

Gestão de Calendário Pessoal

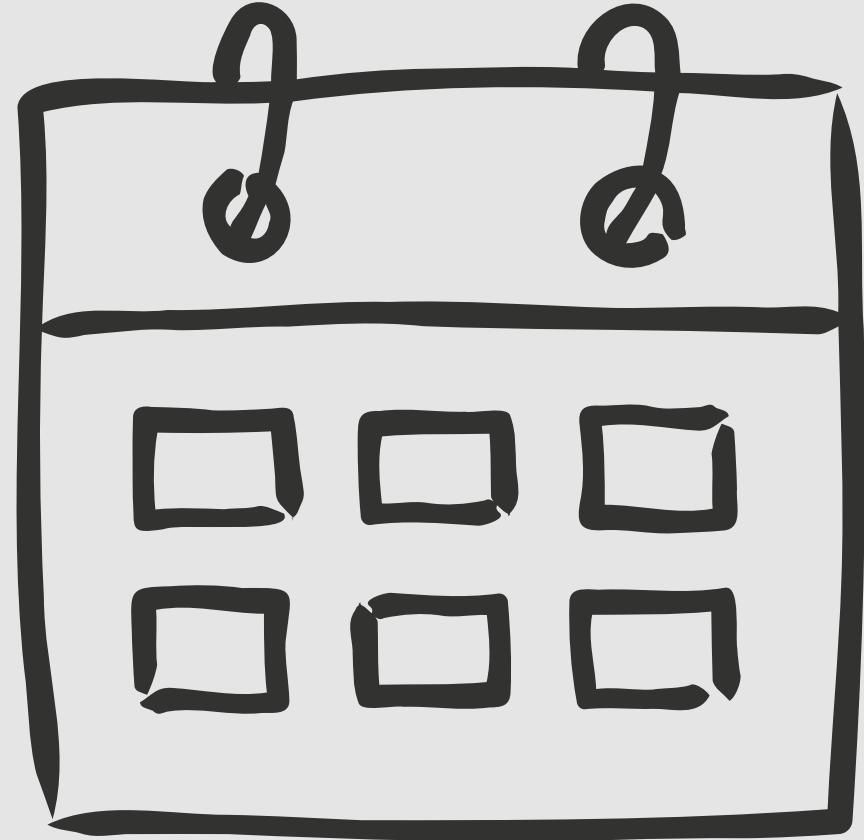
Martim Amador Vasco 28694

Programação Orientada Objetos



Objetivo do Projeto

1



Desenvolver um sistema de gestão de calendário pessoal que permita gerir utilizadores, eventos e categorias, com validações e persistência de dados.

Tecnologias Utilizadas



Python



Programação Orientada
a Objetos



UML (PlantUML)



JSON (persistência)



VS Code

Diagrams

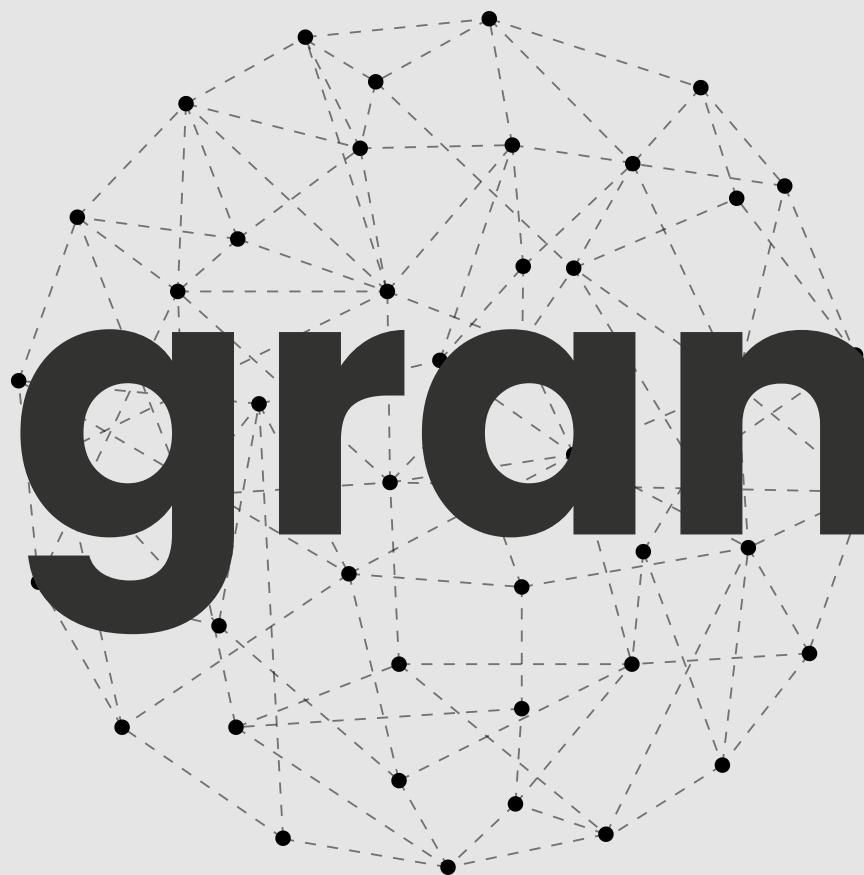


Diagrama de Classes

1

O sistema é composto pelas classes Event, Category, User e CalendarManager, responsáveis pela lógica e gestão dos eventos.

