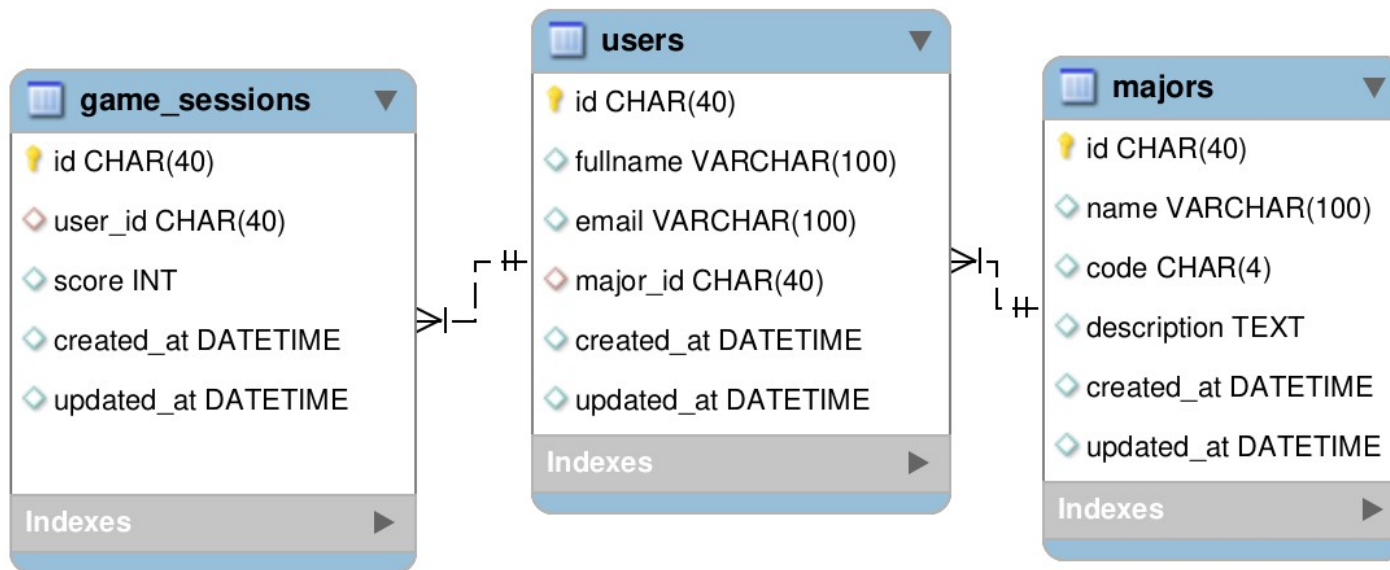




**Prof. David Fernandes de Oliveira**  
**Instituto de Computação**  
**UFAM**

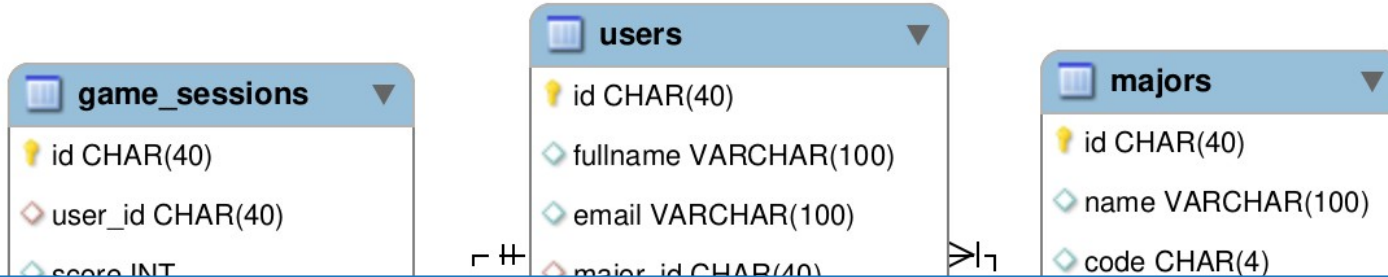
# Modelando a Aplicação

- A aplicação irá conter 3 tabelas: **users**, **game\_sessions** (partida do jogo), **majors** (curso universitário)



# Modelando a Aplicação

- A aplicação irá conter 3 tabelas: **users**, **game\_sessions** (partida do jogo), **majors** (curso universitário)



## Descrição das Tabelas:

- **Tabela users:** dados dos usuários da aplicação
- **Tabela game\_sessions:** dados de cada partida do jogo
- **Tabela marjors:** cursos (cc, es, ec, etc) dos jogadores



# Criando o Banco de Dados

- Neste ponto, é importante que o banco de dados e o usuário MySQL já estejam criados no SGBD local
- Caso não estejam, você pode usar comandos abaixo para criá-los, mantendo o usuário e senha conforme informado

```
mysql -uroot -p --si ~
david@coyote ~ $ mysql -uroot -p --silent
Enter password:
mysql> CREATE DATABASE game;
mysql> CREATE USER 'game'@'localhost' IDENTIFIED BY 'senha123';
mysql> GRANT ALL PRIVILEGES ON game.* TO 'game'@'localhost';
mysql>
mysql>
```

# Criando o Banco de Dados

- Neste ponto, é importante que o banco de dados e o usuário MySQL já estejam criados no SGBD local
- Caso não estejam, você pode usar comandos abaixo para criá-

Além do cliente de linha de comando do MySQL, também é possível criar o banco e o usuário através de aplicações como **phpMyAdmin** e **MySQL Workbench**

```
Enter password:
mysql> CREATE DATABASE game;
mysql> CREATE USER 'game'@'localhost' IDENTIFIED BY 'senha123';
mysql> GRANT ALL PRIVILEGES ON game.* TO 'game'@'localhost';
mysql>
mysql> █
```

# Criando o Banco de Dados

- Neste ponto, é importante que o banco de dados e o usuário MySQL já estejam criados no SGBD local
- Caso não estejam, você pode usar comandos abaixo para criá-

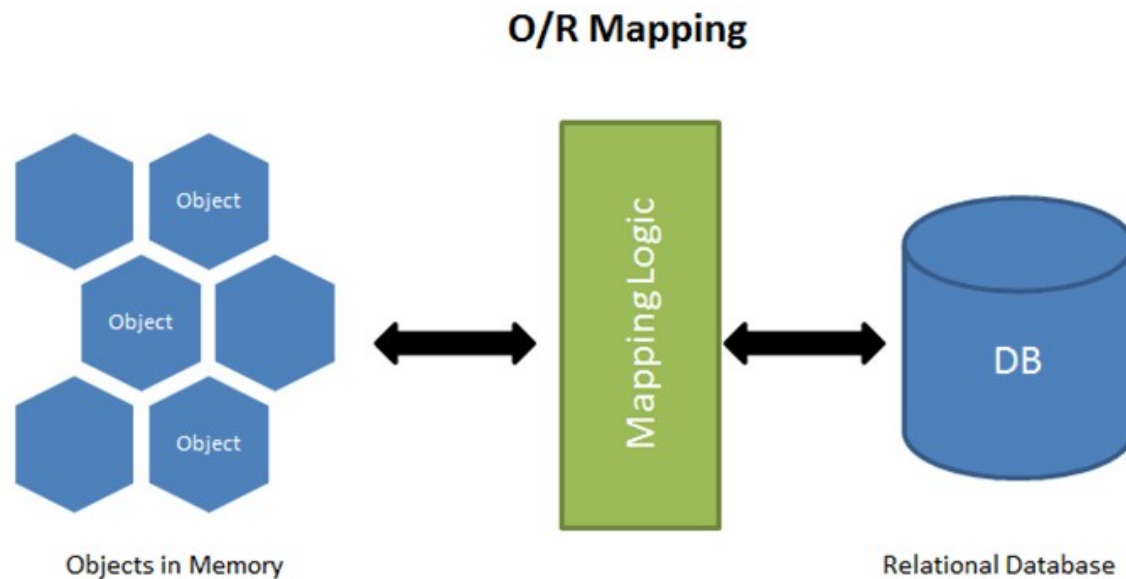
Além do cliente de linha de comando do MySQL, também é possível criar o banco de dados e o usuário através de aplicações como

Vale salientar que uma senha simples como a de nosso exemplo **\*\*não pode\*\*** ser usada em produção

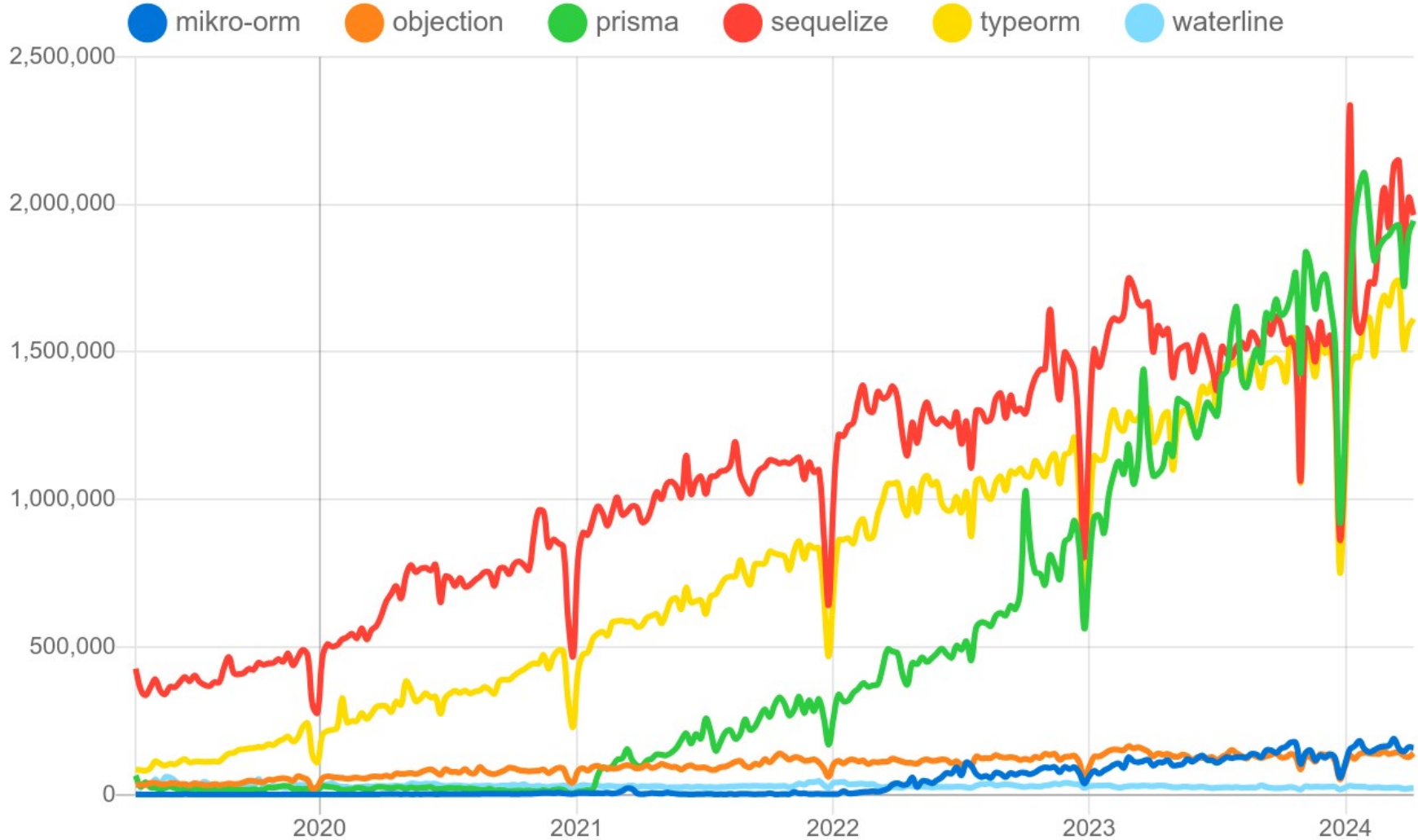
```
mysql> CREATE DATABASE game;
mysql> CREATE USER 'game'@'localhost' IDENTIFIED BY 'senha123';
mysql> GRANT ALL PRIVILEGES ON game.* TO 'game'@'localhost';
mysql>
mysql>
```

# Object Relational Mapper - ORM

- ORM é uma técnica que permite **consultar e manipular dados** de um database usando o paradigma de orientação a objetos
- Desta forma, o acesso aos dados não é feito através da linguagem SQL, e sim através de objetos

