aproximado - l_anterior) < tolerancia:
                break

        valor_l = l_aproximado
        print("Comprimento necessário:", valor_l)
        valor_l_s = str(valor_l)
        print("Valor em string: ", valor_l)
        return valor_l_s

    elif formula == 'varas':
        return
render_template("calculoGeometriaVaras.html")

    elif formula == 'varas2':
        return
render_template("calculoGeometriaVaras2.html")

    elif formula == 'cabo':
```

```
        parte1 = (rs / (4 * math.pi * v))
        print("parte1 ", parte1)
        parte2 = sy.log((4 * v) / d) + sy.log((4 * v) / d)
- 2 + (d / (2 * v)) - (d ** 2 / 16 * v ** 2) + (
                    d ** 2 / 512 * v ** 4)
        print("parte2 ", parte2)

        equacao = parte1 * parte2 - r

        # derivada da função
        derivada = sy.diff(equacao, v)
        print("derivada : ", derivada)

        # método de Newton-Raphson
        tolerancia = 1e-6
        max_iteracoes = 5
        l_aproximado = xn

        # Converter expressões para funções numéricas
        equacao_numerica = sy.lambdify(v, equacao)
        derivada_numerica = sy.lambdify(v, derivada)

        for _ in range(max_iteracoes):
            l_anterior = l_aproximado
            l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
            if abs(l_aproximado - l_anterior) < tolerancia:
                break

        valor_l = l_aproximado
        print("Comprimento necessário:", valor_l)
        valor_l_s = str(valor_l)
        print("Valor em string: ", valor_l)
        return valor_l_s

    elif formula == 'caboReto':

        parte1 = (rs / (4 * math.pi * v))
        print("parte1 ", parte1)
        parte2 = sy.log((2 * v) / d) + sy.log((2 * v) / d)
- 0.2373 + 0.2146 * (d / (v)) + 0.1035 * (
                    d ** 2 / v ** 2) - 0.0424 * (d ** 4 / v
** 4)
        print("parte2 ", parte2)
```

```
        equacao = parte1 * parte2 - r

        # derivada da função
        derivada = sy.diff(equacao, v)
        print("derivada : ", derivada)

        # método de Newton-Raphson
        tolerancia = 1e-6
        max_iteracoes = 3
        l_aproximado = xn

        # Converter expressões para funções numéricas
        equacao_numerica = sy.lambdify(v, equacao)
        derivada_numerica = sy.lambdify(v, derivada)

        for _ in range(max_iteracoes):
            l_anterior = l_aproximado
            l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
            if abs(l_aproximado - l_anterior) < tolerancia:
                break

        valor_l = l_aproximado
        print("Comprimento necessário:", valor_l)
        valor_l_s = str(valor_l)
        print("Valor em string: ", valor_l)
        return valor_l_s

    elif formula == 'tres_pontas':

        parte1 = (rs / (6 * math.pi * v))
        print("parte1 ", parte1)
        parte2 = sy.log((2 * v) / d) + sy.log((2 * v) / d)
+ 1.071 - 0.2009 * (d / (v)) + 0.238 * (
                    d ** 2 / v ** 2) - 0.054 * (d ** 4 / v
** 4)
        print("parte2 ", parte2)

        equacao = parte1 * parte2 - r

        # derivada da função
        derivada = sy.diff(equacao, v)
        print("derivada : ", derivada)
```

```
        # método de Newton-Raphson
        tolerancia = 1e-6
        max_iteracoes = 3
        l_aproximado = xn

        # Converter expressões para funções numéricas
        equacao_numerica = sy.lambdify(v, equacao)
        derivada_numerica = sy.lambdify(v, derivada)

        for _ in range(max_iteracoes):
            l_anterior = l_aproximado
            l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
            if abs(l_aproximado - l_anterior) < tolerancia:
                break

        valor_l = l_aproximado
        print("Comprimento necessário:", valor_l)
        valor_l_s = str(valor_l)
        print("Valor em string: ", valor_l)
        return valor_l_s

    elif formula == '4_pontas':

        parte1 = (rs / (8 * math.pi * v))
        print("parte1 ", parte1)
        parte2 = sy.log((2 * v) / d) + sy.log((2 * v) / d)
+ 2.912 - 1.071 * (d / (v)) + 0.645 * (
                    d ** 2 / v ** 2) - 0.145 * (d ** 4 / v
** 4)
        print("parte2 ", parte2)