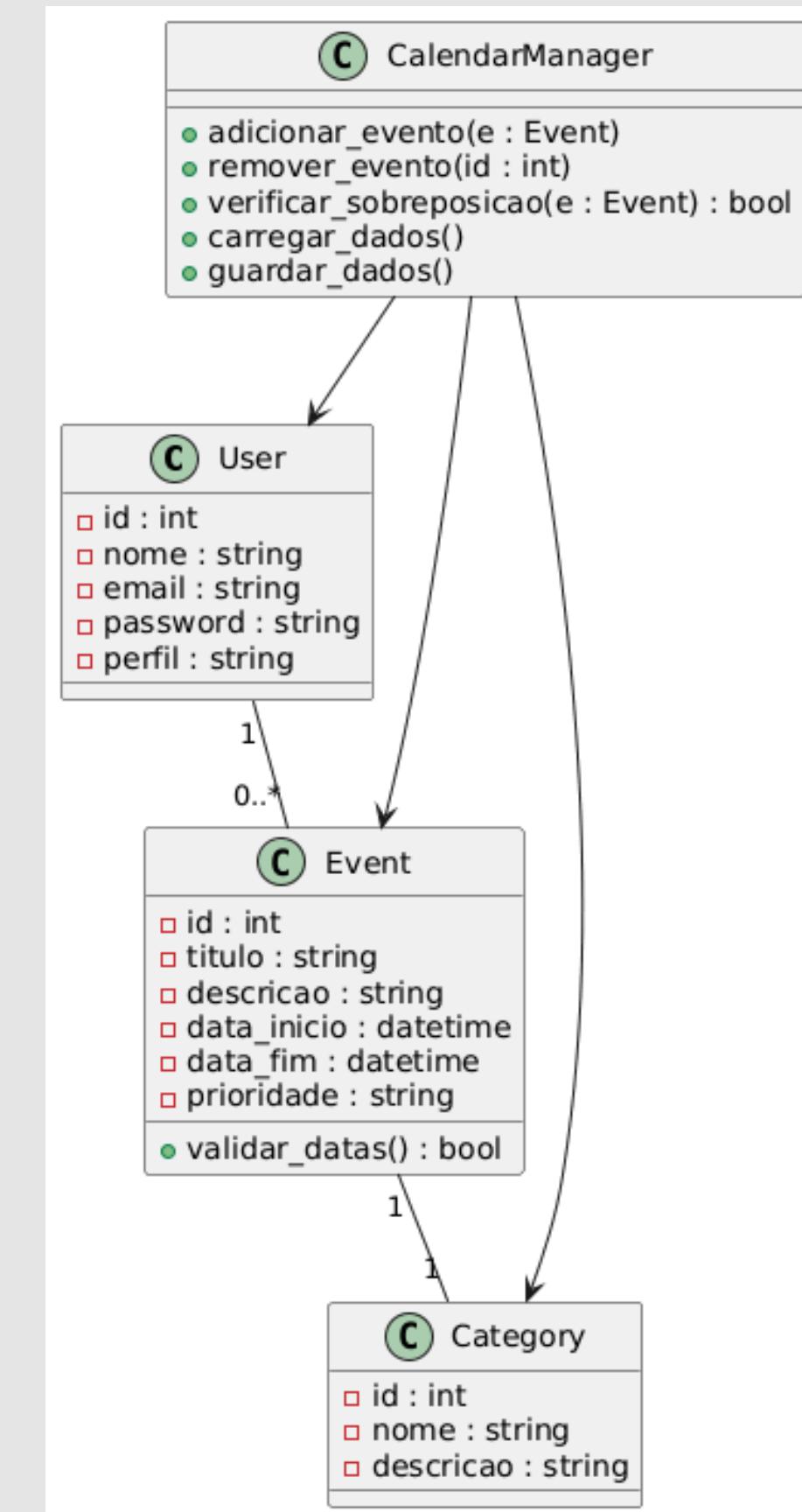


Diagrama de Casos de Uso

1

O utilizador pode adicionar, remover e visualizar eventos no sistema

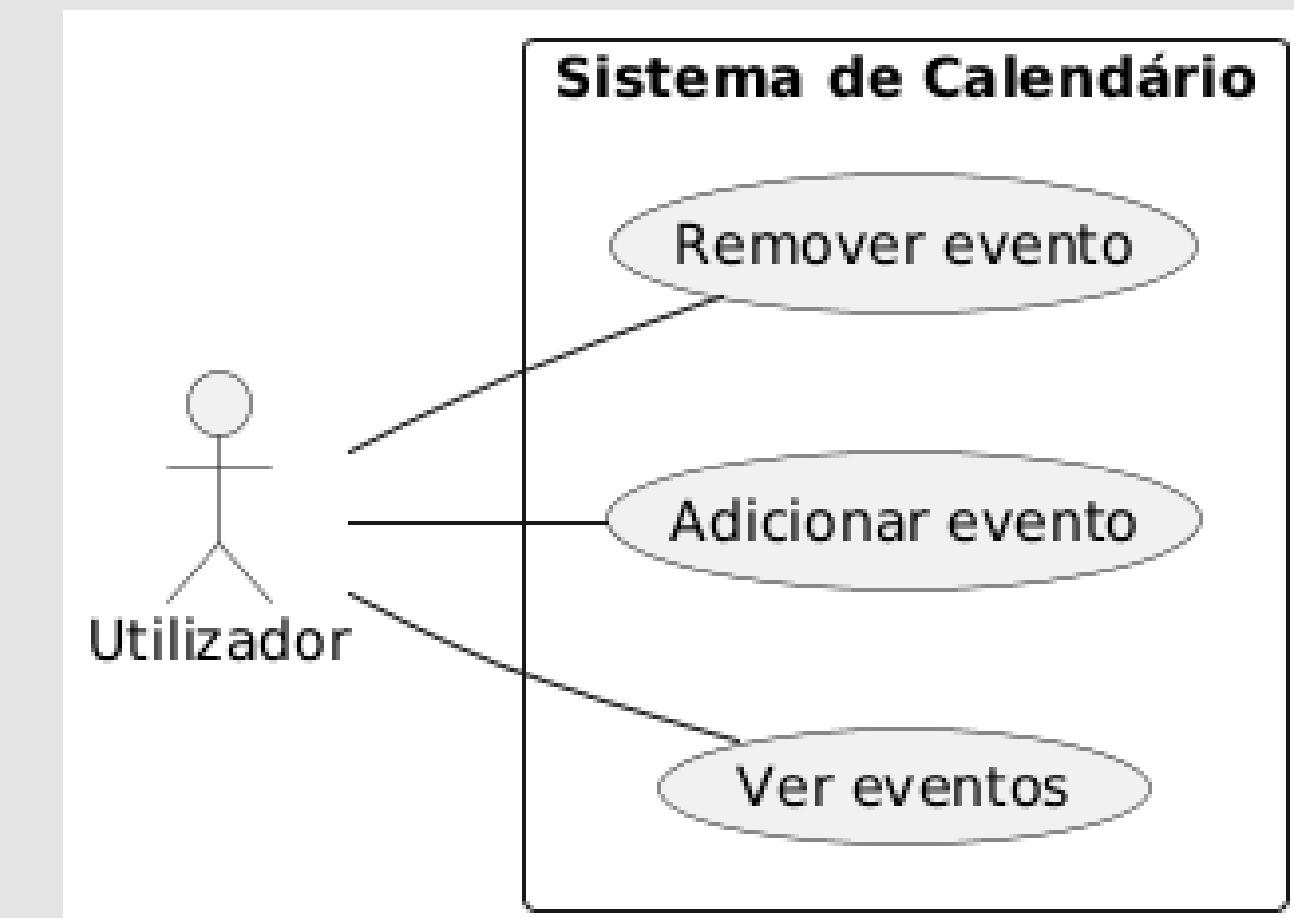
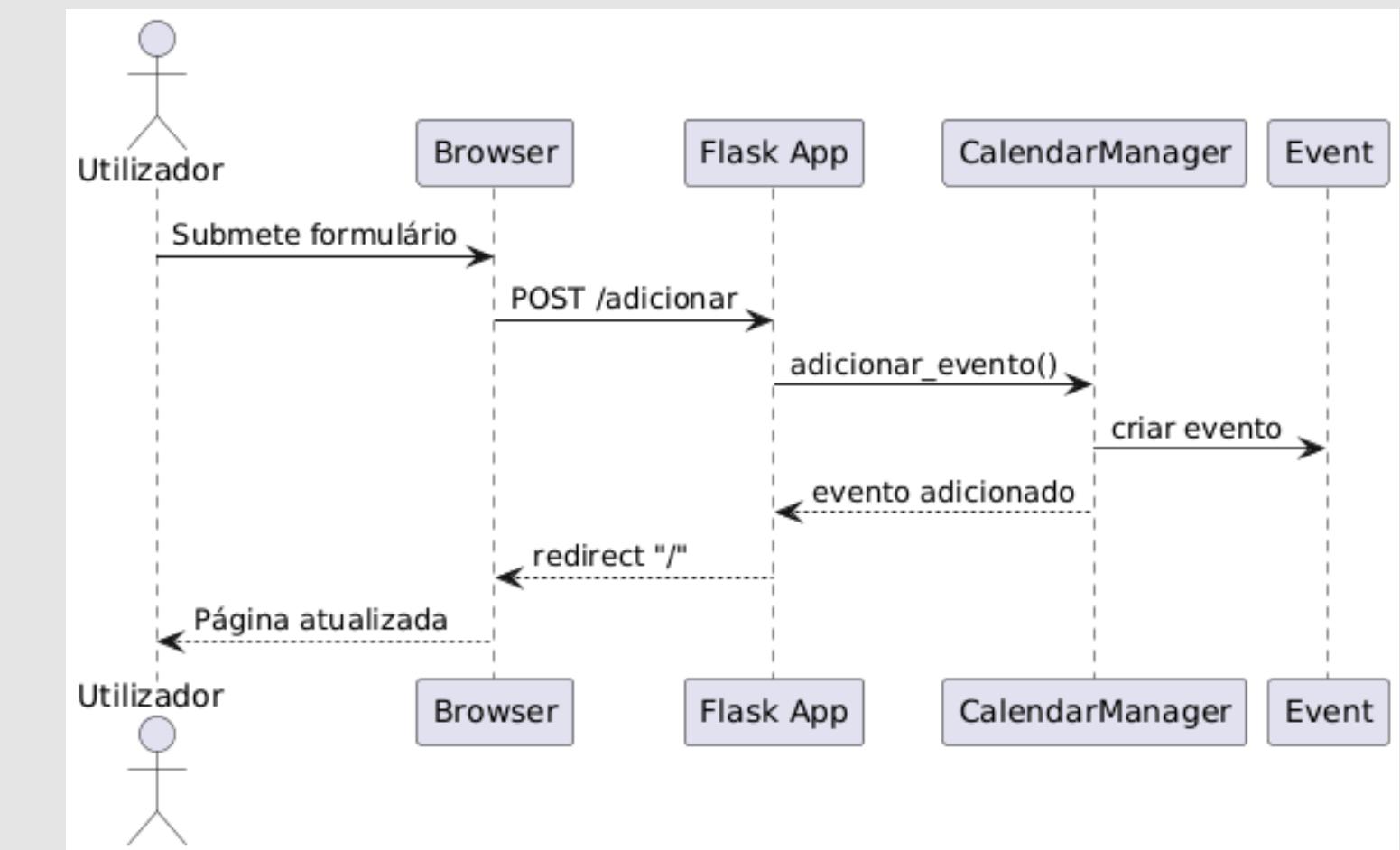