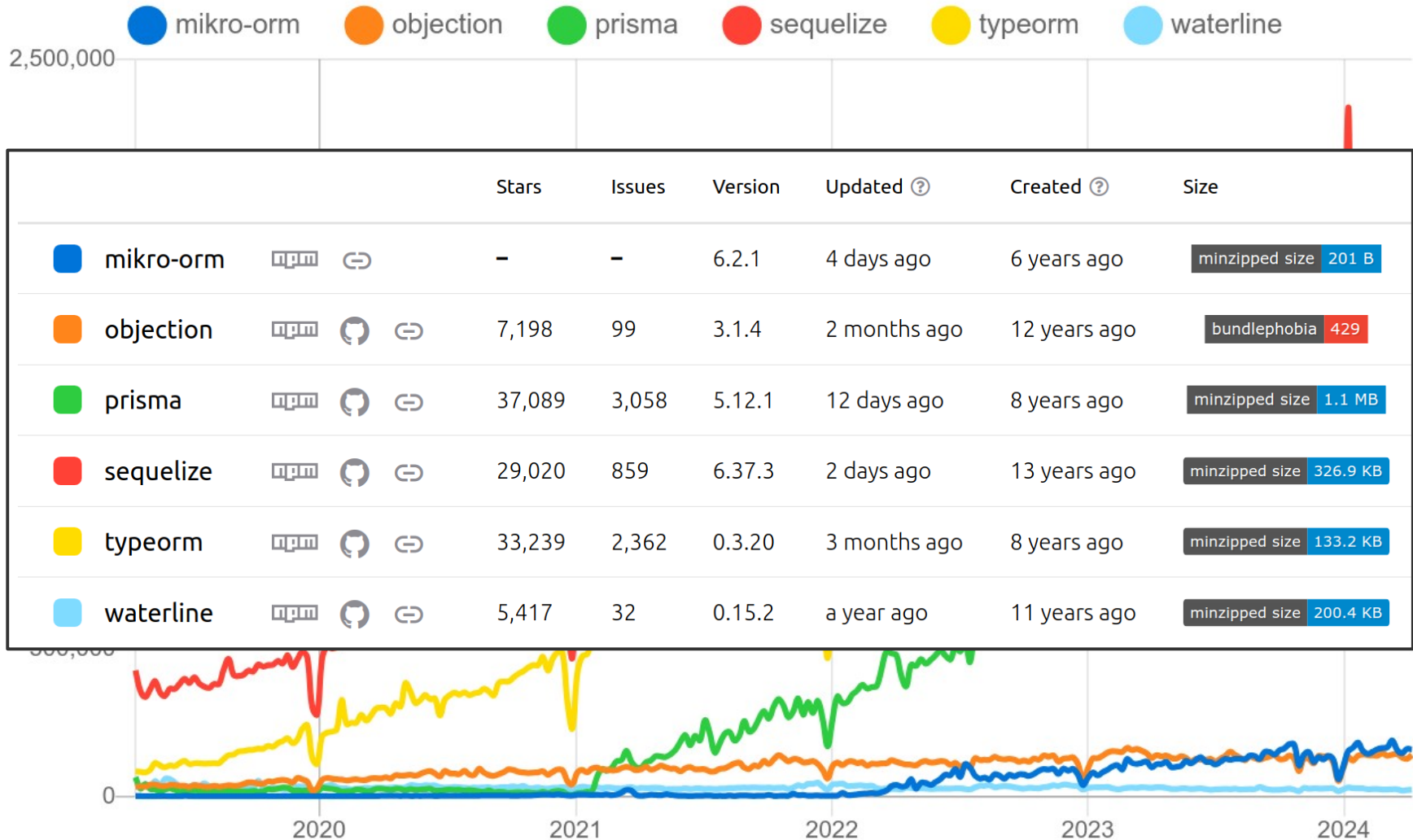


# Escolhendo o ORM

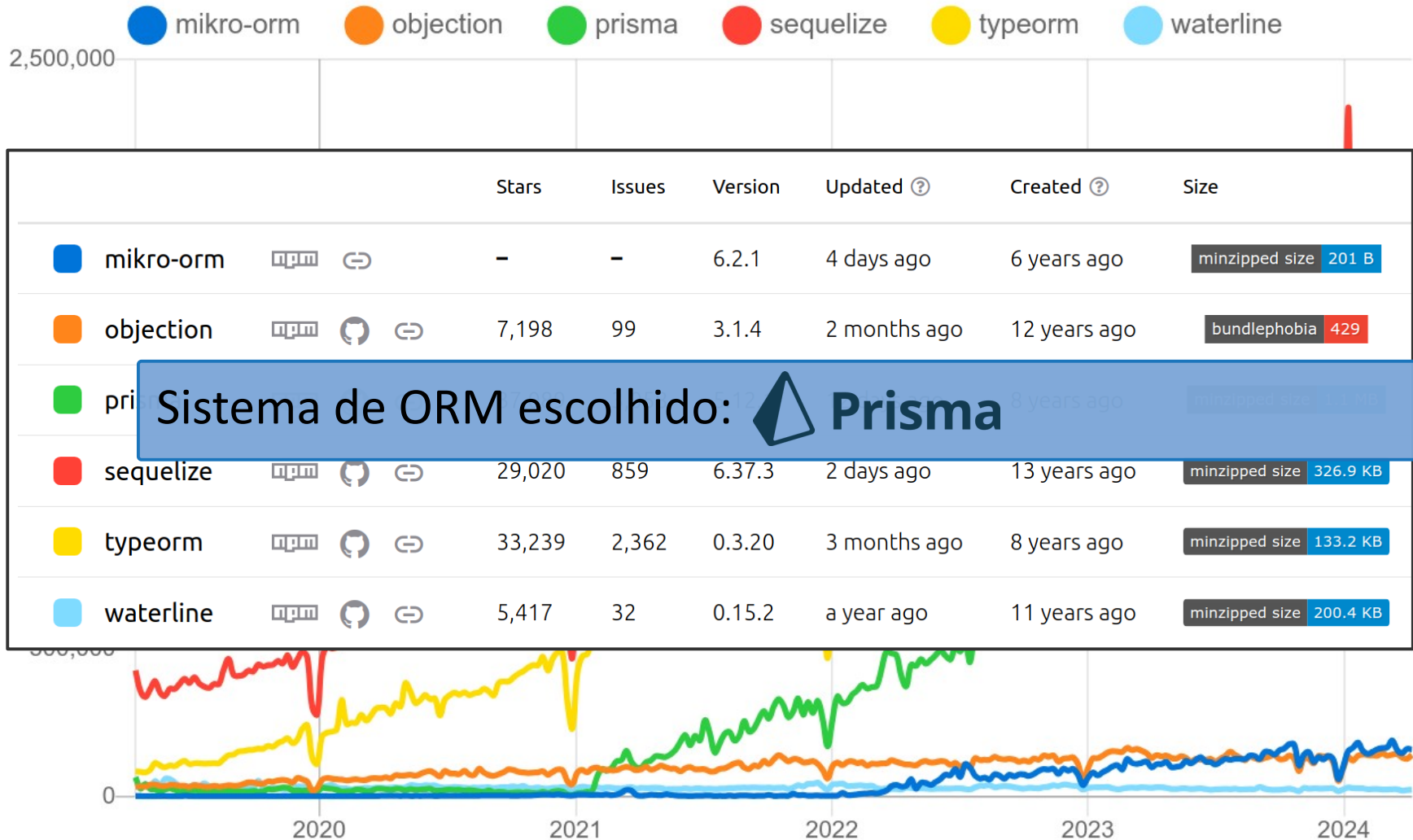




# Escolhendo o ORM



# Escolhendo o ORM





- Use os comandos abaixo para instalar o **Prisma** como dependência de desenvolvimento

```
$ npm install -D prisma  
$ npm install @prisma/client
```

- Após isso, execute o comando abaixo para que o Prisma crie os arquivos iniciais de configuração

```
$ npx prisma init --datasource-provider mysql
```

```
david@coyote ~/dev/game $ npx prisma init --datasource-provider mysql
```

✓ Your Prisma schema was created at `prisma/schema.prisma`  
You can now open it in your favorite editor.

■ Use os comandos abaixo para instalar o Prisma como dependência de seu projeto.  
warn You already have a `.gitignore` file. Don't forget to add ``.env`` in it to not commit any private information.

Next steps:

1. Set the `DATABASE_URL` in the `.env` file to point to your existing database. If your database has no tables yet, read <https://pris.ly/d/getting-started>
2. Run `prisma db pull` to turn your database schema into a Prisma schema.
3. Run `prisma generate` to generate the Prisma Client. You can then start querying your database.

```
$ npx prisma init --datasource-provider mysql
```

More information in our documentation:

<https://pris.ly/d/getting-started>

Developing real-time features?

Prisma Pulse lets you respond instantly to database changes.

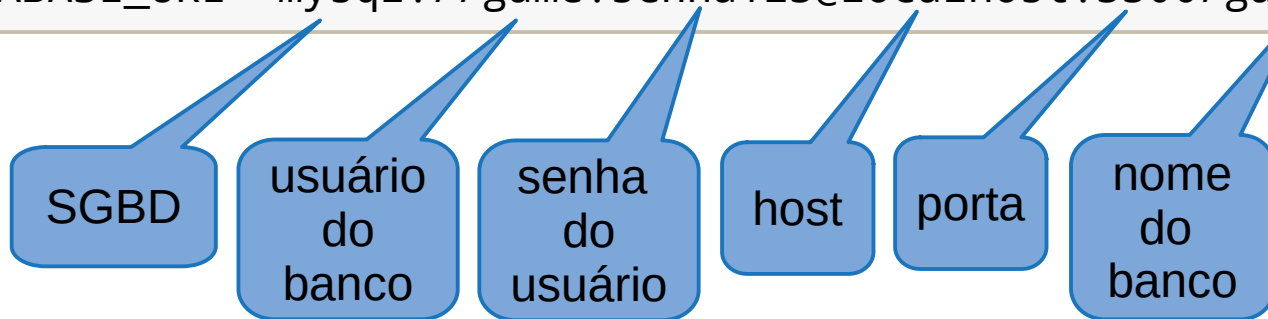
<https://pris.ly/cli/pulse>

```
david@coyote ~/dev/game $
```



- O comando **prisma init** adiciona uma variável `DATABASE_URL` no arquivo **.env**, contendo uma string de conexão com o banco
- Será preciso editar o valor dessa variável conforme a realidade do banco de dados utilizado

```
DATABASE_URL="mysql://game:senha123@localhost:3306/game"
```



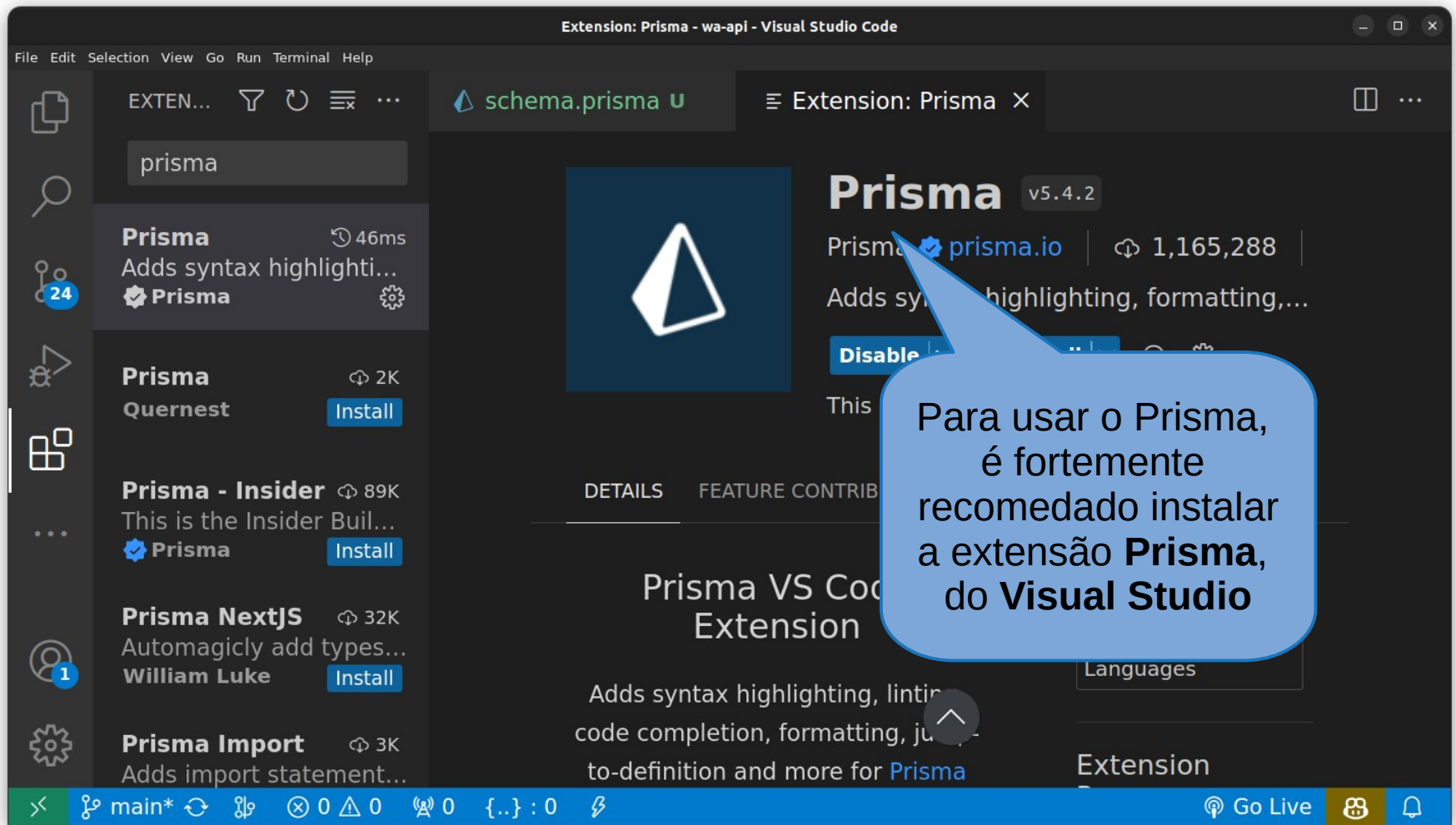


- O comando **prisma init** também cria um arquivo **prisma/schema.prisma**, onde serão criados os modelos da aplicação
- Nesse arquivo, é importante alterar o **provider** para **mysql**, conforme mostrado abaixo

```
generator client {  
  provider = "prisma-client-js"  
}  
  
datasource db {  
  provider = "mysql"  
  url      = env("DATABASE_URL")  
}
```

Mudar o  
**provider**  
para mysql

A blue callout bubble with a tail pointing to the "mysql" value in the Prisma schema code block above.



