OnColo

Documento com as “Padronizações Utilizadas” para a codificação do projeto

Documento elaborado por:

- Debora Maria Coelho Nascimento (professora): créditos do modelo e orientações
- Any Caroliny Sousa Silva (Aluno)
- Arthur Matheus Dos Santos Lima (Aluno)
- Marcel Reinan Sa Dos Santos (Aluno)
- Leticia Cena Dos Santos (Aluno)
- Paulo Ricardo De Jesus Lima (Aluno)
- Rafael Vinicius Sousa (Aluno)

1- Estilos

#Indentação

Frontend: O Framework Flutter utilizado sugere indentação escada que é configurada automaticamente a cada arquivo salvo.

Backend: O Framework Django é mais restrito quanto às indentações devendo qualquer código abaixo de uma função ou abaixo de uma classe/função estar indentado em pelo menos 1 espaço a mais do que o seu texto anterior. Padrão Universal Python;

#Import

São sempre feitos nas primeiras linhas do projeto e com o endereço completo da biblioteca referenciada. Sem atalhos.

#Declaração de variáveis

Backend: snake case para declaração de variáveis, camel case para declaração de funções em portugues exceto nomes específicos do django.

Frontend: camelCase para declaração de variáveis e funções. Nomes em inglês exceto o vocabulário comum relativo ao projeto (Fisioterapeuta, instituição e Paciente);

Exemplo de camelCase: listarPaciente;

 # Espaços em expressões e instruções

Sempre circunde os seguintes operadores binários com um único espaço de cada lado: =, ==, <, >, !=, <>, <=, >=, in, not in, is, and, or, not

• Operadores binários

– Não é necessário espaços em branco entre operadores

```
a += c + d;
a = (a + b) / (c * d);
```

– Exceto: ++, --, += etc

 # Definição de strings (depende da linguagem utilizada)

Utilização de aspas duplas para definição de todas as strings.

 # Estilos de nomes

Os estilos de nomes utilizados para declaração de classes, funções e variáveis foram em lowerCamelCase onde colocam a primeira letra de cada palavra em maiúscula, exceto a primeira que é sempre minúscula, mesmo que seja um acrônimo.

● Para declaração de nomes de arquivos utilizou-se o padrão de lowercase_with_underscores utilizam apenas as letras minúsculas, mesmo para acrônimos com “_”.

 # Convenções para Nomes:

Módulos/Classes

• Utilizou-se o padrão (**UpperCamelCase**), exceto no caso de classes privadas que devem começar com um underscore. Aplicando-se o padrão de uso de substantivos.

Funções / Métodos

• Nomes de Funções/Métodos usam o padrão de (**lowerCamelCase**), são verbos e estão em português;

Nome de arquivos de pacotes usam o padrão de "minúsculas_com_underscores (**lowercase_with_underscores**).

– Variáveis

• Nomes de variáveis deveriam começar com minúsculas, com cada inicial de palavra

interna em maiúscula. "nomeComeçandoPorMinúscula" (**lowerCamelCase**) – Exemplo: idadeAluno

– para os casos de variáveis com escopo “público”, deve-se começar com o prefixo do módulo (maiúsculas)… » … de forma a indicar onde foram declaradas

 # Comentários

Foram feitos comentários com uma descrição geral do objetivo de cada função. No frontend sempre se iniciando com ‘///’ e no backend com #.

 # Catálogo de inspeção

Outras regras importantes que devem ser verificadas:

Imports

Todas bibliotecas importadas estão sendo utilizadas? Não importar bibliotecas que não serão usadas.

Código morto

Não deixar códigos de teste antigos em comentários em qualquer arquivo de código executável e que faça parte do escopo e repositório do projeto.

2- Documentação

 # Arquivo de README1

OnColo - API

Substitua esse parágrafo por uma breve descrição do produto a ser desenvolvido, destacando quais problemas resolve ou quais vantagens apresenta.

Autoria

- **Professor(a) Correspondente**: Ricardo Salgueiro

- Tiago Plácido (PO)
- Debora Maria Coelho Nascimento (professora)
- Edilayne Slagueiro (professora)
- Any Caroliny Sousa Silva (Aluno)
- Arthur Matheus Dos Santos Lima (Aluno)
- Marcel Reinan Sa Dos Santos (Aluno)
- Leticia Cena Dos Santos (Aluno)
- Paulo Ricardo De Jesus Lima (Aluno)
- Rafael Vinicius Sousa (Aluno)

Arquitetura

- Linguagem: Python
- Framework: Django e Django reste framework

MVP

Essas instruções permitirão que você obtenha uma cópia do projeto em operação na sua máquina local para fins de desenvolvimento e teste.