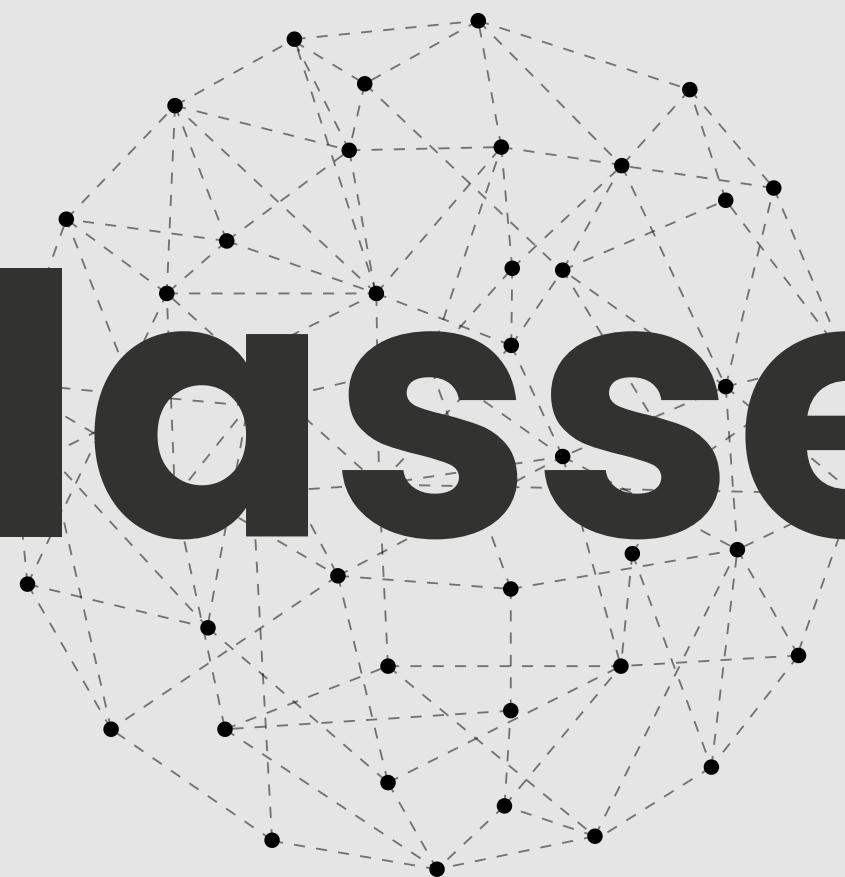
Diagrama de Sequência

1

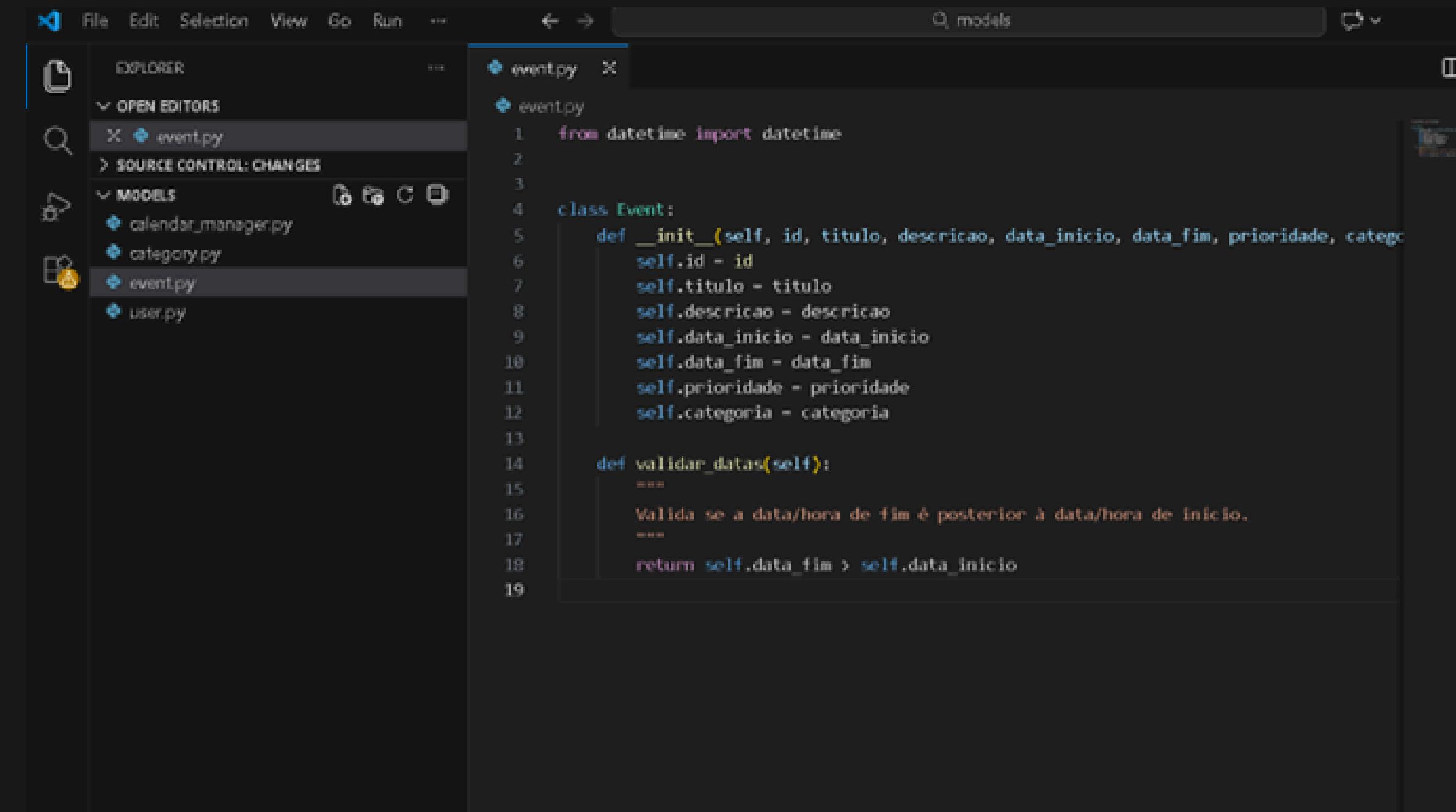
O diagrama representa o fluxo de criação de um evento desde a interface até ao gestor do calendário