Para usar o Prisma,  
é fortemente  
recomendado instalar  
a extensão **Prisma**,  
do **Visual Studio**

# Adicionando um modelo

- Ainda no arquivo **prisma/schema.prisma**, vamos adicionar o primeiro modelo de nossa aplicação

```
model Major {  
  id          String    @id @default(uuid()) @db.Char(40)  
  name        String    @unique @db.VarChar(100)  
  code        String    @unique @db.Char(4)  
  description  String    @db.Text()  
  createdAt   DateTime  @default(now()) @map("created_at")  
  updatedAt   DateTime  @updatedAt @map("updated_at")  
  
  @@map("majors")  
}
```



# Adicionando um modelo

- Após isso, usamos o comando **npx prisma migrate dev --name** para gerar a **migração** e criar a tabela **majors**

```
~ /d/game
david@coyote ~/dev/game $ npx prisma migrate dev --name create-major-table 1
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
Datasource "db": MySQL database "game" at "localhost:3306"

Applying migration `20240708164701_create_major_table`

The following migration(s) have been created and applied from new schema changes:

migrations/
├─ 20240708164701_create_major_table/
└─ migration.sql

Your database is now in sync with your schema.

✓ Generated Prisma Client (v5.16.1) to ./node_modules/@prisma/client in 55ms

david@coyote ~/dev/game $
```

# Adicionando um modelo

- Após isso, usamos o comando **npx prisma migrate dev --name** para gerar a **migração** e criar a tabela **majors**

```
~ /d/game
david@coyote ~/dev/game $ npx prisma migrate dev --name create-major-table
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
```

Tabela	Ação	Registos	Tipo
<input type="checkbox"/> majors	★ Procurar Estrutura Pesquisar Inserir Limpa Eliminar	0	InnoDB
<input type="checkbox"/> _prisma_migrations	★ Procurar Estrutura Pesquisar Inserir Limpa Eliminar	1	InnoDB
2 tabelas	Soma	1	InnoDB

```
└─ 20240708164701_create_major_table/
  └─ migration.sql
```

Your database is now in sync with your schema.

✓ Generated **Prisma Client** (v5.16.1) to ./node\_modules/@prisma/client in 55ms

```
david@coyote ~/dev/game $
```

# Adicionando um modelo

- Após isso, usamos o comando **npx prisma migrate dev --name** para gerar a **migração** e criar a tabela **majors**

	#	Nome	Tipo	Agrupamento (Collation)	Atributos	Nulo	Predefinido
<input type="checkbox"/>	1	id 🔑	char(40)	utf8mb4_unicode_ci		Não	Nenhum
<input type="checkbox"/>	2	name 🔑	varchar(100)	utf8mb4_unicode_ci		Não	Nenhum
<input type="checkbox"/>	3	code 🔑	char(4)	utf8mb4_unicode_ci		Não	Nenhum
<input type="checkbox"/>	4	description	text	utf8mb4_unicode_ci		Não	Nenhum
<input type="checkbox"/>	5	created_at	datetime(3)			Não	CURRENT_TIMESTAMP(3)
<input type="checkbox"/>	6	updated_at	datetime(3)			Não	Nenhum

✓ Generated **Prisma Client** (v5.16.1) to ./node\_modules/@prisma/client in 55ms

david@coyote ~/dev/game \$

# Exercício 1

Tabela	Ação	Registos	Tipo
<input type="checkbox"/> majors	★ Procurar Estrutura Pesquisar Inserir Limpa Eliminar	0	InnoDB
<input type="checkbox"/> _prisma_migrations	★ Procurar Estrutura Pesquisar Inserir Limpa Eliminar	1	InnoDB

#	Nome	Tipo	Agrupamento (Collation)	Atributos	Nulo	Predefinido
<input type="checkbox"/> 1	id 🔑	char(40)	utf8mb4_unicode_ci		Não	Nenhum
<input type="checkbox"/> 2	name 🔑	varchar(100)	utf8mb4_unicode_ci		Não	Nenhum
<input type="checkbox"/> 3	code 🔑	char(4)	utf8mb4_unicode_ci		Não	Nenhum
<input type="checkbox"/> 4	description	text			Não	Nenhum
<input type="checkbox"/> 5	created_at	datetime(3)			Não	CURRENT_TIMESTAMP(3)
<input type="checkbox"/> 6	updated_at	datetime(3)			Não	CURRENT_TIMESTAMP(3)

Instale e configure o **Prisma** em sua aplicação. Após isso, crie o modelo **Major** e rode a migration para que a tabela majors seja criada no banco de dados.

github  
**ExpTS**



express JS

# Encontrando registros

- Para recuperar registros no banco de dados:

```
import { PrismaClient, User } from "@prisma/client";  
const prisma = new PrismaClient();  
const users: User[] = await prisma.user.findMany(criteria)
```

- Exemplos:

```
SELECT * FROM user WHERE nome = 'Carlos':
```

```
await prisma.user.findMany({ where: { nome: 'Carlos' } });
```

```
SELECT nome, idade FROM user WHERE nome = 'Carlos':
```

```
await prisma.user.findMany({  
  where: { nome: 'Carlos' },  
  select: { nome: true, idade: true }  
});
```

# Encontrando registros

- Exemplos de consultas com o objeto **Sequelize.Op**

```
SELECT * FROM user WHERE idade < 30:
```

```
await prisma.user.findMany({ where: { idade: { lt: 30 } } });
```

```
SELECT * FROM user WHERE idade >= 21:
```

```
await prisma.user.findMany({ where: { idade: { lte: 21 } } });
```

```
SELECT * FROM user WHERE nome IN ('Carlos', 'Maria'):
```

```
await prisma.user.findMany({  
  where: { nome: { in: ['Carlos', 'Maria'] } }  
});
```

```
SELECT * FROM user WHERE nome NOT IN ('Carlos', 'Maria'):
```

```
await prisma.user.findMany({  
  where: { nome: { notIn: ['Carlos', 'Maria'] } }  
});
```

# Encontrando registros

- Selecionando usuários cujo nome possuem a string Carlos

```
SELECT * FROM user WHERE nome like '%Carlos%':
```

```
await prisma.user.findMany({  
  where: { nome: { contains: 'Carlos' } }  
});
```

- Selecionando usuários cujo nome começam com Carlos

```
SELECT * FROM user WHERE nome like 'Carlos%':
```

```
await prisma.user.findMany({  
  where: { nome: { startsWith: 'Carlos' } }  
});
```

- Selecionando usuários cujo nome terminam com Oliveira

```
SELECT * FROM user WHERE nome like '%Oliveira':
```

```
await prisma.user.findMany({  
  where: { nome: { endsWith: 'Oliveira' } }  
});
```

# Paginação

- As cláusulas **skip** e **take** podem ser usadas em conjunto para contruir um sistema de paginação

- Selecionando os usuários 1-10 com idade maior que 20

```
SELECT * FROM user WHERE idade > 20 LIMIT 10 OFFSET 0;
```

```
await prisma.user.findMany({  
  where: { idade: { gt: 20 }},  
  skip: 0, take: 10  
});
```

- Selecionando os usuários 11-20 com idade maior que 20

```
SELECT * FROM user WHERE idade > 20 LIMIT 10 OFFSET 10;
```

```
await prisma.user.findMany({  
  where: { idade: { gt: 20 }},  
  skip: 10 , take: 10  
});
```



# Ordenando Registros

- Os resultados recuperados podem ser ordenados de forma crescente (ASC) ou decrescente (DESC) por qualquer atributo
- Usuários >=18 ordenados pelo nome em ordem crescente

```
SELECT * FROM user WHERE idade >= 18 ORDER BY nome ASC;
```

```
await prisma.user.findMany({  
  where: { idade: { gte: 18 }}, orderBy: { nome, 'asc' }  
});
```

- Usuários >=18 ordenados pelo nome em ordem decrescente

```
SELECT * FROM user WHERE idade >= 18 ORDER BY nome DESC;
```

```
await prisma.user.findMany({  
  where: { idade: { gte: 18 }}, orderBy: { nome, 'desc' }  
});
```

# Selecionando apenas um registro

- O comando **findMany()** retorna um array com todos os registros que atenderem os critérios da cláusula **where**
  - Note que, se apenas um registro atender aos critérios da cláusula **where**, será retornado um array com uma única posição
- Uma alternativa é o uso de **findFirst()**, que ao invés de recuperar um array, retorna apenas um objeto

```
SELECT * FROM user WHERE idade >= 18;
```

```
await prisma.user.findFirst({ where: { idade: { lte: 18 } } })
```

# Criando novos registros

- Criação de novos registros no banco de dados:

```
await prisma.user.create(data: { initialValues } );
```

- Exemplo:

```
INSERT INTO user (nome, idade) VALUES ('Carlos', 26);
```

```
await prisma.user.create({ data: {nome: 'Carlos', idade: 16}})
```

# Atualizando registros existentes

- Atualização de registros já existentes no banco de dados:

```
await prisma.user.update({values},{criteria});
```

- Exemplo:

```
UPDATE user SET idade=30 WHERE nome = 'Carlos';
```

```
await prisma.user.update(  
  { idade: 30 },  
  { where: { nome: 'Carlos' } }  
);
```

# Apagando registros existentes

- Apagando registros no banco de dados:

```
await prisma.user.deleteMany(criteria);
```

- Exemplos:

```
DELETE FROM user WHERE nome = 'Carlos';
```

```
await prisma.user.deleteMany({  
  where: { nome: 'Carlos' }  
});
```

```
DELETE FROM user WHERE id IN (3, 97);
```

```
await prisma.user.deleteMany({  
  where: { id: { in: [3, 97] } }  
});
```

# Desenvolvendo CRUDs

- Uma parte importante do desenvolvimento de uma aplicação é a criação do **CRUD** de alguns modelos
  - O CRUD de um modelo é um conjunto de páginas responsáveis por quatro operações sobre esse modelo:



# Desenvolvendo CRUDs

- Para exemplificar a criação de novos CRUDs, vamos desenvolver um para o modelo **Major**
- O primeiro passo é criar um controlador vazio para o CRUD de **Major** dentro do diretório **/src/controllers**

```
// Arquivo src/controllers/major.ts
```

```
import { Request, Response } from 'express'
```

```
const index = (req: Request, res: Response) => {}
```

```
const create = (req: Request, res: Response) => {}
```

```
const read = (req: Request, res: Response) => {}
```

```
const update = (req: Request, res: Response) => {}
```

```
const remove = (req: Request, res: Response) => {}
```

```
export default { index, read, create, update, remove }
```

Operações  
do CRUD

# Definindo as Rotas do CRUD

- O segundo passo é definir as rotas para cada operação do CRUD:

```
// Arquivo src/router/router.ts

import express from "express"
import majorController from "../controllers/major"

const router = express.Router();
...
// MajorController
router.get('/major', majorController.index)
router.get('/major/read/:id', majorController.read)
router.get('/major/create', majorController.create)
router.post('/major/create', majorController.create)
router.get('/major/update/:id', majorController.update)
router.post('/major/update/:id', majorController.update)
router.post('/major/remove/:id', majorController.remove)
...
export default router
```



# Definindo as Rotas do CRUD

- O segundo passo é definir as rotas para cada operação do CRUD:

```
// Arquivo src/router/router.ts
```

```
import express from "express"
```

```
import majorController from "../controllers/major"
```

```
const router = express.Router();
```

```
...
```

```
// Rotas para o CRUD
```

Embora não faça parte do CRUD, o objetivo da rota **/major** é listar os cursos existentes

```
router.get('/major/create', majorController.create)
```

```
router.post('/major/create', majorController.create)
```

```
router.get('/major/update/:id', majorController.update)
```

```
router.post('/major/update/:id', majorController.update)
```

```
router.post('/major/remove/:id', majorController.remove)
```

```
...
```

```
export default router
```

# Definindo as Rotas do CRUD

- O segundo passo é definir as rotas para cada operação do CRUD:

```
// Arquivo src/router/router.ts
```

```
import express from "express"
```

```
import majorController from "../controllers/major"
```

```
const router = express.Router();
```

```
...
```

```
// Método GET para ler
```

```
router.get('/major/:id', majorController.read)
```

Note que as rotas para **read**, **update** e **remove** terminam com a string **:id**, que representa um parâmetro utilizado para informar que curso se deseja **ler**, **atualizar** ou **apagar**

```
router.get('/major/:id', majorController.read)
```

```
router.post('/major/update/:id', majorController.update)
```

```
router.post('/major/remove/:id', majorController.remove)
```

