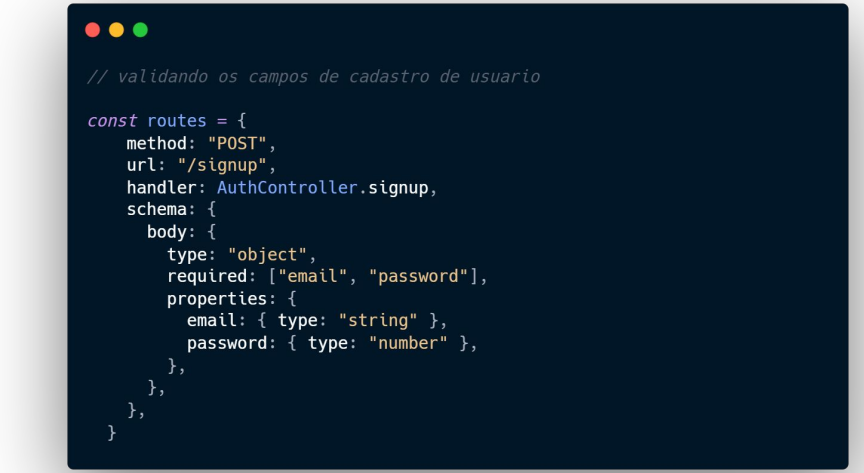

otterwise

Validação de Campos

Validação de Campos

Para que não seja necessário testarmos se todos os campos do *body* de uma requisição estão de acordo com o que a api espera, os frameworks implementam uma forma de validar esses campos através de um “schema”.



fonte:
https://www.fastify.io/docs/latest/Reference/Validation-and-Serialization/

otterwise

Exercício

Inserir validação nas rotas do CRUD de tarefas feito na aula passada.

Filtros e Ordenação

Filtro

Permite inserir filtros em uma query de consulta através da palavra ***where.***

No exemplo ao lado a query que será realizada trará todos os posts onde o autor tem a palavra “prisma.io” contida no seu email

```
const res = await prisma.post.findMany({
  where: {
    author: {
      email: {
        contains: 'prisma.io',
      },
    },
  },
})
```

Fontes: [Filtering and sorting (Concepts) | Prisma Docs]

otterwise

Ordenação

Através da palavra ***orderBy*** é possível ordenar uma lista de registros por um ou mais campos.

O exemplo ao lado retorna todos os registros de usuários ordenados por seu cargo e nome, e a lista de posts de cada usuário ordenada pelo título em ordem decrescente.

```
const usersWithPosts = await prisma.user.findMany({
  orderBy: [
    {
      role: 'desc',
    },
    {
      name: 'desc',
    },
  ],
  include: {
    posts: {
      orderBy: {
        title: 'desc',
      },
      select: {
        title: true,
      },
    },
  },
})
```

Fonte: [Filtering and sorting (Concepts) | Prisma Docs]

otterwise

Exercício

1. Utilizando a API de tarefas implementadas na aula passada, adicione um filtro para que a o usuário possa fazer uma busca entre o título das tarefas. (ou por e-mail se for o exemplo-prima)
2. Cria a opção de ordenar as tarefas por data e ordem alfabética decrescente. (nos títulos caso seja o exemplo-prisma)

Dica: utilize o artigo abaixo para implementar o padrão nas suas rotas.

(Parte 5) Paginação, Ordenação e Filtros em APIs RESTful | by Thiago Lima | thiagolima_br

Seeders & Migrations

Seeders

Serve para alimentar a base de dados com dados. Geralmente é usado para popular a base com dados padrões, necessários pro funcionamento correto da aplicação.

fonte: https://pt.stackoverflow.com/questions/207501/o-que-%C3%A9-seeder-e-migration

Exemplo

Criação de seeders utilizando Prisma ORM
https://www.prisma.io/docs/guides/database/seed-database#example-seed-scripts

Migrations

Migration é a definição que se dá ao gerenciamento de mudanças incrementais e reversíveis em esquemas (estrutura) de banco de dados. Isso permite que seja possível ter um controle "das versões" do banco de dados.