classes



ClasseEvent

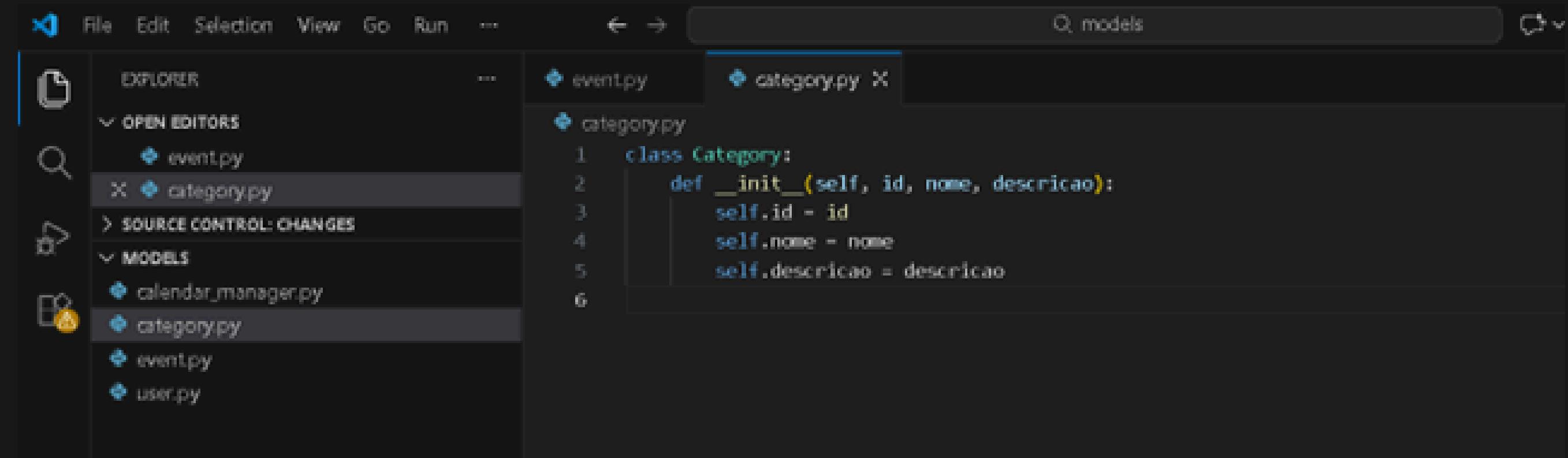


The screenshot shows a code editor interface with a dark theme. The left sidebar contains a tree view of files under 'MODELS': event.py (selected), calendar_manager.py, category.py, user.py, and __init__.py. The right pane displays the contents of the selected 'event.py' file.

```
 1  from datetime import datetime
 2
 3
 4  class Event:
 5      def __init__(self, id, titulo, descricao, data_inicio, data_fim, prioridade, categoria):
 6          self.id = id
 7          self.titulo = titulo
 8          self.descricao = descricao
 9          self.data_inicio = data_inicio
10          self.data_fim = data_fim
11          self.prioridade = prioridade
12          self.categoria = categoria
13
14      def validar_datas(self):
15          """
16              Valida se a data/hora de fim é posterior à data/hora de inicio.
17          """
18          return self.data_fim > self.data_inicio
19
```

A classe Event representa um evento do calendário e valida as datas de início e fim

Classe Category

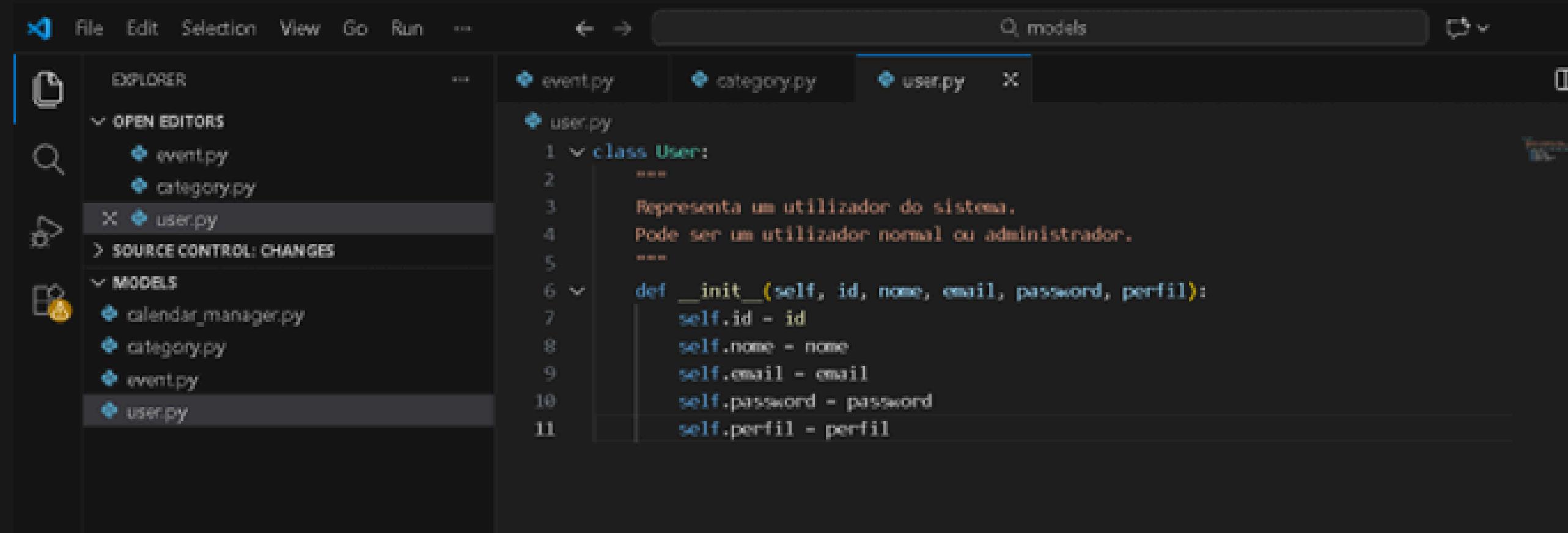


The screenshot shows a code editor interface with a dark theme. The top bar includes icons for File, Edit, Selection, View, Go, Run, and a search bar containing 'models'. The left sidebar has sections for EXPLORER, OPEN EDITORS, SOURCE CONTROL: CHANGES, and MODELS. Under OPEN EDITORS, 'category.py' is selected. In the main editor area, the 'category.py' file is open, displaying the following Python code:

```
1 class Category:
2     def __init__(self, id, nome, descricao):
3         self.id = id
4         self.nome = nome
5         self.descricao = descricao
```

A classe Category permite classificar eventos sem conter regras de negócio próprias