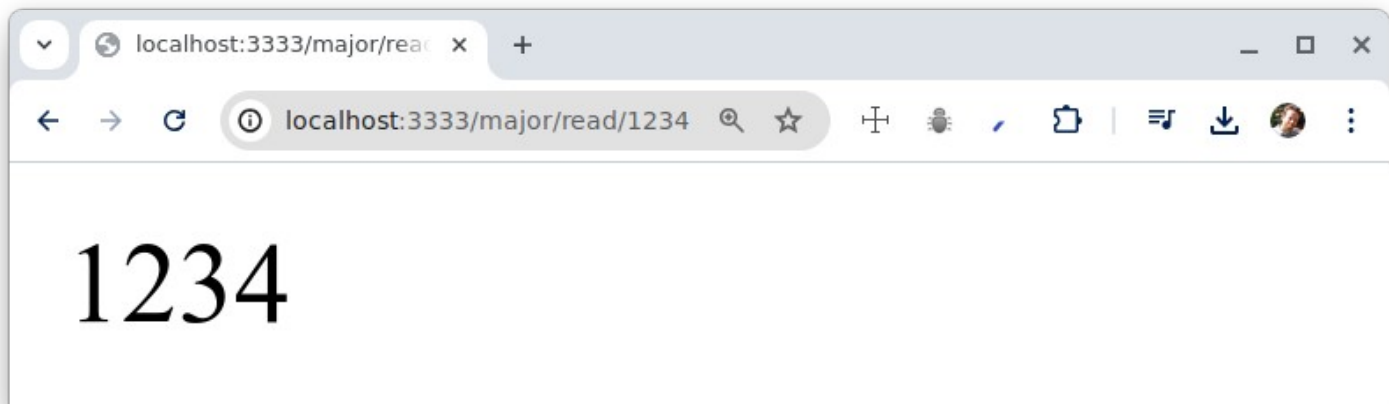
```
...
```

```
export default router
```

# Definindo as Rotas do CRUD

- Por exemplo, na url **http://localhost:3333/major/read/1234**, o valor do parâmetro **id** é 1234
- Para ler o valor de **id** dentro da action, podemos usar o atributo **param** de **req** (objeto da requisição do usuário):

```
const read = (req: Request, res: Response) => {  
  const { id } = req.params  
  res.send(id)  
}
```



# Requisições POST

- Quando o usuário preenche e submete um formulário POST, os dados informados pelo usuário são enviados para o servidor

```
<form action="/major/create" method="post">  
  <input type="text" name="code">  
  <input type="text" name="name">  
  <input type="submit" value="Enviar">  
</form>
```

- Após a submissão, os dados são enviados através do corpo da requisição HTTP (request body)
- Esses dados são enviados no formato **application/x-www-form-urlencoded**, que também é usado em requisições GET:

```
code=IE08&name=Ciencia+da+Computacao
```

# Requisições POST

- O Express possui um middleware nativo chamado **urlencoded**, que pode ser usado para extrair os dados de **request body**
- Para usá-lo, basta inserir a linha abaixo no arquivo **app.js**, em alguma lugar antes da chamada ao middleware router:

```
// Arquivo index.ts  
app.use(express.urlencoded({extended: false}));  
...  
app.use(router);
```

# Requisições POST

- Após isso, o **urlencoded** irá extrair os dados do **request body** de todas as requisições POST e copiá-los no objeto **req.body**

```
// Arquivo src/controllers/major.ts
import { createMajor } from '../major.service'

const create = async (req: Request, res: Response) => {
  if (req.method === 'GET') {
    res.render('major/create')
  } else {
    try {
      await createMajor(req.body)
      res.redirect('/major')
    } catch (err) {
      res.status(500).send(err)
    }
  }
}
```

Uso da função `createMajor`, que será definida na camada de serviço

# Camada de Serviços

- Os **serviços** têm como função orquestrar as regras de negócio e servir de intermediários entre controladores e modelos

```
// Arquivo src/services/major.ts

import { PrismaClient, Major } from '@prisma/client'
import { CreateMajorDto } from '../types/major'

const prisma = new PrismaClient()

export const getAllMajors = async(): Promise<Major[]> => {
  return prisma.major.findMany();
}

export const createMajor = async (
  newMajor: CreateMajorDto
): Promise<Major> => {
  return await prisma.major.create({ data: newMajor })
}
```

# Camada de Serviços

- Os **serviços** têm como função orquestrar as regras de negócio e servir de intermediários entre controladores e modelos

```
// Arquivo src/services/major.ts
```

```
import { PrismaClient, Major } from '@prisma/client'  
import { CreateMajorDto } from '../types/major'
```

Além das funções mostradas, o serviço major precisa ter funções como:

```
const majorAlreadyExists = async (name: string): Promise<boolean>
```

```
const getMajor = async (id: string): Promise<Major>
```

```
const updateMajor = async (id: string, major: MajorUpdateDto):  
  Promise<[affectedCount: number]>
```

```
const removeMajor = async (id: string): Promise<number>
```

```
  newMajor: CreateMajorDto  
)>: Promise<Major> => {  
  return await prisma.major.create({ data: newMajor })  
}
```



# Camada de Serviços

- Os **serviços** têm como função orquestrar as regras de negócio e servir de intermediários entre controladores e modelos

```
// Arquivo src/services/major.ts
```

```
import { PrismaClient, Major } from '@prisma/client'  
import { CreateMajorDto } from '../types/major'
```

Além das funções mostradas, o serviço major precisa ter funções como:

- Uma vantagem do uso de serviços é que suas funções podem ser utilizadas em outras partes da aplicação, diminuindo a réplica de códigos em vários arquivos.

**Promise**<[affectedCount: number]>

**const** removeMajor = **async** (id: string): **Promise**<number>

```
newMajor: CreateMajorDto  
) : Promise<Major> => {  
  return await prisma.major.create({ data: newMajor })  
}
```

# Camada de Serviços

- Os **serviços** têm como função orquestrar as regras de negócio e servir de intermediários entre controladores e modelos

```
// Arquivo src/services/major.ts  
  
import { PrismaClient, Major } from '@prisma/client'  
import { CreateMajorDto } from '../types/major'
```

Além das funções mostradas, o serviço major precisa ter funções como:

- Uma vantagem do uso de serviços é que suas funções podem ser
- Outra vantagem dos serviços é que, caso se queira mudar o ORM da aplicação, o esforço será muito menor. Isso porque eles serão os únicos arquivos que usam os recursos do ORM para recuperar, atualizar e criar dados.

```
const removeMajor = async (id: string): Promise<number>  
  newMajor: CreateMajorDto  
) : Promise<Major> => {  
  return await prisma.major.create({ data: newMajor })  
}
```

# Data Transfer Objects (DTO)

- Os arquivos **types/\*\*/\*types.ts** possuem as **interfaces** e **types**, em especial os **DTOs**, usados dentro dos services e controllers
- DTO é uma interface ou type usado para representar os objetos de dados que são trocados entre a API e as aplicações client
  - Por exemplo, para criar um novo produto, a aplicação cliente precisa enviar para a API os dados desse novo produto, sendo o formato desses dados é definido através de um DTO

# Data Transfer Objects (DTO)

- Os DTOs geralmente contêm um subconjunto dos atributos de um dado modelo, e para gerá-los podemos usar o comando Pick

```
// Arquivo types/major.ts  
  
import { Major } from '@prisma/client'  
  
export type CreateMajorDto =  
  Pick<Major, 'name' | 'code' | 'description'>  
export type UpdateMajorDto =  
  Pick<Major, 'name' | 'code' | 'description'>
```

Cria um novo type contendo apenas as propriedades name, code e description do modelo Major

# Data Transfer Objects (DTO)

- Os DTOs são usados principalmente na camada de serviço, mas também podem ser utilizados nos controladores

```
// Arquivo services/major.ts

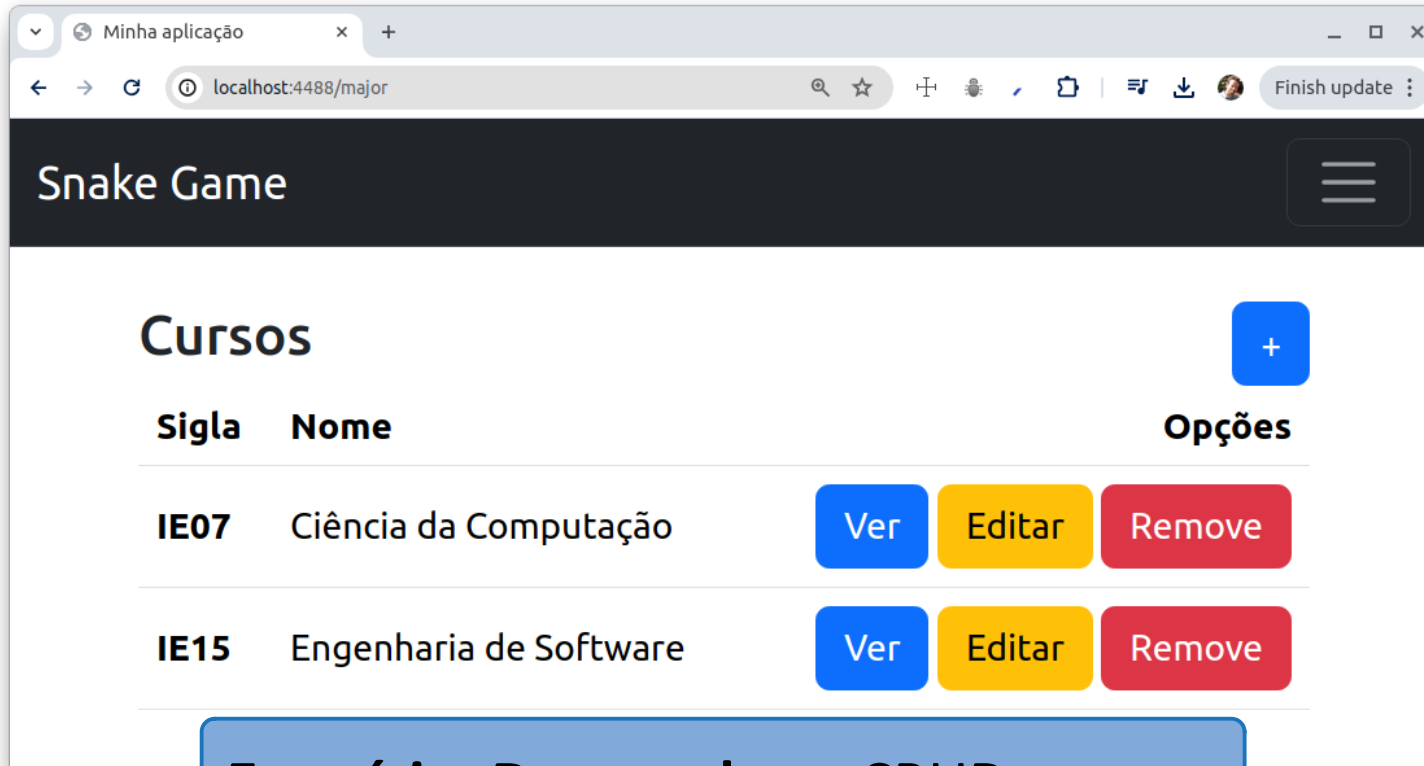
import { PrismaClient, Major } from '@prisma/client'
import { CreateMajorDto } from '../types/major'

const prisma = new PrismaClient()

export const getAllMajors = async(): Promise<Major[]> => {
  return prisma.major.findMany();
}

export const createMajor = async (
  newMajor: CreateMajorDto
): Promise<Major> => {
  return await prisma.major.create({ data: newMajor })
}
```

# Exercício 2



**Exercício:** Desenvolva o CRUD para o modelo **Major**. Além das funções **index**, **create** e **read** desenvolvidas durante as aulas, implemente também as funções **update** e **delete**.