📋 Pré-requisitos

Para instalação do software é necessário:

- Python 3.10
- Django 4.1.7
- django cors headers 3.14.0
- djangorestframework 3.14.0
- pytz 2022.7.1
- sqlparse 0.4.3
- tzdata 2022.7
- django filter 22.1
- validate docbr 1.10.0

🔧 Instalação

1. Clone o repositório:

> git clone https://github.com/DCOMP-UFS/2022-2-praticas-projetoinca.git

2. Crie um ambiente virtual:

> python -m venv venv

3. Ative o ambiente virtual:

> venv/Scripts/activate

4. Instale as dependências:

> pip install -r requisitos.txt

6. Execute as migrações:

> python manage.py makemigrations

> python manage.py migrate

5. Execute o servidor:

> python manage.py runserver

⚙ Executando os testes

Para executar os testes automatizados deste sistema, você pode seguir os seguintes

passos:

1. Ative o ambiente virtual:

> venv/Scripts/activate

2. Execute o seguinte comando para executar todos os testes automatizados:

python manage.py test

🔩 Analise os testes de ponta a ponta

Os testes de ponta a ponta (end-to-end) são testes que verificam se todo o sistema funciona corretamente, do início ao fim. Eles testam a interação de diferentes partes do sistema, incluindo a API, banco de dados e interface do usuário. Alguns exemplos de testes de ponta a ponta incluir:

- Testes para verificar se a API retorna os resultados corretos com base em diferentes entradas de usuário
- Testes para verificar se a integração com o banco de dados funciona corretamente
- Testes para verificar se a interface do usuário se comporta corretamente quando interagindo com a API

⌨️ E testes de estilo de codificação

Os testes de estilo de codificação (code style tests) verificam se o código segue as convenções de estilo e formatação de código definidas pela equipe de desenvolvimento. Eles ajudam a garantir que o código seja legível e consistente em todo o projeto. Alguns exemplos de testes de estilo de codificação para o seu projeto podem incluir:

- Testes para verificar se a indentação do código está correta
- Testes para verificar se a nomenclatura de variáveis e funções está em conformidade com as diretrizes do projeto
- Testes para verificar se a documentação do código está completa e precisa

📦 Implantação

Adicione notas adicionais sobre como implantar isso em um sistema ativo

🛠️ Construído com

- Django - O framework web usado
- MySQL - Banco de dados usado

📌 Versão

Nós usamos Git para controle de versão.

API

Visão Geral

Esta API permite aos usuários fazerem CRUD (Create, Retrieve, Update e Delete) de pacientes, fisioterapeutas, instituições e administradores.

Pacientes

ENDPOINT	HTTP METHOD	CRUD METHOD	RESULTADO
`paciente/`	GET	LEITURA	listagem dos pacientes
`paciente/`	POST	CRIAR	adiciona uma paciente
`paciente/{id}`	GET	LEITURA	listagem de uma única paciente
`paciente/{id}`	PUT	ATUALIZAR	atualiza uma paciente existente
`paciente/{id}`	DELETE	DELETAR	deleta uma paciente existente

Exemplo de corpo da requisição:


```
{
	"matricula": "222399",
	"nome": "Telma Silva",
	"altura": "1.5",
	"peso": "50.0",
	"cpf": "77229285048",
	"telefone": "9725798520",
	"data_nascimento": "1999-12-10"
}
```


Exemplo de resposta:


```
{
	"id": 5,
	"matricula": "222399",
	"nome": "Telma Silva",
	"altura": "1.5",
	"peso": "50.0",
	"imc": "22.7"
	"cpf": "77229285048",
	"telefone": "9725798520",
	"data_nascimento": "1999-12-10"
```

}

Fisioterapeuta

ENDPOINT	HTTP METHOD	CRUD METHOD	RESULTADO
`fisioterapeuta/`	GET	LEITURA	listagem dos fisoterapeutas
`fisioterapeuta/`	POST	CRIAR	adiciona um fisioterapeuta
`fisioterapeuta/{id}`	GET	LEITURA	listagem de um único fisioterapeuta
`fisioterapeuta/{id}`	PUT	ATUALIZAR	atualiza um fisioterapeuta existente
`fisioterapeuta/{id}`	DELETE	DELETAR	deleta um fisioterapeuta existente