Classe User

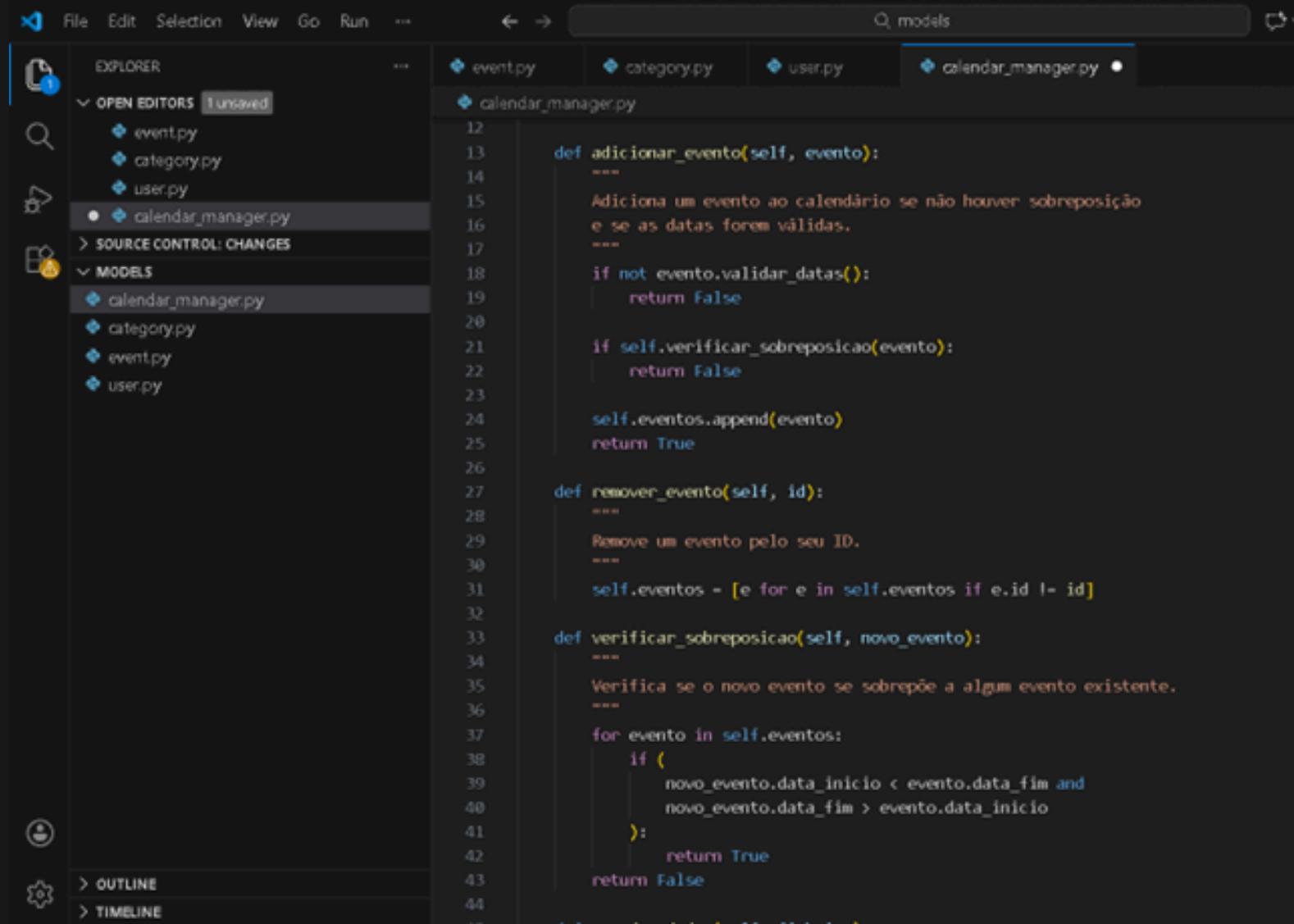


The screenshot shows a code editor interface with a dark theme. The top bar includes standard menu items like File, Edit, Selection, View, Go, Run, and a search bar containing the text "models". The left sidebar, titled "EXPLORER", lists several files under "OPEN EDITORS": event.py, category.py, user.py, calendar_manager.py, category.py, event.py, and user.py. The "user.py" file under "OPEN EDITORS" is currently selected. The main editor area displays the following Python code:

```
1  #!/usr/bin/python
2  """
3      Representa um utilizador do sistema.
4      Pode ser um utilizador normal ou administrador.
5  """
6  class User:
7      """"
8          Representa um utilizador do sistema.
9          Pode ser um utilizador normal ou administrador.
10         """
11     def __init__(self, id, nome, email, password, perfil):
12         self.id = id
13         self.nome = nome
14         self.email = email
15         self.password = password
16         self.perfil = perfil
```

Representa os utilizadores do sistema, distinguindo perfis.

Classe CalendarManager



The screenshot shows a code editor interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, ...
- Toolbar:** Back, Forward, Search, Refresh, Save.
- Explorer:** Shows the project structure with files: event.py, category.py, user.py, and calendar_manager.py.
- Open Editors:** event.py, category.py, user.py, calendar_manager.py (selected).
- Source Control:** Changes.
- Models:** calendar_manager.py, category.py, event.py, user.py.
- Bottom Bar:** Outline, Timeline.

The code editor displays the `calendar_manager.py` file with the following content:

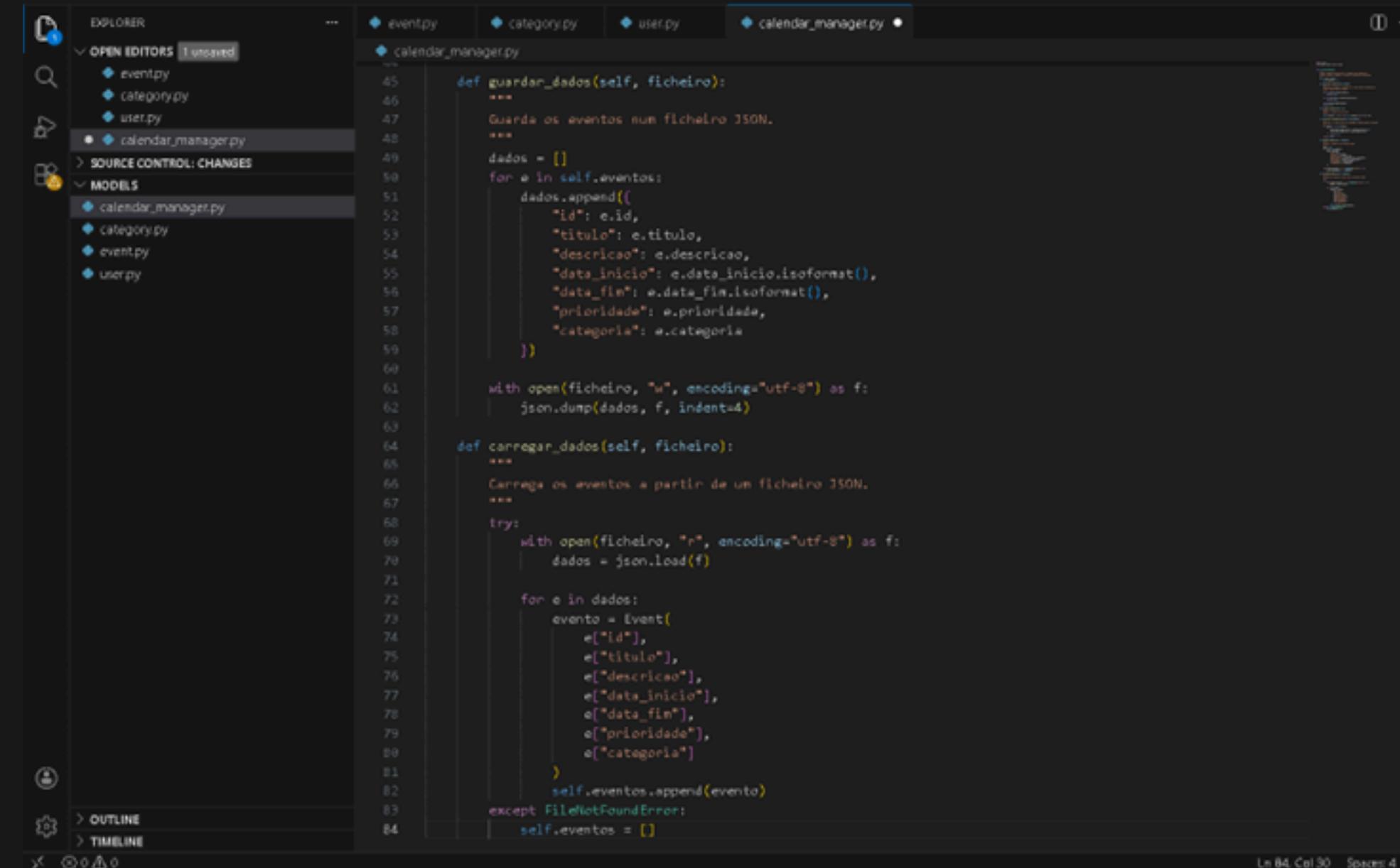
```
12 def adicionar_evento(self, evento):
13     """
14     Adiciona um evento ao calendário se não houver sobreposição
15     e se as datas forem válidas.
16     """
17     if not evento.validar_datas():
18         return False
19
20     if self.verificar_sobreposicao(evento):
21         return False
22
23     self.eventos.append(evento)
24     return True
25
26 def remover_evento(self, id):
27     """
28     Remove um evento pelo seu ID.
29     """
30     self.eventos = [e for e in self.eventos if e.id != id]
31
32 def verificar_sobreposicao(self, novo_evento):
33     """
34     Verifica se o novo evento se sobrepõe a algum evento existente.
35     """
36     for evento in self.eventos:
37         if (
38             novo_evento.data_inicio < evento.data_fim and
39             novo_evento.data_fim > evento.data_inicio
40         ):
41             return True
42     return False
```

adicionar evento

verificar sobreposição

Centraliza as regras de negócio, validações e gestão de eventos.