        equacao = parte1 * parte2 - r

        # derivada da função
        derivada = sy.diff(equacao, v)
        print("derivada : ", derivada)

        # método de Newton-Raphson
        tolerancia = 1e-6
        max_iteracoes = 3
        l_aproximado = xn

        # Converter expressões para funções numéricas
        equacao_numerica = sy.lambdify(v, equacao)
```

```
        derivada_numerica = sy.lambdify(v, derivada)

        for _ in range(max_iteracoes):
            l_anterior = l_aproximado
            l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
            if abs(l_aproximado - l_anterior) < tolerancia:
                break

        valor_l = l_aproximado
        print("Comprimento necessário:", valor_l)
        valor_l_s = str(valor_l)
        print("Valor em string: ", valor_l)
        return valor_l_s

    elif formula == '6_pontas':

        parte1 = (rs / (12 * math.pi * v))
        print("parte1 ", parte1)
        parte2 = sy.log((2 * v) / d) + sy.log((2 * v) / d)
+ 6.851 - 3.128 * (d / (v)) + 1.758 * (
                    d ** 2 / v ** 2) - 0.490 * (d ** 4 / v
** 4)
        print("parte2 ", parte2)

        equacao = parte1 * parte2 - r

        # derivada da função
        derivada = sy.diff(equacao, v)
        print("derivada : ", derivada)

        # método de Newton-Raphson
        tolerancia = 1e-6
        max_iteracoes = 3
        l_aproximado = xn

        # Converter expressões para funções numéricas
        equacao_numerica = sy.lambdify(v, equacao)
        derivada_numerica = sy.lambdify(v, derivada)

        for _ in range(max_iteracoes):
            l_anterior = l_aproximado
            l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
```

```
            if abs(l_aproximado - l_anterior) < tolerancia:
                break

        valor_l = l_aproximado
        print("Comprimento necessário:", valor_l)
        valor_l_s = str(valor_l)
        print("Valor em string: ", valor_l)
        return valor_l_s

    elif formula == '8_pontas':

        parte1 = (rs / (16 * math.pi * v))
        print("parte1 ", parte1)
        parte2 = sy.log((2 * v) / d) + sy.log((2 * v) / d)
+ 10.98 - 5.51 * (d / (v)) + 3.26 * (
                    d ** 2 / v ** 2) - 1.17 * (d ** 4 / v
** 4)
        print("parte2 ", parte2)

        equacao = parte1 * parte2 - r

        # derivada da função
        derivada = sy.diff(equacao, v)
        print("derivada : ", derivada)

        # método de Newton-Raphson
        tolerancia = 1e-6
        max_iteracoes = 3
        l_aproximado = xn

        # Converter expressões para funções numéricas
        equacao_numerica = sy.lambdify(v, equacao)
        derivada_numerica = sy.lambdify(v, derivada)

        for _ in range(max_iteracoes):
            l_anterior = l_aproximado
            l_aproximado = l_anterior -
(equacao_numerica(l_anterior) /
derivada_numerica(l_anterior))
            if abs(l_aproximado - l_anterior) < tolerancia:
                break

        valor_l = l_aproximado
        print("Comprimento necessário:", valor_l)
        valor_l_s = str(valor_l)
        print("Valor em string: ", valor_l)
```

```
        return valor_l_s

    elif formula == 'circulo':
        return
render_template("calculoGeometriaCirculo.html")

    texto = f"O comprimento necessário para a {formula} é
de {valor_l_s} metros"

    return render_template("calculoGeometriaVaras2.html",
texto=texto, valor_l_s=valor_l_s)
```

1.14. Função menu()

```
#criar a homepage
#route -> link da página/caminho depois do dominio
xxxx.com/utilizador
#nome do site é app, route definir a página
@app.route("/")
#função -> o que quero fazer na homepage
def menu():
    return render_template("MenuIncial.html")#rodar site
```

1.15. Função Tabela()