Exemplo de corpo da requisição:

```
{

    "matricula": "222399",

    "nome": "Thiago",

    "sobrenome": "Silva",

    "email": "thiago@email.com",

    "senha": "11111111111",

    "instituicao": "2",

}
```

Exemplo de resposta:

```
{
```

```
    "id": "25",
    "matricula": "222399",
    "nome": "Thiago",
    "sobrenome": "Silva",
    "email": "thiago@email.com",
    "senha": "11111111111",
    "instituicao": "2",
}
```


Fisioterapeuta

ENDPOINT	HTTP METHOD	CRUD METHOD	RESULTADO
`instituicao/`	GET	LEITURA	listagem das instituicões
`instituicao/`	POST	CRIAR	adiciona uma instituicao
`instituicao/{id}/`	GET	LEITURA	listagem de uma única instituicao
`instituicao/{id}/`	PUT	ATuALIZAR	atualiza uma instituicao existente
`instituicao/{id}/`	DELETE	DELETAR	deleta uma instituicao existente
`instituicao/{id}/fisioterapeuta/`	GET	LEITURA	listagem dos fisioterapeutas cadastrado em um determinada instituição

Exemplo de corpo da requisição:


```
{
    "nome": "INCA",
```

"cnpj": "71930356000172",

"email": "INCA@email.com",

"senha": "11111111111",

}

Exemplo de resposta:

{

"id": "34",

"nome": "INCA",

"cnpj": "71930356000172",

"email": "INCA@email.com",

"senha": "11111111111",

}

Erros

A API pode retornar os seguintes erros:

1. 400 Bad Request - O corpo da requisição está mal formado ou faltando algum parâmetro obrigatório.
2. 401 Unauthorized - O usuário não está autenticado.
3. 404 Not Found - O recurso solicitado não foi encontrado.
4. 405 Method Not Allowed - O método HTTP não é suportado pelo endpoint solicitado.
5. 500 Internal Server Error - Ocorreu um erro interno no servidor.

 # Arquivo de README2

OnColo - Frontend

Tecnologias:

- Flutter 3.7.6
- **Dart 2.19.3**
- **Bibliotecas externas:**
    - responsive_builder: 0.6.3
    - dartdoc: 6.2.0
    - http: 0.13.5
    - google_fonts: 3.0.1

Arquitetura

- Camadas
- Padrão MVC

Ferramentas para testes:

- Emuladores Android Studio
- Flutter Debugger
- Google Chrome
- adb devices - para teste nativo em dispositivo móvel via cabo USB

Repositório do GitHub (2022-2-praticas-projetoinca-front-end)

Estrutura de Pastas do Projeto

Divisão das pastas entre 3 categorias (view, model e control). A view subdivide-se em pages, widgets e resources. As pages são arquivos contendo código fonte correspondente as telas do projeto. A pasta widgets correspondente aos mini componentes visuais da tela que são reutilizáveis por uma ou mais páginas. Já a pasta resources corresponde aos textos e cores utilizados no OnColo.Na model temos as classes que representam a modelagem do nosso banco de dados de acordo com os diversos agentes e stakeholders do projeto. Por fim, a pasta control subdivide-se em repositories e controllerss. Os repositories são responsáveis por executar funções de get, post e put na API remota e os controllers de realizar tratamentos nos dados coletados da view antes de envia-los as funções da API.

Branchs

> main

> leticia

> paulo

> marcel

3 branchs foram criadas cada uma no respectivo nome dos 3 membros do frontend. Todas as branchs iniciaram a partir da mesma estrutura de pastas e cada membro ia fazendo commits e push gradual das suas funcionalidades. Logo após essa etapa concluída foi realizado reunião geral para merge das branchs na main e correção de conflitos.

Como executar o projeto

Pré-requisito Setup do Flutter realizado:

https://docs.flutter.dev/get-started/install

https://docs.flutter.dev/get-started/editor

- Clone do repositório

git clone https://github.com/DCOMP-UFS/2022-2-praticas-projetoinca-front-end.git

- Abrir a pasta oncolo

cd oncolo (para usuários linux)

- Para executar o projeto

flutter run (Será escolhido o ambiente nativo da sua maquina para execução [Windows, Linux, Device externo)

flutter run -d chrome (Para executar via Google Chrome)