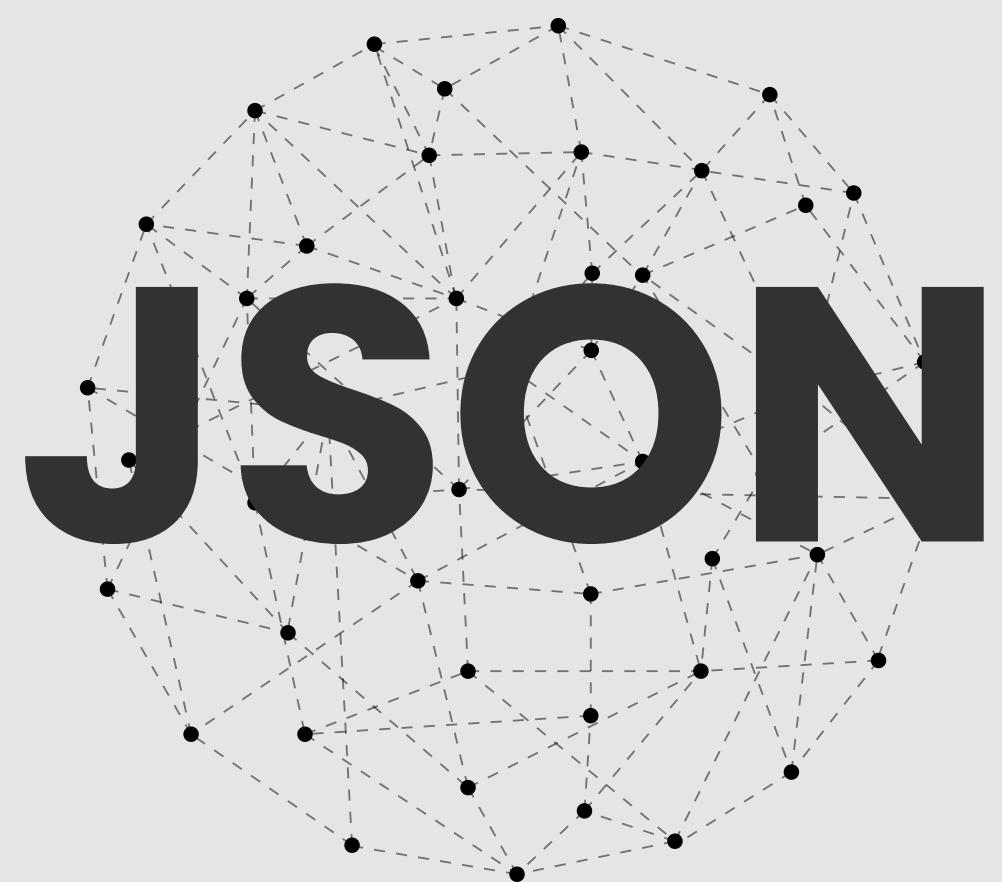
Classe CalendarManager



The screenshot shows a code editor interface with a dark theme. On the left, the Explorer sidebar displays several files: event.py, category.py, user.py, and calendar_manager.py. The calendar_manager.py file is currently selected and open in the main editor area. The code implements a class named CalendarManager with two methods: guardar_dados and carregar_dados. The guardar_dados method saves events to a JSON file, while the carregar_dados method loads events from a JSON file. The code uses the json module for serialization and deserialization.

```
45     def guardar_dados(self, ficheiro):
46         """
47             Guarda os eventos num ficheiro JSON.
48         """
49         dados = []
50         for e in self.eventos:
51             dados.append({
52                 "id": e.id,
53                 "titulo": e.titulo,
54                 "descricao": e.descricao,
55                 "data_inicio": e.data_inicio.isoformat(),
56                 "data_fim": e.data_fim.isoformat(),
57                 "prioridade": e.prioridade,
58                 "categoria": e.categoria
59             })
60
61         with open(ficheiro, "w", encoding="utf-8") as f:
62             json.dump(dados, f, indent=4)
63
64     def carregar_dados(self, ficheiro):
65         """
66             Carrega os eventos a partir de um ficheiro JSON.
67         """
68         try:
69             with open(ficheiro, "r", encoding="utf-8") as f:
70                 dados = json.load(f)
71
72                 for e in dados:
73                     evento = Event(
74                         e["id"],
75                         e["titulo"],
76                         e["descricao"],
77                         e["data_inicio"],
78                         e["data_fim"],
79                         e["prioridade"],
80                         e["categoria"]
81                     )
82                     self.eventos.append(evento)
83             except FileNotFoundError:
84                 self.eventos = []
```

Guardar eventos e Carrega os eventos a partir de um ficheiro JSON



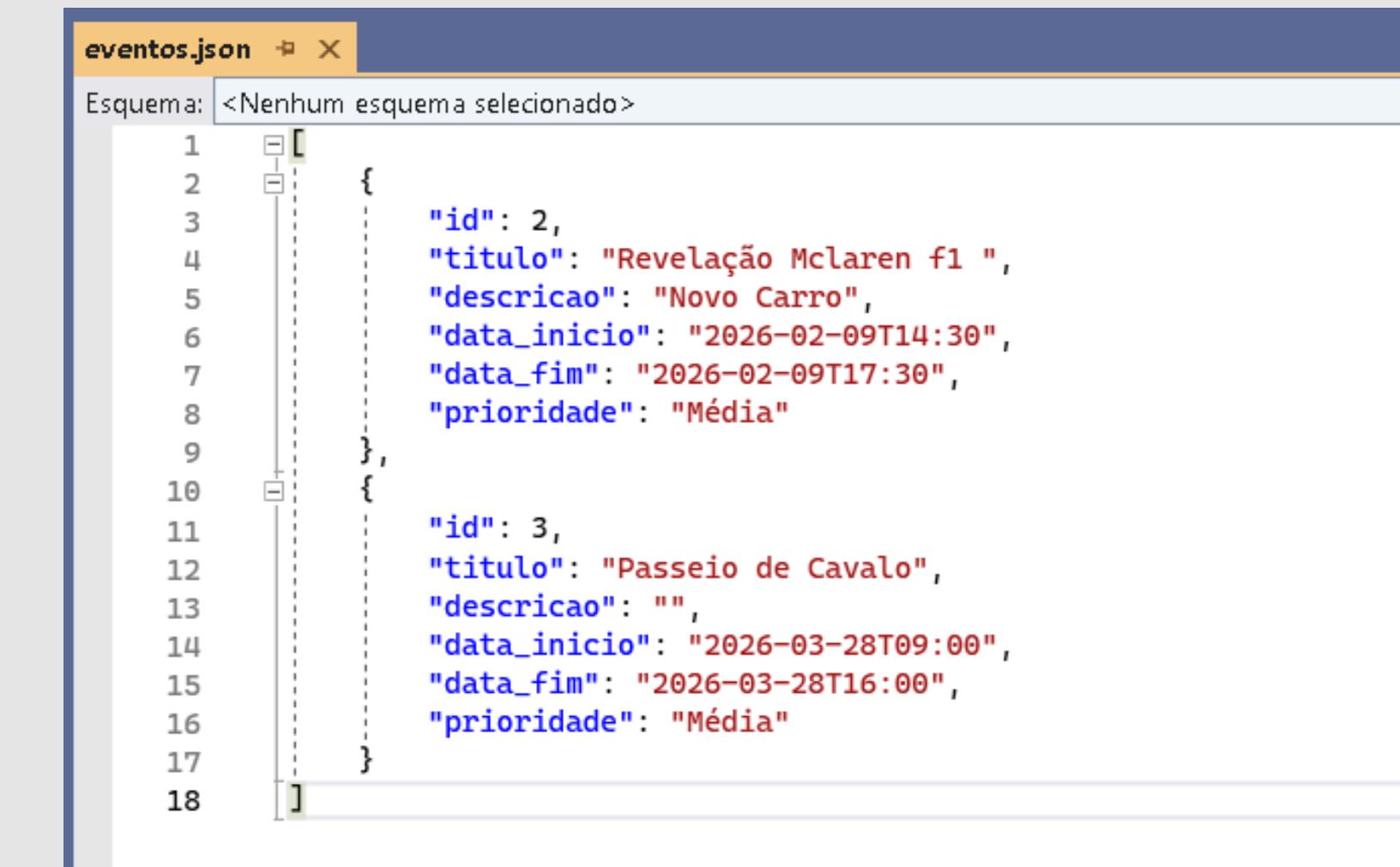
A network graph consisting of numerous small black dots connected by a web of thin, dashed grey lines. The dots are densely packed and form a roughly circular pattern, creating a complex web-like structure.

JSON

Persistência de dados (JSON)