github  
**ExpTS**

express **JS**

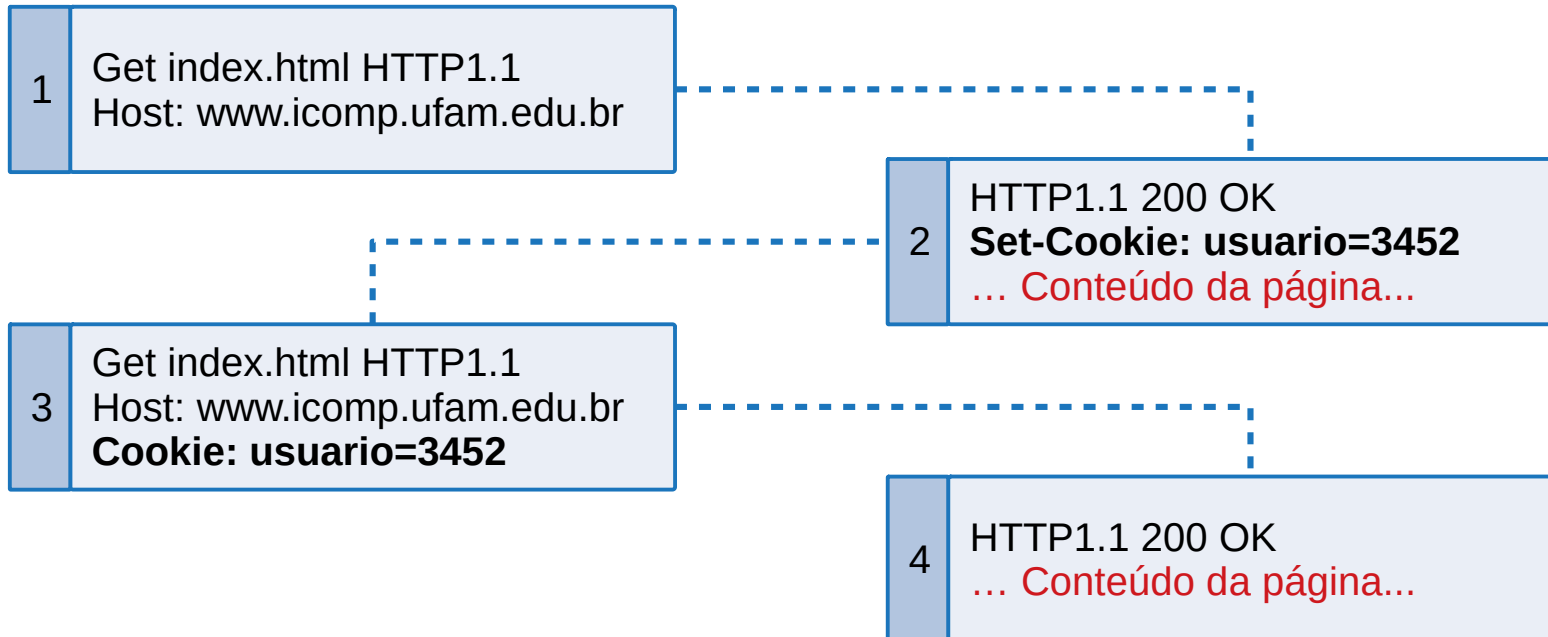
# Cookies



- **Cookies** são variáveis enviadas pelo servidor Web para o browser através do **protocolo HTTP**
  - Ficam armazenados no lado cliente
  - São enviados para o servidor em futuros acessos do browser

## Requisições do Browser

## Requisições de **icomp.ufam.edu.br**



# Cookies



- Para habilitar o uso de cookies em sua aplicação, é necessário instalar o middleware **cookie-parser**

```
$ npm install cookie-parser  
$ npm install -D @types/cookie-parser
```

- Para usar o middleware, precisamos importar o módulo e adicioná-lo em nossa aplicação com o método **use**

```
// Arquivo src/server.ts  
import cookieParser from 'cookie-parser';  
  
// Método middleware():  
app.use(cookieParser())
```



# Cookies



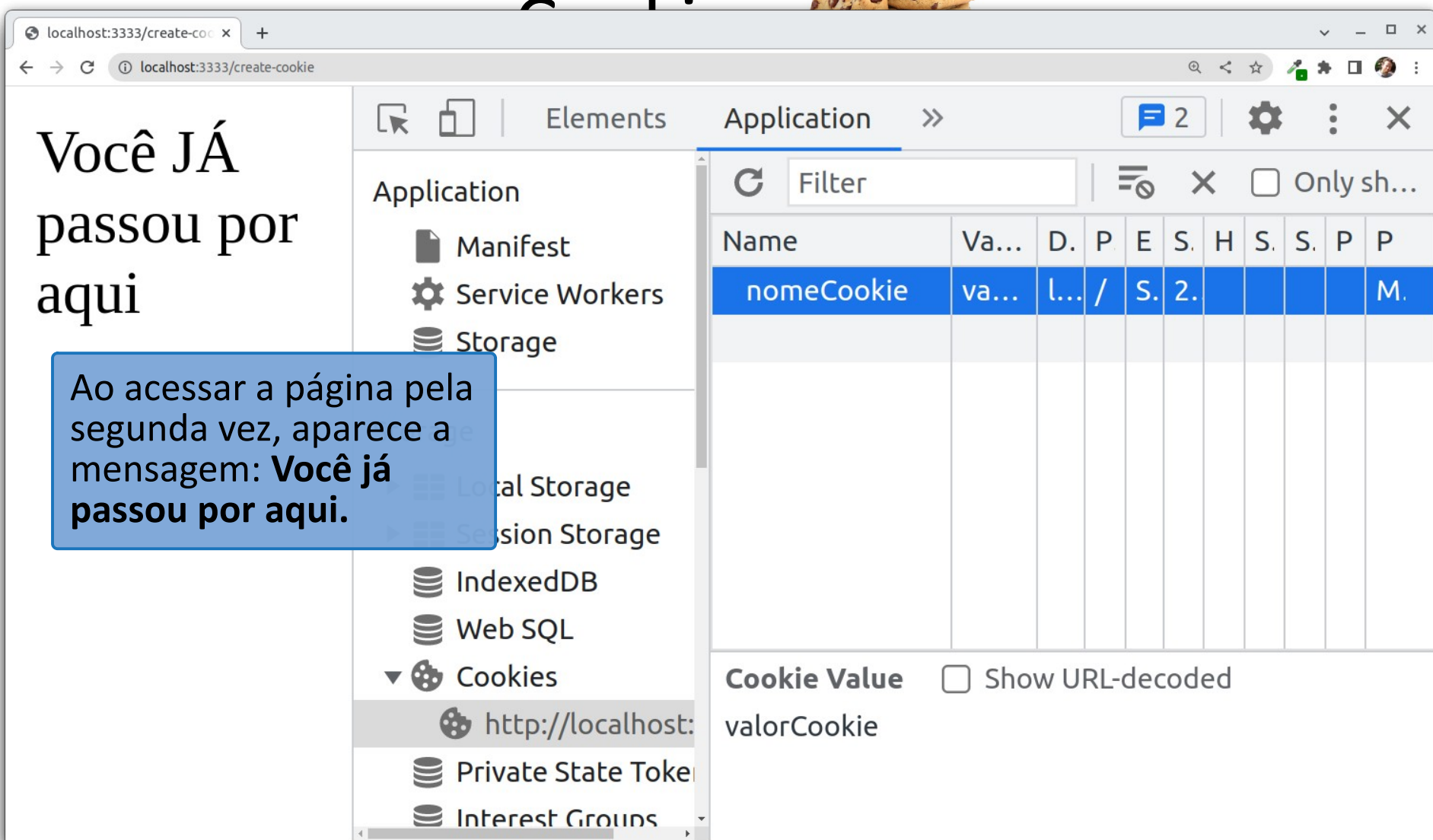
- A partir deste momento, podemos i) criar novos cookies e ii) identificar os cookies enviados pelo browser para o servidor
  - O array **req.cookies** armazena os cookies enviados pelo browser
  - O método **res.cookie** é usado para enviar um pedido de criação de um novo cookie no browser

```
// Arquivo src/router/router.ts
router.get('/create-cookie', mainController.createCookie);
```

```
// Arquivo src/controllers/main.ts
const createCookie = function (req: Request, res: Response) {
  if (!('nomeCookie' in req.cookies)) {
    res.cookie('nomeCookie', 'valorCookie');
    res.send('Você NUNCA passou por aqui!');
  } else {
    res.send('Você JÁ passou por aqui!');
  }
}
});
```







# Cookies



- Também podemos criar cookies com data de expiração

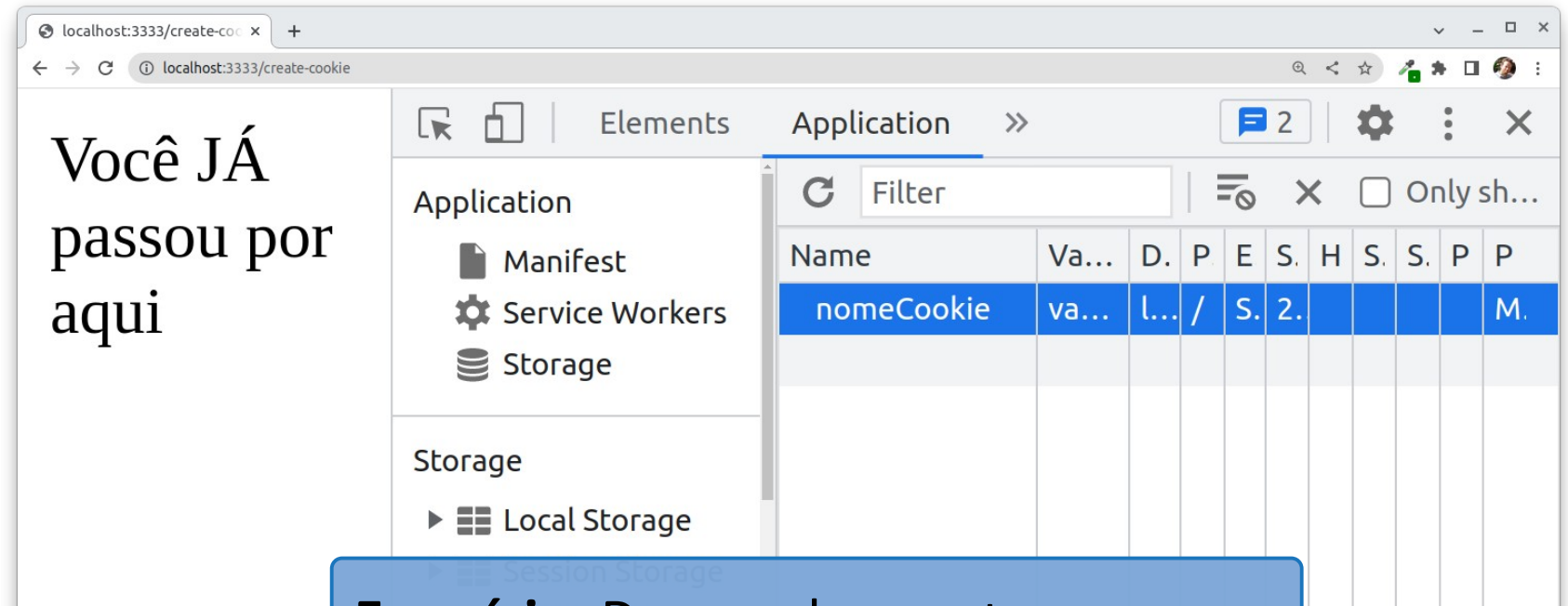
```
// Expira 360000 ms (6 minutos) após ser criado  
res.cookie(name, 'value', { maxAge: 360000 } );
```

- Usamos a função **clearCookie** para apagar um cookie já criado

```
// Arquivo src/router/router.ts  
router.get('/clear-cookie', mainController.clearCookie);
```

```
// Arquivo src/controllers/main.ts  
const clearCookie = function (req: Request, res: Response) {  
  res.clearCookie('nomeCookie');  
  res.send('cookie apagado');  
});
```

# Exercício 3



**Exercício:** Desenvolva a rota **create-cookie**, usando os passos apresentados nos slides e durante as aulas.

github  
**ExpTS**

express JS

# Sessões

- Através de sessões, podemos armazenar informações de estado (variáveis) no lado servidor
- Em vez do browser guardar um cookie por dado, ele guarda apenas um cookie contendo um **id de sessão (connect.sid)**

## Requisições do Browser

1 Get index.html HTTP1.1  
Host: server.com

3 Get index.html HTTP1.1  
Host: server.com  
**Cookie: connect.sid=6Bh**

## Requisições de server.com

2 HTTP1.1 200 OK  
**Set-Cookie: connect.sid=6Bh**  
... Conteúdo da página...

4 HTTP1.1 200 OK  
... Conteúdo da página...

## Sessão 6Bh Dados da sessão no servidor

user: Bruna  
itens-carrinho: 3  
start: 15:30



# Sessões

- Para usarmos as sessões, precisamos instalar um módulo para geração de **valores únicos para os IDs** das sessões
- Uma opção é o módulo **uuid** – Universally Unique Identifier – que é uma implementação do UUID descrito na [RFC 4122](https://tools.ietf.org/html/rfc4122)

```
$ npm install uuid
```

- Os UUIDs são valores de 128 bits que podem ser usados como ID únicos de qualquer coisa em sistemas computacionais
  - Ex: f0221c72-ac30-4796-83f5-fd7a8a4f6b15
- Embora a probabilidade de um UUID ser duplicado não seja nula, ela é próximo o suficiente de zero e pode ser ignorada



# Sessões

- Para usarmos as sessões, precisamos instalar um módulo para geração de **valores únicos para os IDs** das sessões
- Uma opção é o módulo **uuid** – Universally Unique Identifier – que é uma implementação do UUID descrito na [RFC 4122](https://tools.ietf.org/html/rfc4122)

```
$ npm install uuid
```

- Os UUIDs são valores de 128 bits que podem ser usados como  
Alguns desenvolvedores preferem usar o uuid como **chave primária** de tabelas, ao invés de um ID auto incrementado.
- Embora a probabilidade de um UUID ser duplicado não seja nula, ela é próximo o suficiente de zero e pode ser ignorada