```
2. @app.route("/verTabela")
   def tabela():
       cursor.execute("Select TipoSolo,
   ResistividadeTipica, LimiteNormaisMin, LimiteNormalMax
   From solo")
       tabela = cursor.fetchall()
       return render_template("tabela.html", tabela=tabela)
```

2. Menu Inicial

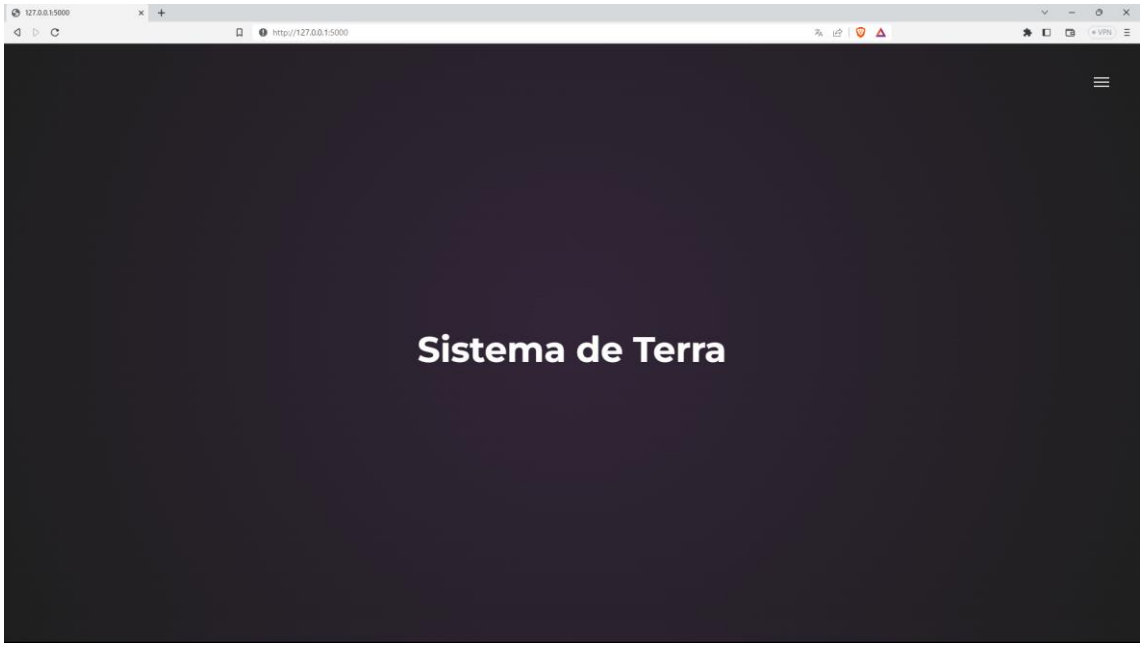


Figura 34-Menu Inicial 1

Figura 35-Menu Inicial 2

3. Visualizar Tabela

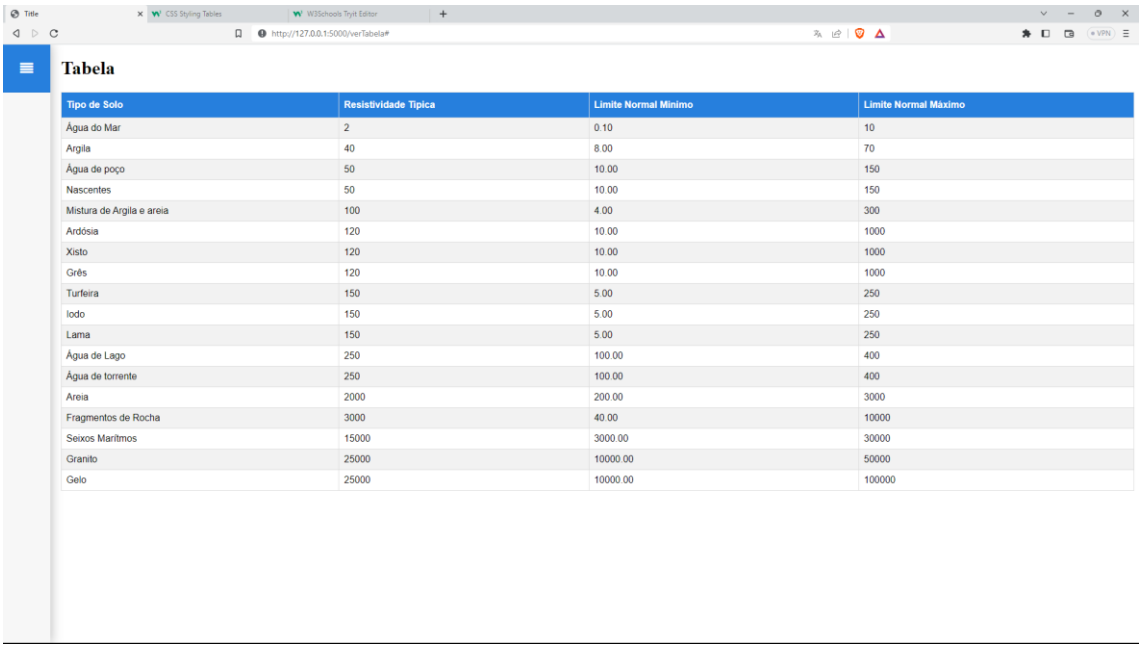


Figura 36-Interface da Tabela

4. SQL

4.1. Base de Dados

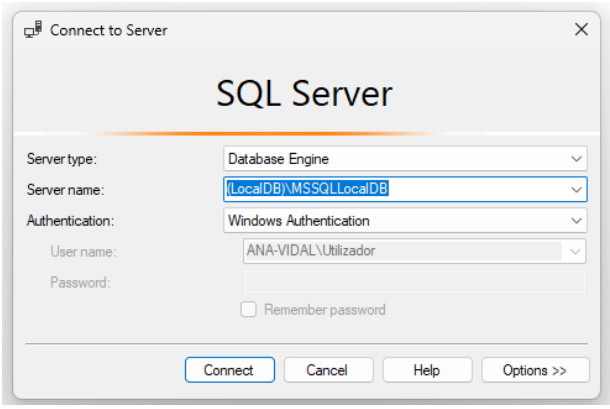


Figura 37-Conectar com o servidor

4.2. Script da Tabela Solo