As migrations são executadas sempre que for necessário atualizar a estrutura do banco ou reverter as alterações para uma migration antiga.

Fonte: https://pt.stackoverflow.com/questions/207501/o-que-%C3%A9-seeder-e-migration

Exemplo

1. Criar a primeira migration rodando o comando *npx prisma migrate dev --name first-migration*
2. Mudar o tamanho máximo de um conteúdo de um post para 200 caracteres e rodar o comando *npx prisma migrate dev*

https://www.prisma.io/docs/guides/database/developing-with-prisma-migrate

Paginação

Paginação

Quando é necessário fazer a listagem de uma grande quantidade de dados através de uma rota, é necessário incluir a paginação para que a rota não fique sobrecarregada.

A paginação consiste em dividir a listagem de um retorno em partes, ou melhor, páginas. O cliente que está requisitando o recurso pode controlar que página e tamanho que cada página pode ter.

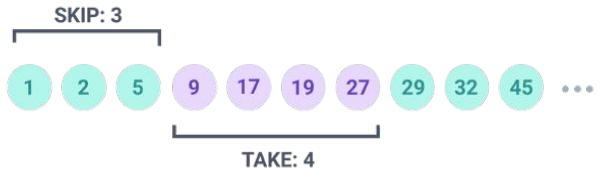
A utilização de paginação faz com que seja exigido menos recurso da máquina em que está hospedado o backend e também do banco de dados, também melhorando consideravelmente a velocidade de retorno da listagem se houver muitos registros na tabela requisitada.

Fonte: (Parte 5) Paginação, Ordenação e Filtros em APIs RESTful | by Thiago Lima | thiagolima_br

Tipos de Paginação

O Prisma ORM dá suporte a offset pagination e cursor-based pagination.

Offset pagination usa *skip* e *take* para pular um certo número de resultados e selecionar um limite de dados. Abaixo um exemplo de query que pula os 3 primeiros posts em uma lista de posts e retorna os 4 seguintes registros:




```
const results = await prisma.post.findMany({
  skip: 3,
  take: 4,
})
```

Fonte:
https://www.prisma.io/docs/concepts/components/prisma-client/pagination

otterwise

Tipos de Paginação

Cursor-based usa *cursor* e *take* para retornar um número limitado de resultados antes ou depois de um cursor. O cursor marca o local em uma listagem e deve ser único e sequencial, como um *id* ou um *timestamp.*
O código ao lado retorna os 4 primeiros posts que contém a palavra “prisma” e save o id do ultimo post como “myCursor”.

O diagrama abaixo mostra os ids dos 4 primeiros resultados, ou página 1. O cursor para a próxima página é 29.


```
const firstQueryResults = prisma.post.findMany({
  take: 4,
  where: {
    title: {
      contains: 'Prisma' // filtro opcional
    },
  },
  orderBy: {
    id: 'asc',
  },
})

const lastPostInResults = firstQueryResults[3]
const myCursor = lastPostInResults.id // Exemplo: 29
```

Fonte:
https://www.prisma.io/docs/concepts/components/prisma-client/pagination

otterwise

Vantagens e desvantagens de Offset Pagination

Vantagens:

- Você pode pular para qualquer página imediatamente. Por exemplo, você pode pular os 200 primeiros registros e pegar apenas 10, o que simula pular direto para a página 21 da listagem. Isso não é possível com o *cursor-based pagination.*
- Você pode paginar a listagem de registros em qualquer ordem. Por exemplo, você pode pular para a página 21 de uma listagem de usuários ordenados pelo nome. Isso não é possível com *cursor-based pagination,* que precisa ordenar por um valor único e sequencial.

Desvantagens:

- **Não escala** a nível de banco de dados. Por exemplo, se você pular 200.000 registros e pegar os próximos 10, o banco de dados ainda precisa passar pelos 200k registros antes de retornar os próximos 10, o que afeta bastante a performance de forma negativa.