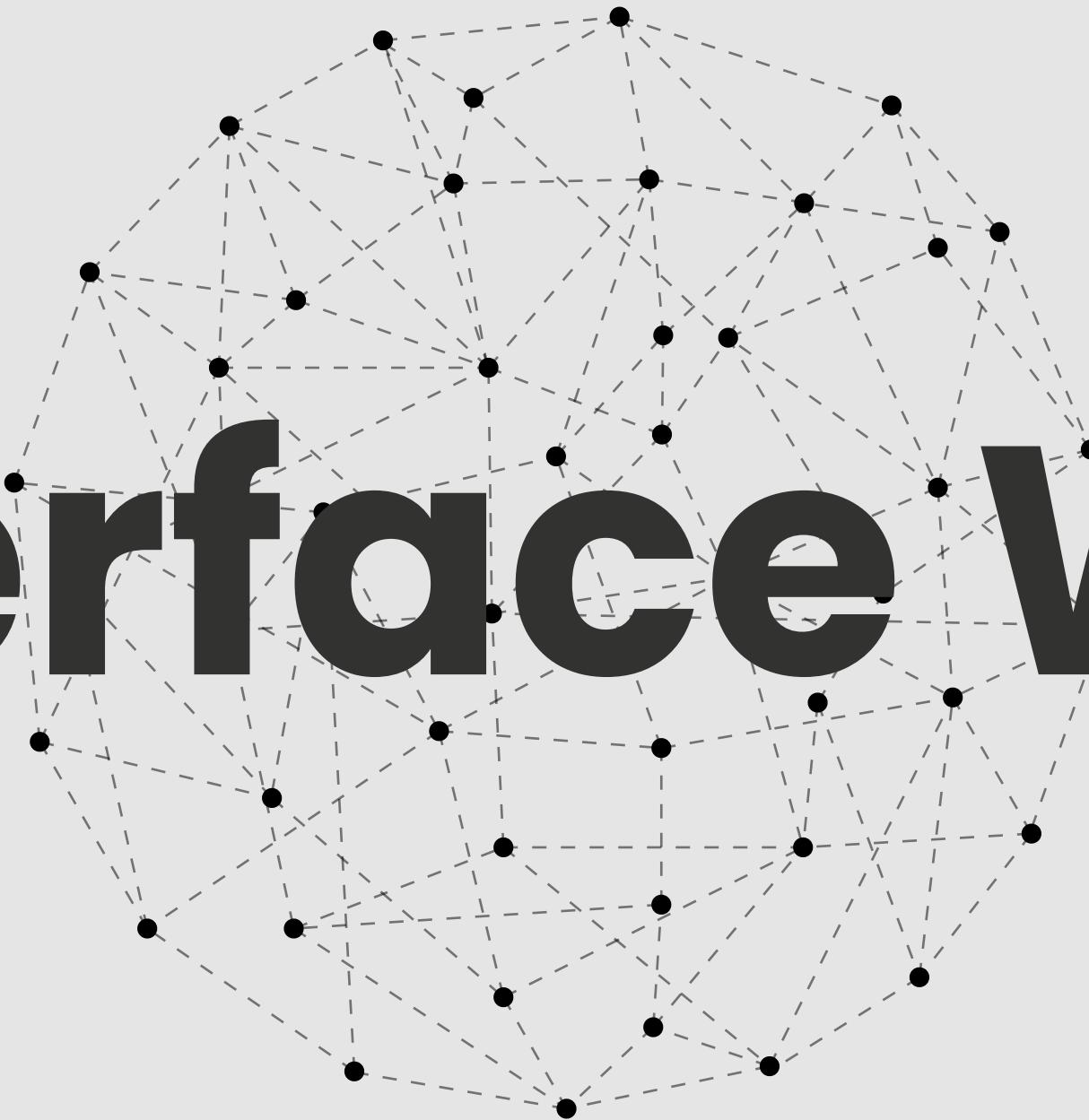
Os eventos são guardados num ficheiro JSON, permitindo persistência dos dados entre execuções da aplicação.

1

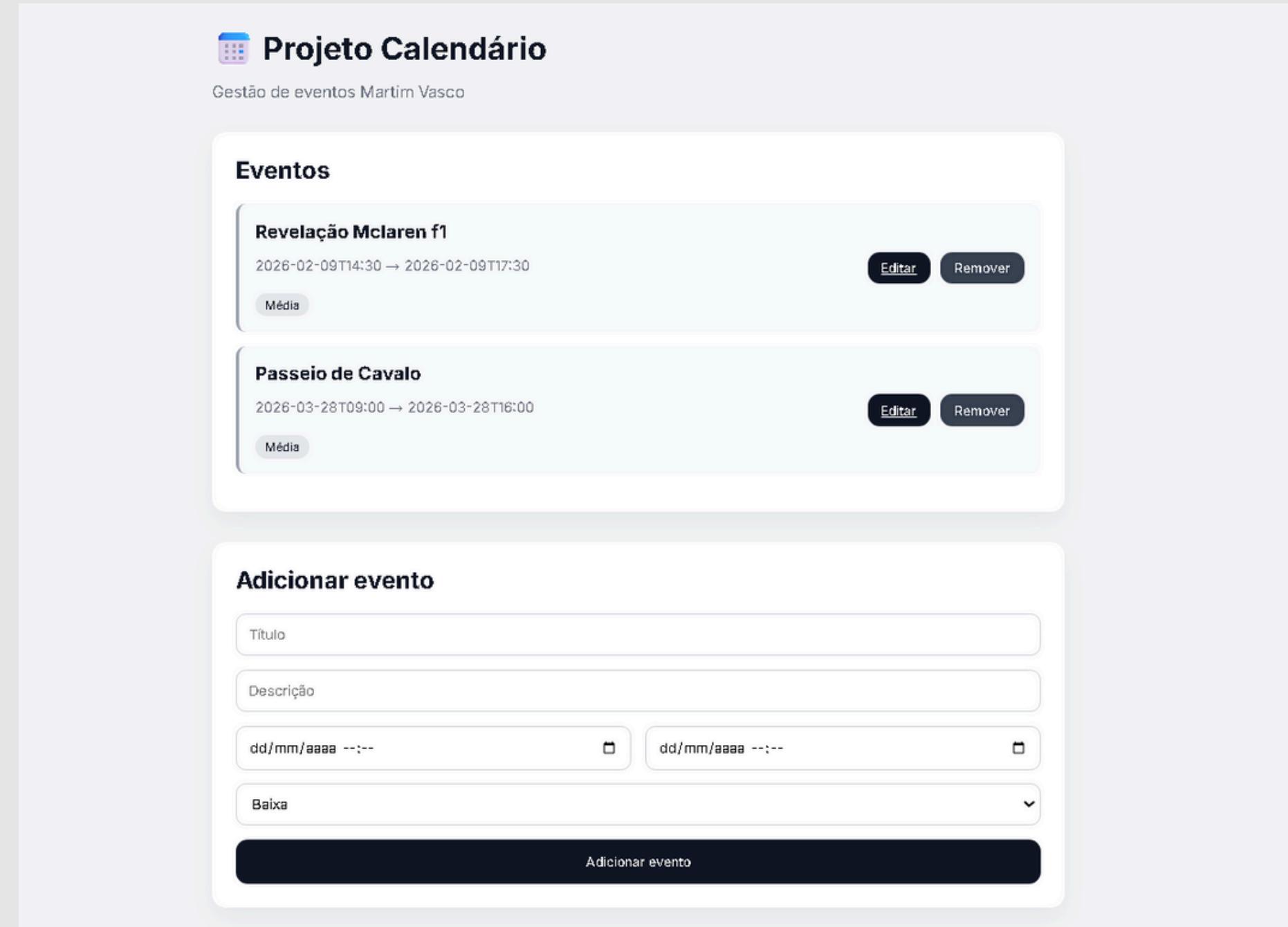


```
eventos.json ✘ X
Esquema: <Nenhum esquema selecionado>
1  [
2    {
3      "id": 2,
4      "titulo": "Revelação McLaren f1",
5      "descricao": "Novo Carro",
6      "data_inicio": "2026-02-09T14:30",
7      "data_fim": "2026-02-09T17:30",
8      "prioridade": "Média"
9    },
10   {
11     "id": 3,
12     "titulo": "Passeio de Cavalo",
13     "descricao": "",
14     "data_inicio": "2026-03-28T09:00",
15     "data_fim": "2026-03-28T16:00",
16     "prioridade": "Média"
17   }
18 ]
```