```
use pythonsiteSQL

create table Solo(
	SoloId int,
	TipoSolo varchar(30),
	ResistividadeTipica int,
	LimiteNormaisMin decimal(9,2),
	LimiteNormalMax int
)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(1, 'Água do Mar', 2, 0.10, 10)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(2, 'Argila', 40, 8, 70)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(3,'Água de poço', 50, 10, 150)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(4, 'Nascentes', 50, 10, 150)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(5, 'Mistura de Argila e areia', 100, 4, 300)
```

```
insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(6, 'Ardósia', 120, 10, 1000)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(7, 'Xisto', 120, 10, 1000)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(8, 'Grês', 120, 10, 1000)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(9, 'Turfeira', 150, 5, 250)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(10, 'Iodo', 150, 5, 250)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(11, 'Lama', 150, 5, 250)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(12, 'Água de Lago', 250, 100, 400)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(13, 'Água de torrente', 250, 100, 400)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(14, 'Areia', 2000, 200, 3000)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(15, 'Fragmentos de Rocha', 3000, 40, 10000)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(16, 'Seixos Marítmos', 15000, 3000, 30000)

insert into Solo(SoloId, TipoSolo, ResistividadeTipica, LimiteNormaisMin,
LimiteNormalMax)
values(17, 'Granito', 25000, 10000, 50000)
```

5. Biblioteca pyodbc

```
(venv) PS C:\Users\Utilizador\PycharmProjects\pythonProject> pip install pyodbc
Collecting pyodbc
  Downloading pyodbc-4.0.39-cp39-cp39-win_amd64.whl (69 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 69.8/69.8 kB 1.9 MB/s eta 0:00:00
Installing collected packages: pyodbc
Successfully installed pyodbc-4.0.39
(venv) PS C:\Users\Utilizador\PycharmProjects\pythonProject>
```

Figura 38-Instalar pyodbs

6. Flask instalação

```
Terminal   Local ×  +  ⌄                                                    ⋮  —
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

(venv) PS C:\Users\Utilizador\PycharmProjects\pythonProject> pip install flask
Collecting flask
  Using cached Flask-2.3.2-py3-none-any.whl (96 kB)
Collecting Werkzeug>=2.3.3
  Downloading Werkzeug-2.3.6-py3-none-any.whl (242 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 242.5/242.5 kB 5.0 MB/s eta 0:00:00
Collecting importlib-metadata>=3.6.0
  Downloading importlib_metadata-6.6.0-py3-none-any.whl (22 kB)
```

Figura 39-Instalar Flask