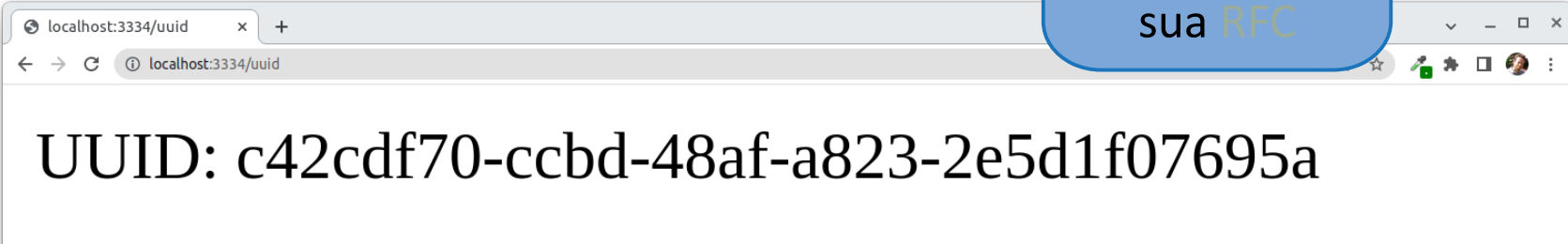
# Sessões

- O código abaixo mostra como os UUIDs podem ser gerados a partir do módulo **uuid**

```
// Arquivo src/router/router.ts  
router.get('/uuid', mainController.uuid);
```

```
// Arquivo src/controllers/main.ts  
const uuid = (req, res, next) => {  
  const uniqueId = uuidv4();  
  res.send(`UUID: ${uniqueId}`);  
};
```

Existem 5  
versões de  
UUID, conforme  
descrito em  
sua **RFC**



UUID: c42cdf70-ccbd-48af-a823-2e5d1f07695a

# Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session  
$ npm install -D @types/express-session
```

- Para usar o middleware, precisamos importá-lo e adicioná-lo em nossa aplicação com o método **use**

```
// Arquivo src/index.ts  
import session from 'express-session';  
...  
app.use(session({  
  genid: () => uuidv4(), // usamos UUID para gerar os SESSID  
  secret: 'Hi9Cf#mK98',  
  resave: true,  
  saveUninitialized: true,  
}));
```

# Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session
$ npm install -D @types/express
```

- Para usar o middleware, precisamos adicioná-lo em nossa aplicação com o seguinte código:

```
// Arquivo src/index.ts
import session from 'express-session'
...
app.use(session({
  genid: () => uuidv4(), // usamos UUID para gerar os SESSID
  secret: 'Hi9Cf#mK98',
  resave: true,
  saveUninitialized: true,
}));
```

Usado para adicionar uma assinatura (similar ao checksum) ao **session.id** enviado para o usuário. Quando o usuário devolve o **session.id**, a assinatura é usada para checar se o **session.id** é válido. Usa uma técnica chamada HMAC.

# Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session
$ npm install -D express-session
```

- Para usar o middleware em nossa aplicação

```
// Arquivo src/index.js
import session from 'express-session'
...
app.use(session({
  genid: () => '...',
  secret: 'Hi9Cf#mK98',
  resave: true,
  saveUninitialized: true,
}));
```

Quando **true**, a sessão do usuário é salva a cada requisição, mesmo que os dados da sessão não tenham sido modificados durante a requisição. Isso mantém a sessão ativa, visto que ela pode ser deletada após algum tempo de desuso.

Quando **session.id** é adicionado ao objeto de configuração, ele adiciona um **session.id** ao usuário. Isso evolui o **session.id** a cada requisição. A assinatura é baseada no **session.id** e no **secret**. Usa uma função **genid** para gerar os **SESSID**.

# Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session
$ npm install -D express-session
```

- Para usar o middleware **express-session** em nossa aplicação, precisamos adicioná-lo

```
// Arquivo src/index.js
import session from 'express-session'
...
app.use(session({
  genid: () => {
    secret: 'H!cf#mK98',
    resave: true,
    saveUninitialized: true,
  }
}));
```

Quando **true**, a sessão do usuário é salva a cada requisição.

Quando **true**, força que as sessões não inicializadas sejam salvas no store. Uma sessão não inicializada ocorre quando a sessão é nova e ainda não foi modificada.

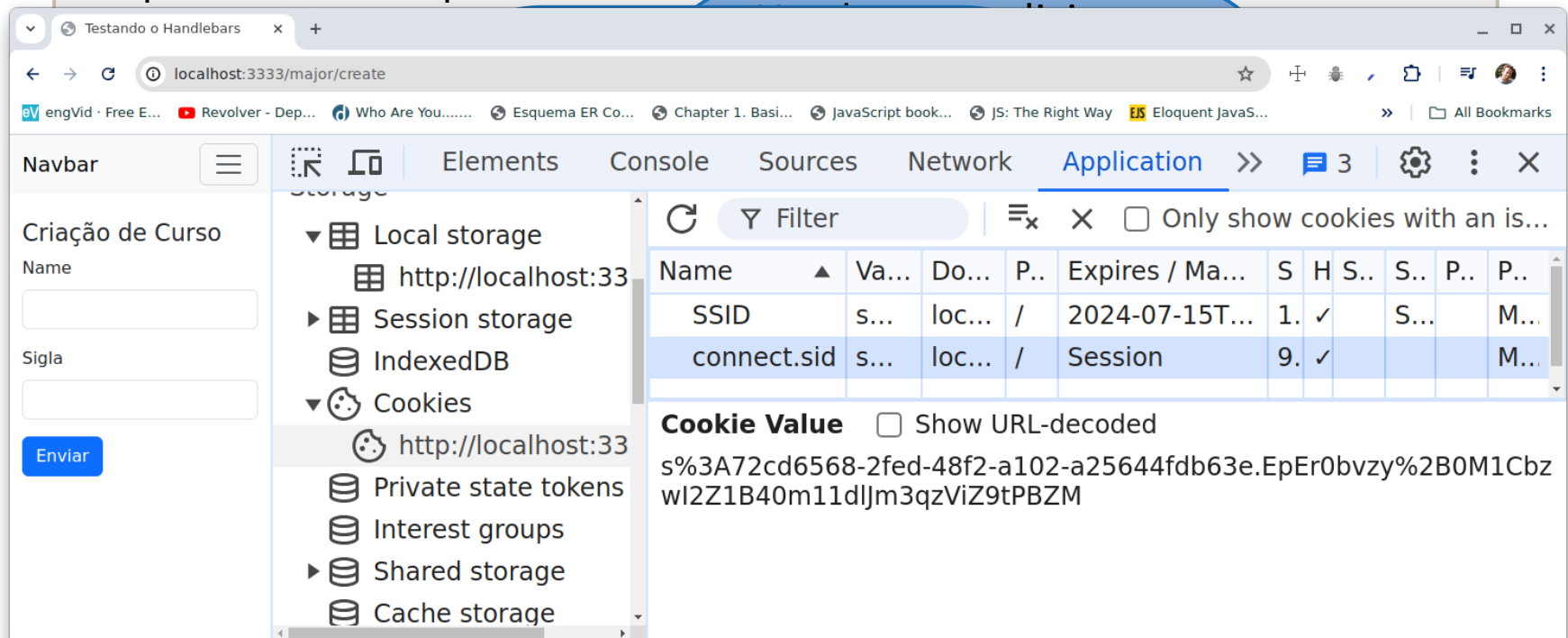
Adicionar **session.id** ao objeto de configuração. Isso gera o **session.id** para cada usuário. Envolve o **session.id** e gera uma MAC.

gerar os SESSID

# Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session
```



The screenshot shows a web browser with the URL `localhost:3333/major/create`. The Application tab is open, displaying a table of cookies. The table has columns: Name, Value, Domain, Path, Expires / Max-Age, Status, HttpOnly, Secure, SameSite, and Partitioned. Two cookies are listed: 'SSID' and 'connect.sid'.

Name	Value	Domain	Path	Expires / Max-Age	Status	HttpOnly	Secure	SameSite	Partitioned
SSID	s...	loc...	/	2024-07-15T...	1.	✓		S...	M...
connect.sid	s...	loc...	/	Session	9.	✓			M...

Below the table, the 'Cookie Value' section shows the value for the selected cookie: `s%3A72cd6568-2fed-48f2-a102-a25644fdb63e.EpEr0bvzy%2B0M1CbzwI2Z1B40m11dIjm3qzViZ9tPBZM`. The 'Show URL-decoded' checkbox is unchecked.

```
}});
```

# Cadastro de Usuários

- O cadastro de usuário requer a criação do modelo User, que por sua vez se relaciona com o modelo Major

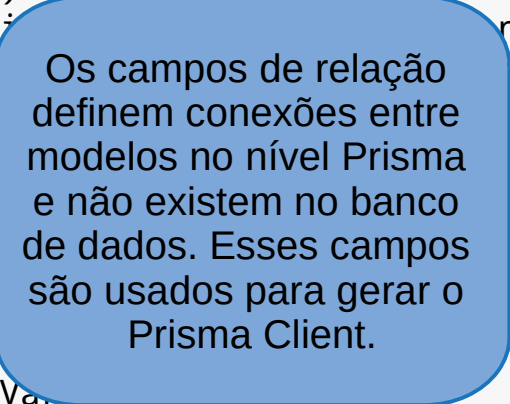
```
model User {  
  id          String      @id @default(uuid()) @db.Char(40)  
  name        String      @db.VarChar(100)  
  email       String      @unique @db.VarChar(100)  
  password    String      @db.Char(60)  
  major       Major       @relation(fields: [majorId], references: [id])  
  majorId     String      @db.Char(40)  
  createdAt   DateTime    @default(now()) @map("created_at")  
  updatedAt   DateTime    @updatedAt @map("updated_at")  
  @@map("usuarios")  
}  
  
model Major {  
  id          String      @id @default(uuid()) @db.Char(40)  
  name        String      @unique @db.VarChar(100)  
  code        String      @unique @db.Char(40)  
  description  String?    @db.Text()  
  createdAt   DateTime    @default(now()) @map("created_at")  
  updatedAt   DateTime    @updatedAt @map("updated_at")  
  User        User[]  
  @@map("majors")  
}
```



# Cadastro de Usuários

- O cadastro de usuário requer a criação do modelo User, que por sua vez se relaciona com o modelo Major

```
model User {  
  id          String      @id @default(uuid()) @db.Char(40)  
  name        String      @db.VarChar(100)  
  email       String      @unique @db.VarChar(100)  
  password    String      @db.Char(60)  
  major       Major       @relation(fields: [majorId], references: [id])  
  majorId     String      @db.Char(40)  
  createdAt   DateTime     @db.Timestamp(6)  
  updatedAt   DateTime     @updatedAt  
  @@map("usuarios")  
}  
  
model Major {  
  id          String      @id @default(uuid()) @db.Char(40)  
  name        String      @db.VarChar(100)  
  code        String      @unique @db.Char(40)  
  description  String?     @db.Text()  
  createdAt   DateTime     @default(now()) @map("created_at")  
  updatedAt   DateTime     @updatedAt @map("updated_at")  
  User        User[]  
  @@map("majors")  
}
```



Os campos de relação definem conexões entre modelos no nível Prisma e não existem no banco de dados. Esses campos são usados para gerar o Prisma Client.

```

david@coyote ~/dev/expApi/backend [main]× $ npx prisma migrate dev
--name add-usuarios-tables
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
Datasource "db": MySQL database "loja" at "localhost:3306"

model Usuario {
  id          String      @id @default(uuid()) @db.Char(40)
  nome        String      @db.VarChar(100)
  email       String      @unique @db.VarChar(100)
  senha       String      @db.VarChar(100)
  tipoUsuario TipoUsuario @relation(fields=[id], references=[id])
  tipoUsuarioId String    @db.Char(40)
  createdAt   DateTime    @default(now())
  updatedAt   DateTime    @updatedAt
}

migrations/
└─ 20231028115757 add_usuarios_tables/
   └─ migration.sql

Applying migration `20231028115757 add_usuarios_tables`

The following migration(s) have been created and applied from new
schema changes:
└─ 20231028115757 add_usuarios_tables/
   └─ migration.sql

Your database is now in sync with your schema.

model TipoUsuario {
  id          String      @id @default(uuid()) @db.Char(40)
  nome        String      @db.VarChar(100)
  usuarios    Usuario[]
}

@map("tipos_usuarios")

✓ Generated Prisma Client (v5.4.2) to ./node_modules/@prisma/client in 129ms

david@coyote ~/dev/expApi/backend [main]× $

```

# Criptografando as senhas

- Após a submissão do formulário, é necessário criptografar a senha antes de salvá-la no banco de dados
- Mas por quê? Por que não guardar a senha crua?
  - Todas as pessoas com acesso ao banco poderiam ver a senha
  - Os usuários frequentemente **usam a mesma senha** em vários sites
  - A senha iria aparecer nos **backups** do banco
  - Se o banco estiver na **cloud**, as senhas ficariam expostas na web
  - As senhas ficariam expostas a **ataques de SQL-injection**

# Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela **Estado**

estado	capital
Rio de Janeiro	Rio de Janeiro
Amazonas	Manaus
Minas Gerais	Belo Horizonte
Ceará	Fortaleza

Tabela **Usuario**

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

Busca da capital pelo estado

# Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela **Estado**

estado	capital
Rio de Janeiro	Rio de Janeiro
Amazonas	Manaus
Minas Gerais	Belo Horizonte
Ceará	Fortaleza

Tabela **Usuario**

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

Busca da capital pelo estado