Fonte: https://www.prisma.io/docs/concepts/components/prisma-client/pagination

Vantagens e desvantagens de Cursor-based pagination

Vantagens:

- **Escala** a nível de banco de dados. O SQL gerado pelo ORM não utiliza OFFSET, mas lista todos os registros com um ID maior que o valor do cursor.

Desvantagens:

- Você precisa ordenar sua query pelo seu cursor, que precisa ser único e sequencial.
- Você não pode pular até uma página específica usando apenas um cursor. Por exemplo, você não pode predizer qual cursor representa o início da página 400 (tamanho de página 20) sem fazer a requisição das páginas 1 - 399.

Fonte: https://www.prisma.io/docs/concepts/components/prisma-client/pagination

Quando utilizar cada tipo de paginação

Offset pagination: paginação de poucos registros. Por exemplo, um blog que permite o usuário filtrar posts por autor e paginar os resultados.

Cursor-based pagination: scroll infinito. Por exemplo, ordenar posts de um blog por data de forma descendente e listar 10 posts por vez. Outro exemplo seria paginar resultados de um grande lista em páginas menores, pois o *offset pagination* não lida bem com uma lista muito grande de registros.

Fonte: https://www.prisma.io/docs/concepts/components/prisma-client/pagination

Exercício

Utilizando a API de tarefas implementadas na aula passada, adicione paginação no formato de *offset pagination.* Defina o tamanho de página padrão caso o usuário não mande nenhum parâmetro na requisição.

Dica: utilize o artigo abaixo para implementar o padrão nas suas rotas.

(Parte 5) Paginação, Ordenação e Filtros em APIs RESTful | by Thiago Lima | thiagolima_br

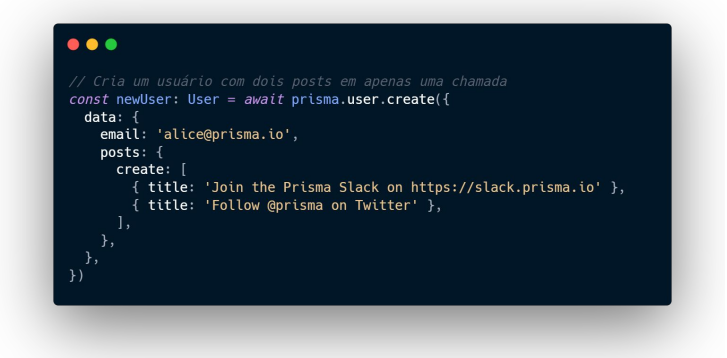
Transações

Transações

Uma transação em banco de dados se refere a uma sequência de operações de leitura/escrita que garantem o sucesso ou falha como um todo.

O Prisma ORM resolve isso de duas formas:

- **Nested queries:** realizar mais de uma operação em apenas uma chamada a API do Prisma.



```
// Cria um usuário com dois posts em apenas uma chamada
const newUser: User = await prisma.user.create({
  data: {
    email: 'alice@prisma.io',
    posts: {
      create: [
        { title: 'Join the Prisma Slack on https://slack.prisma.io' },
        { title: 'Follow @prisma on Twitter' },
      ],
    },
  },
})
```

Fonte: https://www.prisma.io/docs/concepts/components/prisma-client/transactions

Transações

- **$transaction API:** essa API permite encapsular mais de uma chamada de api em uma transação que só dará como "pronto" se todas as chamadas de API derem certo. Se houver algum erro, a transação é desfeita e não é feita nenhuma modificação no banco de dados.

```
// A query abaixo retorna todos os posts que condizem com o filtro e
também realiza a contagem de todos os posts:
const [posts, totalPosts] = await prisma.$transaction([
  prisma.post.findMany({ where: { title: { contains: 'prisma' } } }),
  prisma.post.count(),
])
```

Fonte: https://www.prisma.io/docs/concepts/components/prisma-client/transactions