Referencia

Campos universais

Lista de campos que todas as classes tem:

```
- Id
- CreatedAt
- UpdatedAt
```

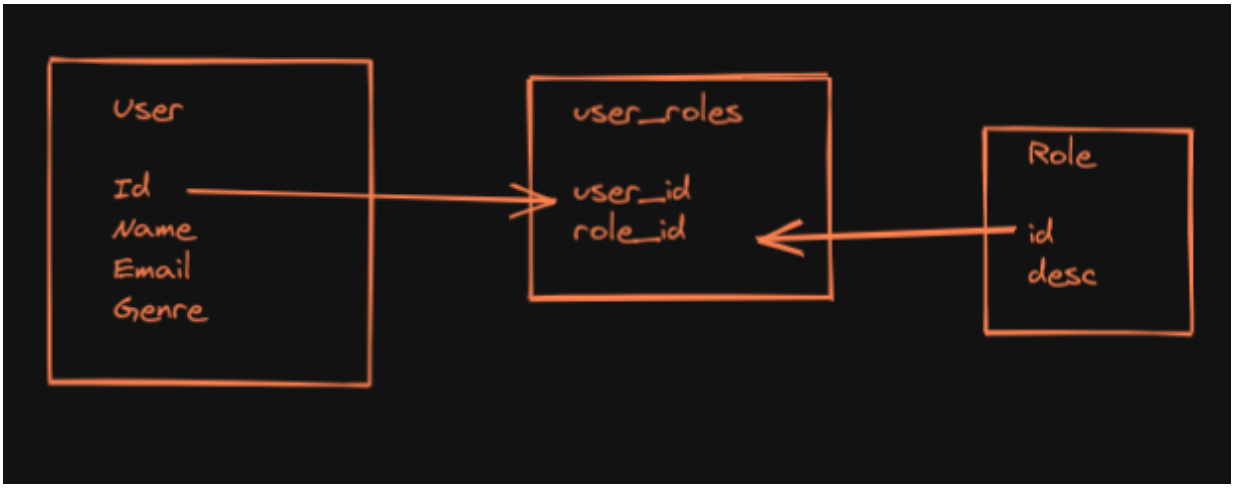
Classes com suas propriedades

As classes que temos no banco de dados são as seguintes:

> *note que sempre que aparecerem campos com -> quer dizer que há uma ligação, então são campos ocultos no output e com rotas especiais para acesso.*

```
User:
    - Name
    - Email
    - Pass       # Campo oculto no output
    - Genre
    -> Roles
    -> Serviços  # Serviços em que os usuários estão inscritos
Role:
    -> Owner     # Serviço dono da role
    - Id         # Nesse caso o Id é o nome da role
    - Desc
    -> Users
    -> Perms     # Permissões
Permission:
    - Type       # Tipo de permissão pode ser except ou pode ser omitido
                 # (Todas as roles menos uma)
    - Level      # W(rite) | R(ead) | WR | RW
    -> Parent    # Role que tem permissão sobre child
Service:
    -> Parent    # Serviço pai
    -> Child     # Serviço filho (herda usuários e roles do pai)
    - Name
```

```
    - Desc
    -> Users     # Membros inscritos no serviço
    -> Roles     # Roles inscritas no serviço
```

> *Para deixar claro esse é o esquema em linguagem mais "humana", na pratica essas tabelas funcionam mais ou menos assim (exemplo de usuários e roles que são `Many : Many`):*
>
> 
>

Rotas universais

Antes de mais nada todas as classes dentro do banco tem as seguintes rotas padrões:

`POST /<classe>/{id}`

Cria objeto dentro do banco.

`GET /<classe>/{id}`

Busca o objeto especifico no banco e mostra todos os campos que podem ser exibidos.

`GET /<classe>/{id}/filter?{prop1}&{prop2}&{propN}`

Essa rota busca o objeto especifico, mas nela você pode filtrar apenas os campos que vai precisar.

PUT /<classe>/{id}

Edita as propriedades de um objeto (apenas os campos citados serão atualizados).

DELETE /<classe>/{id}

Deleta um objeto especifico.

Dependencia entre classes

Basicamente são as rotas para classes que tem dependentes ou que dependem de outra classe, exemplo, a classe `<child>` (Filho) depende da classe `<parent>` (Pai).

E nesse caso em especifico temos 3 tipos basicos, um em que o `parent` tem apenas 1 `child` (**One : One**), um em que o `parent` tem mais de 1 `child` e vise-e-versa (**One : Many** ou **Many : One**), e quando um `parent` tem mais de 1 `child` e `child` tem mais de um `parent` (**Many : Many**)

One : One

GET /<parent>/<child>

Vai retornar a `child` inteira

GET /<parent>/<child>/filter?={prop1}&{prop2}&{propN}

Retorna campos especificos da `child`

PUT /<parent>/<child>/{child id}

Seta a `child` do `parent`

DELETE /<parent>/<child>

"Deserda" a `child`

O mesmo vai funcionar se trocar o prefixo por `/<child>/<parent>`

One : Many

`GET /<parent>/<child>`

Vai retornar a lista de `child`s

`GET /<parent>/<child>/filter?={prop1}&{prop2}&{propN}`

Retorna campos especificos das `child`s

`PUT /<parent>/<child>/{child id}`

Seta a `child` do `parent`

`DELETE /<parent>/<child>/{child id}`

"Deserda" a `child`

> *Para o prefixo `/<child>/<parent>` valem as regras de **One : One** se o `<parent>` tem apenas uma `<child>`*

E Vice-E-Versa.

Many : Many

Mesmas regras do `One : Many` mas unilateral, onde `/<parent>/<child>` e `/<child>/<parent>` usam as regras do `One : Many`