```
SELECT estado, capital FROM estado  
WHERE estado = 'Amazonas'
```

estado	capital
Amazonas	Manaus

# Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela **Estado**

estado	capital
Rio de Janeiro	Rio de Janeiro
Amazonas	Manaus
Minas Gerais	Belo Horizonte
Ceará	Fortaleza

Tabela **Usuario**

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

Busca da capital pelo estado

' UNION SELECT login, senha FROM Usuario WHERE login != '

Submit



**SELECT** estado, capital **FROM** Estado  
**WHERE** estado = ' UNION SELECT login,  
senha **FROM** Usuario **WHERE** login != '

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

# Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela **Estado**

estado	capital
--------	---------

Tabela **Usuario**

login	senha
-------	-------

Os ataques de SQL-Injection só são possíveis se o invasor conseguir incluir caracteres como ' (aspas simples), " (aspas duplas) ou \b (caracter de backspace) nos inputs dos formulários.

Busca da capital pelo estado

' UNION SELECT login, senha FROM Usuario WHERE login != '

Submit

**SELECT** estado, capital **FROM** Estado  
**WHERE** estado = ' UNION SELECT login,  
senha **FROM** Usuario **WHERE** login != '

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

# Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela **Estado**

estado	capital
--------	---------

Tabela **Usuario**

login	senha
-------	-------

O Sequelize resolve este problema adicionando um caracter de **escape** a tais elementos. Por exemplo, o caracter ' vira \', " vira \" e \b vira \\b. Vide [função de escape de Sequelize](#).

Mesmo assim, é sempre saudável fazer validação dos inputs.

Busca da capital pelo estado

' UNION SELECT login, senha FROM Usuario WHERE login != '

Submit

**SELECT** estado, capital **FROM** Estado  
**WHERE** estado = ' UNION SELECT login,  
senha **FROM** Usuario **WHERE** login != '

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon



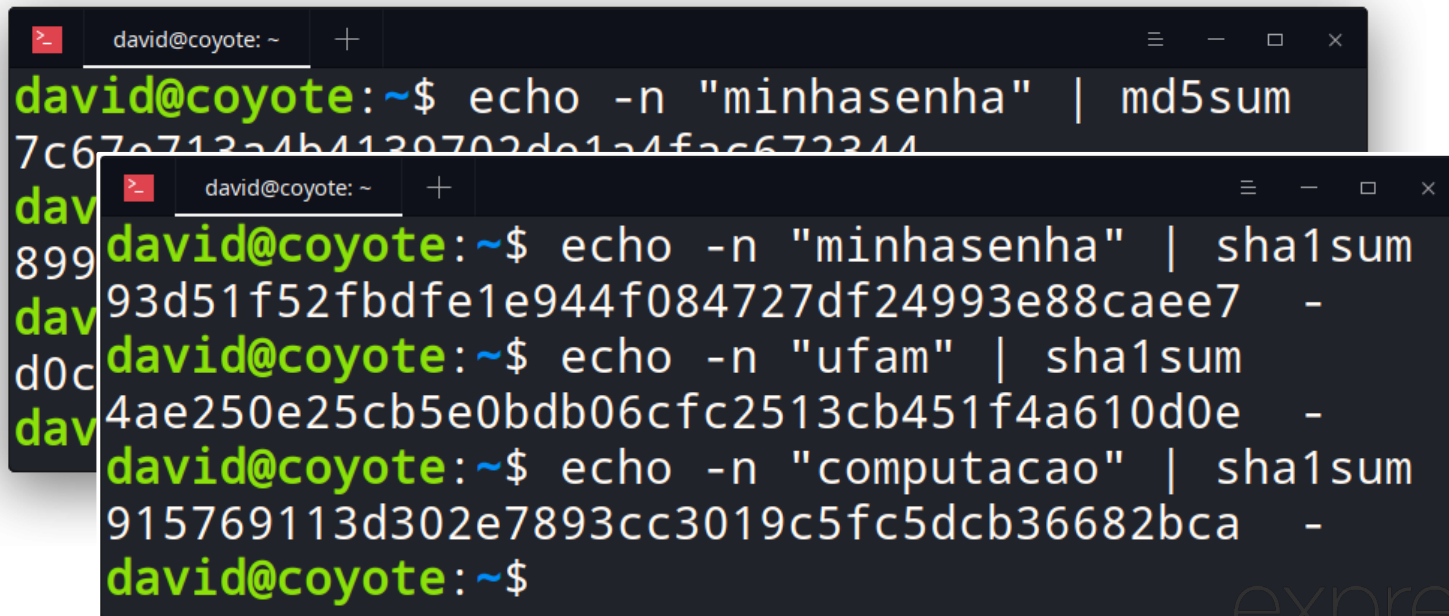
# Funções HASH

- Uma função Hash é um algoritmo que transforma um bloco de dados qualquer em série de caracteres de comprimento fixo
  - Ex: md5 (32 caracteres), sha1 (40 caracteres), sha256 (64 caracteres), etc
- São funções de **mão-única**, isto é, não é possível recuperar o bloco de dados original a partir do hash gerado

```
david@coyote: ~  
david@coyote:~$ echo -n "minhasenha" | md5sum  
7c67e713a4b4139702de1a4fac672344 -  
david@coyote:~$ echo -n "ufam" | md5sum  
8996fe161805927c27a92fae2ca238d2 -  
david@coyote:~$ echo -n "computacao" | md5sum  
d0cc5ed8aecb1898032c48af57057b9f -  
david@coyote:~$
```

# Funções HASH

- Uma função Hash é um algoritmo que transforma um bloco de dados qualquer em série de caracteres de comprimento fixo
  - Ex: md5 (32 caracteres), sha1 (40 caracteres), sha256 (64 caracteres), etc
- São funções de **mão-única**, isto é, não é possível recuperar o bloco de dados original a partir do hash gerado



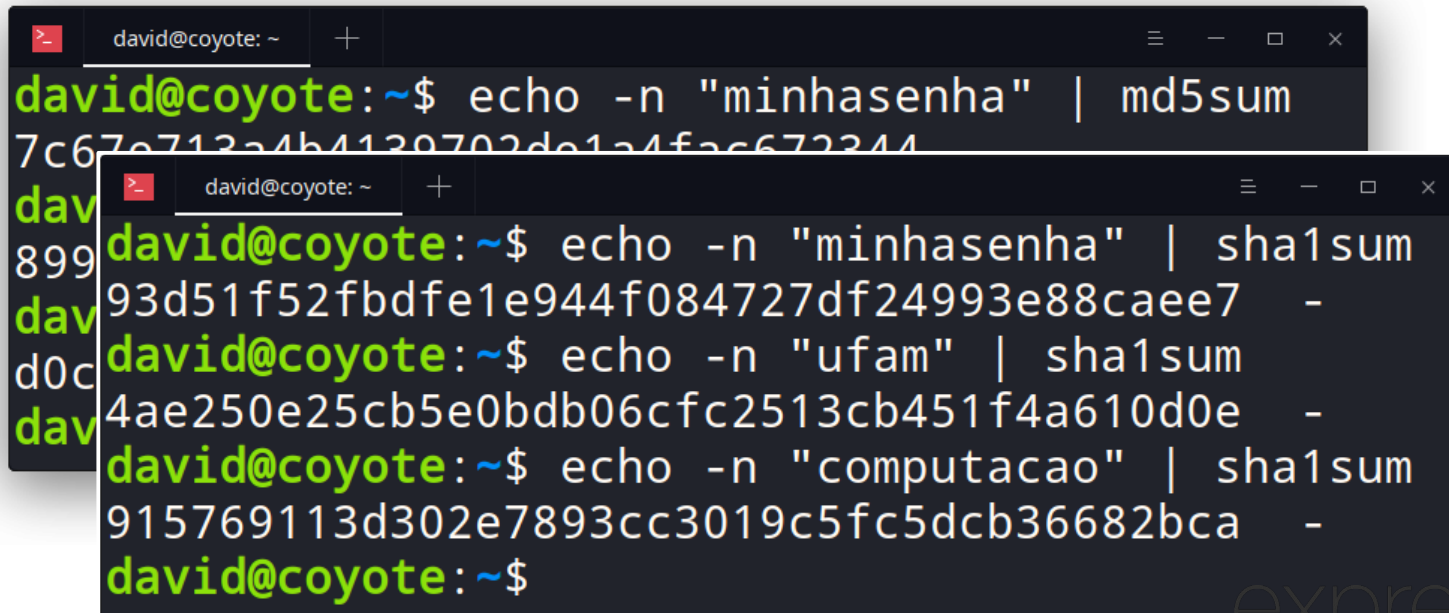
```
david@coyote: ~$ echo -n "minhasenha" | md5sum
7c67e712a4b4130702de1a4fac672344
david@coyote: ~$ echo -n "minhasenha" | sha1sum
93d51f52fbdfef1e944f084727df24993e88caee7 -
david@coyote: ~$ echo -n "ufam" | sha1sum
4ae250e25cb5e0bdb06cfc2513cb451f4a610d0e -
david@coyote: ~$ echo -n "computacao" | sha1sum
915769113d302e7893cc3019c5fc5dcb36682bca -
david@coyote: ~$
```

# Funções HASH

- Uma função Hash é um algoritmo que transforma um bloco de dados qualquer em série de caracteres de comprimento fixo

Os comandos **md5sum** e **sha1sum** são instalados por padrão em sistemas UNIX, GNU/Linux e BSD. Eles são usados para verificar a integridade de dados transmitidos através da Web. Para maiores informações, vide [Soma de Verificação \(Checksum\)](#)

bloco de dados original a partir do hash gerado

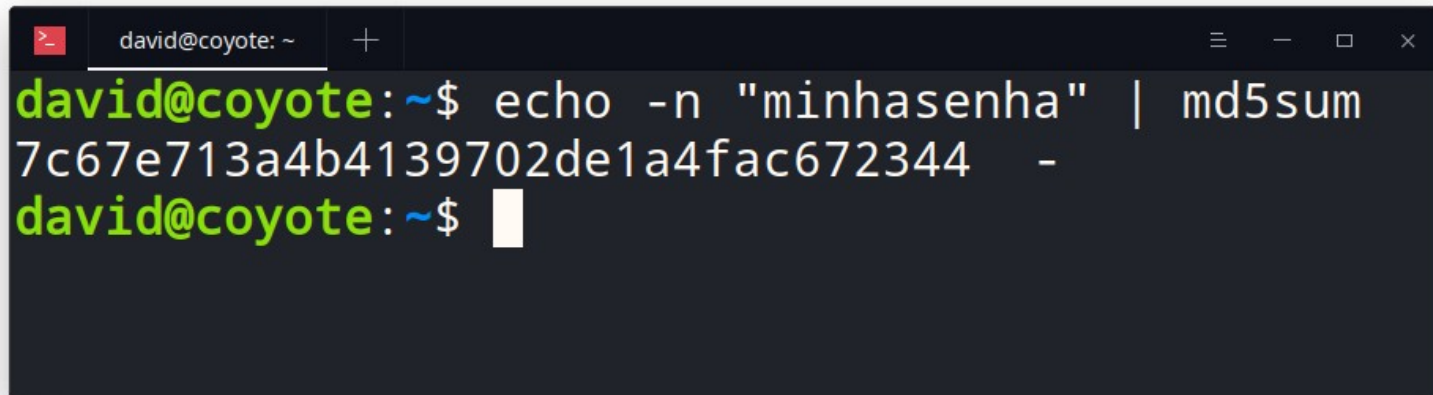


```
david@coyote: ~$ echo -n "minhasenha" | md5sum
7c67e712a4b4130702de1a4fac672344

david@coyote: ~$ echo -n "minhasenha" | sha1sum
93d51f52fbdfef1e944f084727df24993e88caee7 -
david@coyote: ~$ echo -n "ufam" | sha1sum
4ae250e25cb5e0bdb06cfc2513cb451f4a610d0e -
david@coyote: ~$ echo -n "computacao" | sha1sum
915769113d302e7893cc3019c5fc5dcb36682bca -
david@coyote: ~$
```

# Funções HASH

- Considere um sistema que usa a função hash MD5 para criptografar as senhas de seus usuários
  - Ex: se a senha de um usuário é **minhasenha**, a string armazenada no banco será **7c67e713a4b4139702de1a4fac672344**
  - Quando o usuário tentar logar no sistema, o servidor irá comparar o md5 da senha informada com a string armazenada no banco

A terminal window with a dark background. The title bar shows 'david@coyote: ~' and standard window controls. The prompt is 'david@coyote:~\$'. The command 'echo -n "minhasenha" | md5sum' is entered. The output '7c67e713a4b4139702de1a4fac672344 -' is displayed. The prompt 'david@coyote:~\$' is shown again with a cursor.