Interface web

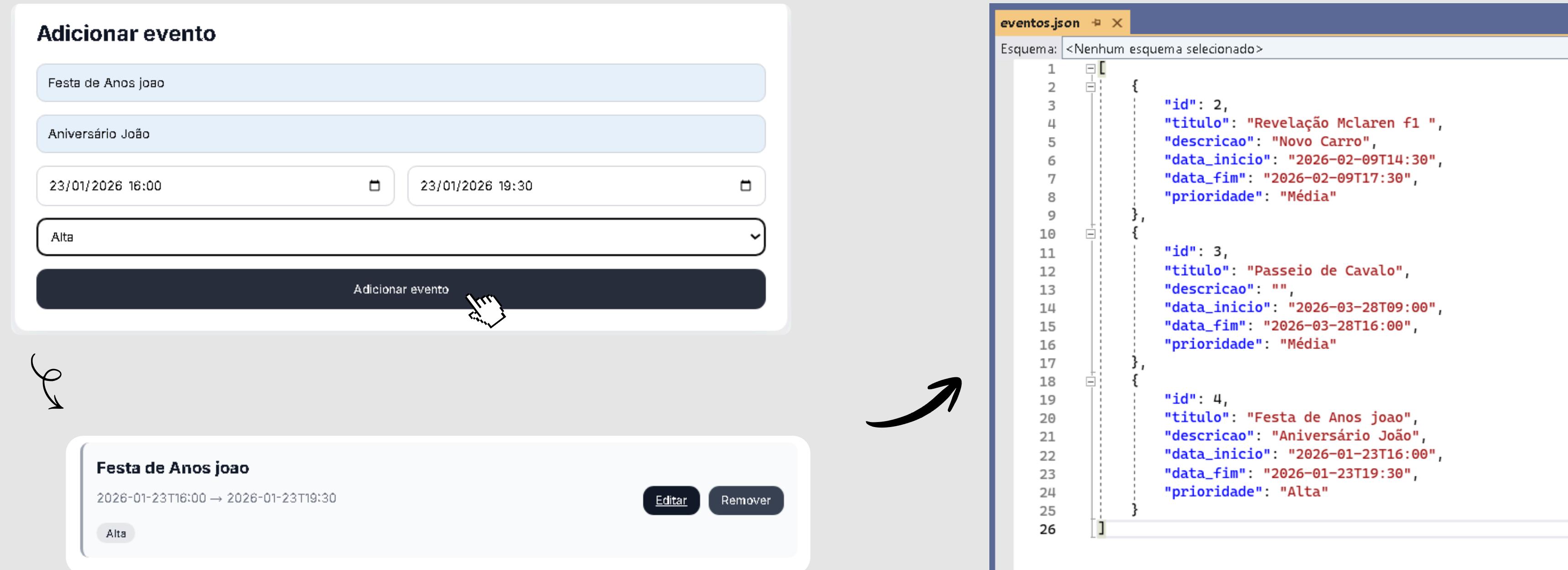


Interface Web da Aplicação



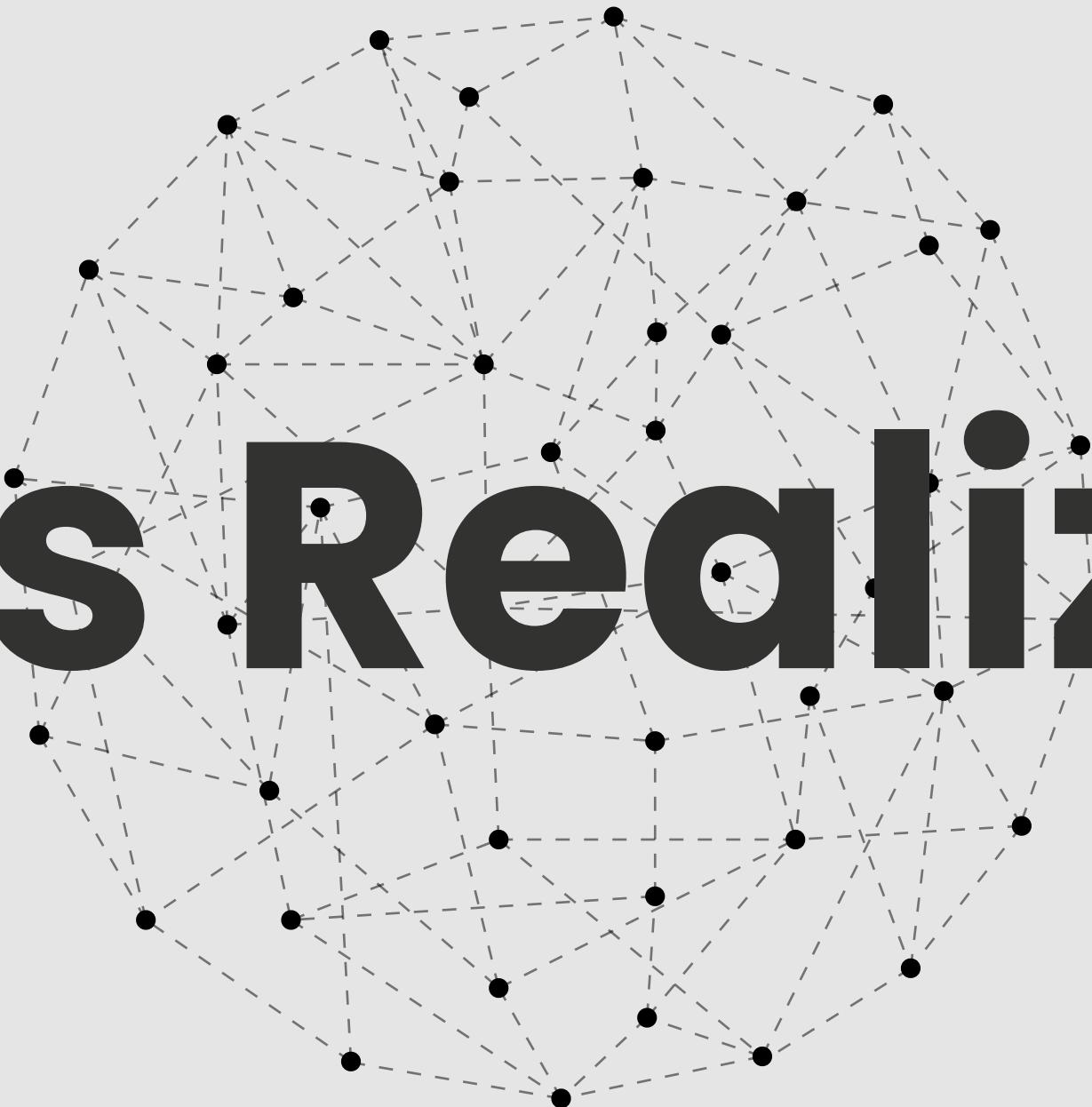
A aplicação disponibiliza uma interface web desenvolvida com Flask, permitindo adicionar, editar e remover eventos de forma intuitiva. <http://127.0.0.1:5000/>

Fluxo de Funcionamento



O utilizador interage com a interface web, os dados são processados em Python e armazenados em JSON.

Testes Realizados



Testes

```
PS C:\Users\marti\OneDrive\Ambiente de Trabalho\Projeto_Calendario\E4_Projeto> python app.py
● >>
True
False
❖ PS C:\Users\marti\OneDrive\Ambiente de Trabalho\Projeto_Calendario\E4_Projeto> █
```

Foram realizados testes para validar a criação de eventos e deteção de sobreposição.

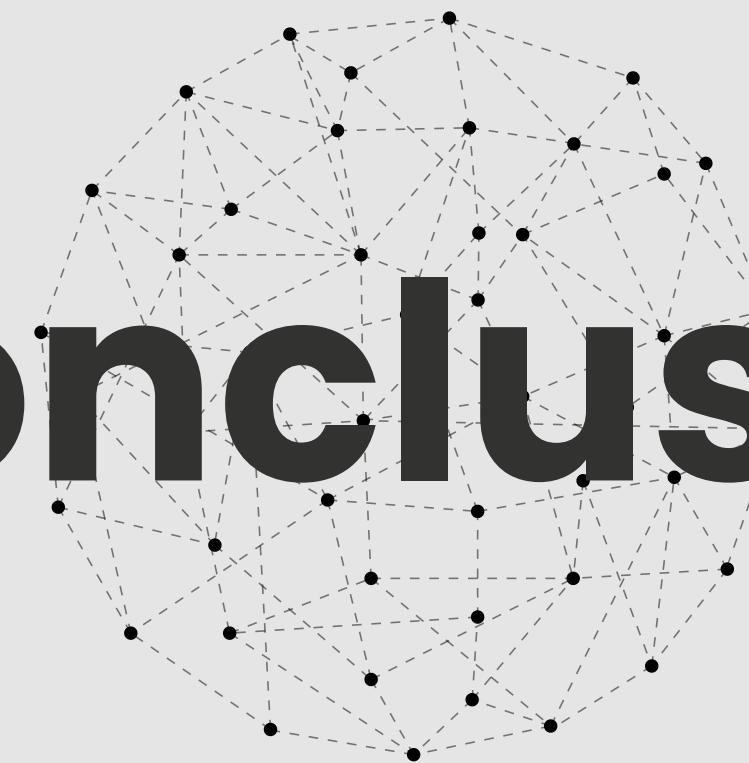
Limitações e melhorias futuras



Tecnologias Utilizadas

- 1 Visualização em calendário mensal
- 2 Filtros por prioridade
- 3 Filtros por prioridade

Conclusão



O projeto cumpre os requisitos propostos e demonstra a aplicação de POO, validações e persistência de dados.