```
david@coyote:~$ echo -n "minhasenha" | md5sum
7c67e713a4b4139702de1a4fac672344 -
david@coyote:~$
```

# Funções HASH

- Considere um sistema que usa a função hash MD5 para criptografar as senhas de seus usuários
    - Ex: se a senha de um usuário é **minhasenha**, a string armazenada no banco será **7c67e713a4b4139702de1a4fac672344**
    - Quando o usuário tentar logar no sistema, o servidor irá comparar o md5 da senha informada com a string armazenada no banco
- Note que **ninguém** poderá descobrir a senha do usuário caso acesso a tabela Usuário do banco de dados

```
david@coyote:~$ echo -n "minhasenha" | md5sum
7c67e713a4b4139702de1a4fac672344  -
david@coyote:~$
```

# Funções HASH

- Considere um sistema que usa a função hash MD5 para criptografar as senhas de seus usuários
    - Ex: se a senha de um usuário é **minhasenha**, a string armazenada no banco será **7c67e713a4b4139702de1a4fac672344**
    - Quando o usuário tentar logar no sistema, o servidor irá comparar o md5 da senha informada com a string armazenada no banco
- Note que **ninguém** poderá descobrir a senha do usuário caso a
- Será mesmo? Então vamos no Google e inserir a string hash **7c67e713a4b4139702de1a4fac672344** no campo de busca

```
7c67e713a4b4139702de1a4fac672344 -  
david@coyote:~$
```

# Funções HASH

- Considerar a criptografia

- Ex: se a senha não bater

- Quando o md5

Note que

a senha

7c67e

7c67e  
david

A screenshot of a Google search interface. The search bar at the top contains the MD5 hash `7c67e713a4b4139702de1a4fac672344`. Below the search bar, the navigation menu includes 'Todas', 'Maps', 'Vídeos', 'Imagens', 'Shopping', 'Mais', and 'Configurações'. The search results show 'Aproximadamente 43 resultados (0,27 segundos)'. The first result is titled '7c67e713a4b4139702de1a4fac672344 - Hash Toolkit' with a URL `https://hashtoolkit.com › reverse-md5-hash › 7c67e71...` and a snippet: 'Decrypt md5 Hash Results for: 7c67e713a4b4139702de1a4fac672344 ... md5, 7c67e713a4b4139702de1a4fac672344, minhasenha ...'. The second result is titled 'Hash Md5: 7c67e713a4b4139702de1a4fac672344 - MD5Hashing.net' with a URL `https://md5hashing.net › hash › 7c67e713a4b4139702de1a4fac672344` and a snippet: 'Decoded hash Md5: 7c67e713a4b4139702de1a4fac672344: minhasenha.'. The third result is titled 'Usando o md5 - Recursos do PHP' with a URL `recursosdophp.blogspot.com › normal-0-21-false-false-false-pt-br-x` and a snippet: 'md5("minhasenha") = "7c67e713a4b4139702de1a4fac672344". Com isso se você ... valor de "\$senha" ele vai dar senha Invalida. Bom, isso até mais .'. The Google logo is visible in the bottom right corner.

Cor  
crip

Ex

no

Q

N  
a

Best MD5 & SHA1 Passwords x

hashtoolkit.com/decrypt-hash/?hash=7c67e713a4b4139702de1a4fac672344

### Hash Toolkit

Search in 27,338,908,645 decrypted hashes

Hash: 7c67e713a4b4139702de1a4fac672344

Decrypt Hash Results for: 7c67e713a4b4139702de1a4fac672344

Algorithm	Hash	Decrypted
md5	7c67e713a4b4139702de1a4fac672344	minhasenha

Hashes for: minhasenha

Algorithm	Hash	Decrypted
sha1	93d51f52fbdfef1e944f084727df24993e88caee7	minhasenha
sha256	79809644a830ef92424a66227252b87bbdfb633a9dab18ba450c1b8d35665f20	minhasenha
sha384	f703d9fdd6044fc595652fb161ba90db9293b7b83c11b1c5e74bc76705d7917244136ee4ebef03b49b6b3168e0e41f	minhasenha

Configurações

la

rar

gina

Google



- Cor
- crip
- Ex
- no

Sistemas como o **Hash Toolkit** usam tabelas contendo bilhões de valores de hash pré-calculados. Essas tabelas normalmente são chamadas de **rainbow tables**.

Best MD5 & SHA1 Passwords

hashtoolkit.com/decrypt-hash/?hash=7c67e713a4b4139702de1a4fac672344

### Hash Toolkit

Search in 27,338,908,645 decrypted hashes

Hash: 7c67e713a4b4139702de1a4fac672344

md5 7c67e713a4b4139702de1a4fac672344 minhasenha

Hashes for: minhasenha

Algorithm	Hash	Decrypted
sha1	93d51f52fbdfef1e944f084727df24993e88caee7	minhasenha
sha256	79809644a830ef92424a66227252b87bbdfb633a9dab18ba450c1b8d35665f20	minhasenha
sha384	f703d9fdd6044fc595652fb161ba90db9293b7b83c11b1c5e74bc76705d7917244136ee4eb03b49b6b3168e0e41f	minhasenha



# Salt



- O **salt** é uma estratégia utilizada para evitar que duas senhas idênticas produzam **hashes** idênticos
- A ideia é concatenar a senha original do usuário com uma sequência adicional de caracteres aleatórios, chamada salt
- Para exemplificar, podemos aplicar o salt **Us8#upK12MjsM** à senha original do usuário, **minhasenha**

```

david@coyote: ~$ echo -n "Us8#upK12MjsMminhasenha" | md5sum
d3eeb71a83744a4bbaa3ce41be87a292 -
david@coyote: ~$
```

# Salt



- O **salt** é uma estratégia utilizada para evitar que duas senhas idênticas produzam **hashes** idênticos
- A ideia é concatenar a senha original do usuário com uma

d3eeb71a83744a4bbaa3ce41be87a292

Todas Maps Vídeos Imagens Shopping Mais Configurações

Sua pesquisa - **d3eeb71a83744a4bbaa3ce41be87a292** - não encontrou nenhum documento correspondente.

Sugestões:

- Certifique-se de que todas as palavras estejam escritas corretamente.
- Tente palavras-chave diferentes.
- Tente palavras-chave mais genéricas.

Google

# Salt



- O **salt** é uma estratégia utilizada para evitar que duas senhas idênticas produzam **hashes** idênticos
- A ideia é concatenar a senha original do usuário com uma

De uma forma geral, o salt é gerado aleatoriamente para cada usuário e armazenado no banco junto com o hash MD5 resultante:

**Us8#upK12MjsMd3eeb71a83744a4bbaa3ce41be87a292**

Sua pesquisa - **d3eeb71a83744a4bbaa3ce41be87a292** - não encontrou nenhum documento correspondente.

Sugestões:

- Certifique-se de que todas as palavras estejam escritas corretamente.
- Tente palavras-chave diferentes.
- Tente palavras-chave mais genéricas.

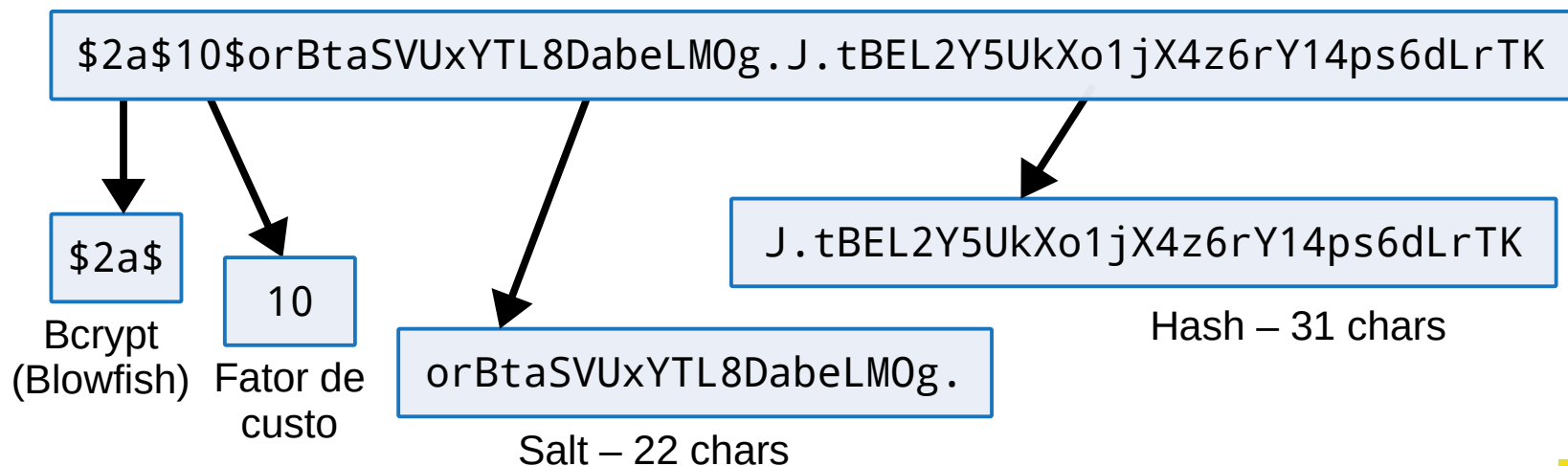


# O módulo bcrypt

- O módulo **bcrypt** é uma boa opção para geração de senhas, pois incorpora uma função hash à uma estratégia de salt

```
$ npm install bcryptjs  
$ npm install -D @types/bcryptjs
```

- O bcrypt é um **algoritmo de hash** usado para geração de senhas em sistemas como OpenBSD e algumas distribuições linux



# O módulo bcrypt

- Para criar um usuário com senha criptografada, podemos usar o código abaixo

```
// Arquivo src/services/user.ts
import { PrismaClient, User } from '@prisma/client'
import bcrypt from 'bcryptjs';
import { UserCreateDto } from '../types/user'

export const createUser = async (user: UserCreateDto):
  Promise<User> =>
{
  const salt = await bcrypt.genSalt(10)
  const password = await bcrypt.hash(user.password, salt)
  return await prisma.user.create({
    data: { ...user, password }
  })
}
```

# O módulo bcrypt

- Para criar um usuário com senha criptografada, podemos usar o código abaixo

```
// Arquivo src/services/user.ts
import { PrismaClient, User } from '@prisma/client'
import bcrypt from 'bcryptjs';
import { UserCreateDto } from '../types/user'

export const createUser = async (user: UserCreateDto): Promise<User> =>
{
  const salt = await bcrypt.genSalt(10)
  const password = await bcrypt.hash(user.password, salt)
  return await prisma.user.create({
    data: { ...user, password }
  })
}
```

Número de rounds para geração do hash

Senha informada pelo usuário

# O módulo bcrypt

- Para criar um usuário com senha criptografada, podemos usar o código abaixo

```
// Arquivo src/services/user.ts
import { PrismaClient, User } from '@prisma/client'
import bcrypt from 'bcryptjs';
import { UserCreateDto } from '../types/user'
```

Número de rounds para geração do

nome	email	senha
David Fernandes	david@icomp.ufam.edu.br	\$2a\$10\$LDsc/xMa91HiUY03KiQxVuZUJCbN0CNcA6F/k9vPH9H/NII/MTNqO

```
const salt = await bcrypt.genSalt(rounds)
const password = await bcrypt.hash(user.password, salt)
return await prisma.user.create({
  data: { ...user, password }
})
}
```

Senha informada pelo usuário



# Login de Usuários

- As variáveis de sessão são criadas imediatamente após o **login** do usuário e destruídas logo após o **logout**
- Desta forma, para saber se um usuário está logado, basta verificar se uma data variável de seção existe ou não
- Para adotar essa estratégia, podemos usar uma variável de sessão chamada **uid**, cujo valor será o **id** do usuário
- Mas para isso precisaremos expandir o tipo do objeto **req.session**:

```
// Arquivo src/index.ts  
declare module 'express-session' {  
  interface SessionData {  
    uid: string;  
  }  
}
```

# Login de Usuários

- Da mesma forma que o **signup**, podemos criar uma função em `service/auth.ts` para checar as credenciais do usuário
- Na action login, usamos o método **bcrypt.compare()** para verificar se a senha digitada pelo usuário está correta ou não

```
import { LoginDto } from '../types/user'
import bcrypt from 'bcryptjs'

export const checkAuth = async (credentials: LoginDto):
  Promise<boolean> =>
{
  const user = await prisma.user.findUnique({
    where: { email: credentials.email },
  })
  if (!user) return false
  return await bcrypt.compare(
    credentials.password, user.password
  )
}
```

# Logout de Usuários

- O controlador main também deverá conter uma action **logout**, que será usada para **encerrar a sessão** do usuário



- Para encerrar a sessão, destruindo todas as suas variáveis, usamos o método **req.session.destroy()**

```
const logout = (req, res) => {  
  req.session.destroy(function (err) {  
    if (err) res.send(err);  
    else res.redirect("/")  
  });  
}
```

# Exercício 4

```
model User {  
  id      String      @id @default(uuid()) @db.Char(40)  
  name    String      @db.VarChar(100)  
  email   String      @unique @db.VarChar(100)  
  password String      @db.Char(60)  
  major   Major        @relation(fields: [majorId], references: [id])  
  majorId String      @db.Char(40)  
  createdAt DateTime   @default(now()) @map("created_at")  
  updatedAt DateTime   @updatedAt @map("updated_at")  
  @@map("usuarios")  
}
```

**Exercício:** Desenvolva o controlador Auth, com as funções **signup**, **login** e **logout**, seguindo os passos apresentados nos slides e durante as aulas

github  
**ExpTS**

express **JS**