

# Desenvolvimento Web

Trilha JavaScript com Angular e Node

Instrutor: Júlio Pereira Machado ([julio.machado@pucrs.br](mailto:julio.machado@pucrs.br))



# NoSQL



# NoSQL

- NoSQL engloba diversas tecnologias de bases de dados alternativas ao modelo relacional
  - Documentos
  - Grafos
  - Chaves-valor
  - Colunas
  - etc

# NoSQL – Modelo Relacional

## Person

ID	FIRST_NAME	LAST_NAME
0	Steven	Edouard
1	Sam	Brightwood

## Account

ID	ACCOUNT_TYPE	ACCOUNT_BALANCE	CURRENCY	HOLDER (FK: Persons)
0	Investment	80000.00	USD	0
1	Savings	70400.00	USD	0
2	Checking	4500.00	USD	0
3	Checking	4500.00	YEN	1
4	Investment	4500.00	YEN	1
5	Savings	4500.00	YEN	1

# NoSQL – Modelo de Documentos

# Person Collection (Person is the root entity)

```
[
{
  "ID": 0,
  "first_name": "Steven",
  "last_name": "Edouard",
  "accounts": [
    {
      "id": 0,
      "account_type": "Investment",
      "account_balance": "80000.00",
      "currency": "USD"
    },
    {
      "id": 1,
      "account_type": "Savings",
      "account_balance": "70400.00",
      "currency": "USD"
    },
    {
      "id": 2,
      "account_type": "Checking",
      "account_balance": "80000.00",
      "currency": "USD"
    }
  ]
}
```

```
{
  "ID": 1,
  "first_name": "Sam",
  "last_name": "Brightwood",
  "accounts": [
    {
      "id": 3,
      "account_type": "Checking",
      "account_balance": "4500.00",
      "currency": "YEN"
    },
    {
      "id": 4,
      "account_type": "Investment",
      "account_balance": "4500.00",
      "currency": "YEN"
    },
    {
      "id": 5,
      "account_type": "Savings",
      "account_balance": "4500.00",
      "currency": "YEN"
    }
  ]
}
```

# NoSQL – Modelo de Documentos

## User Document

```
{  
  "ID": 0,  
  "username": "sedouard",  
  "created_at": "2015:02:14:01:00:00:00"  
}
```

## Post Document

```
{  
  "ID": 0,  
  "created_by": "ObjectID(ref User)",  
  "created_at": "2015:02:14:01:00:00:00",  
  "created_at": "2015:02:14:01:00:00:00",  
  "text": "this is a new cheep!"  
}
```

# MongoDB



# MongoDB

- Base de dados do tipo NoSQL
  - Suporta diferentes modelos de dados (chave-valor, documentos, etc)
- Código aberto
- Sintaxe semelhante a JSON
  - Formato interno é chamado de BSON (Binary JSON) <http://bsonspec.org/>

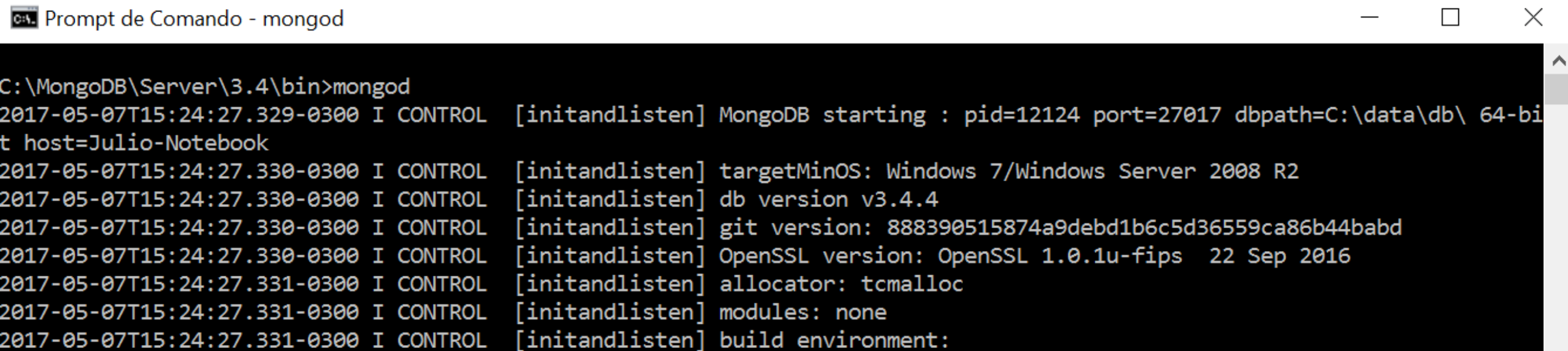
<https://www.mongodb.com/>





# MongoDB - Servidor

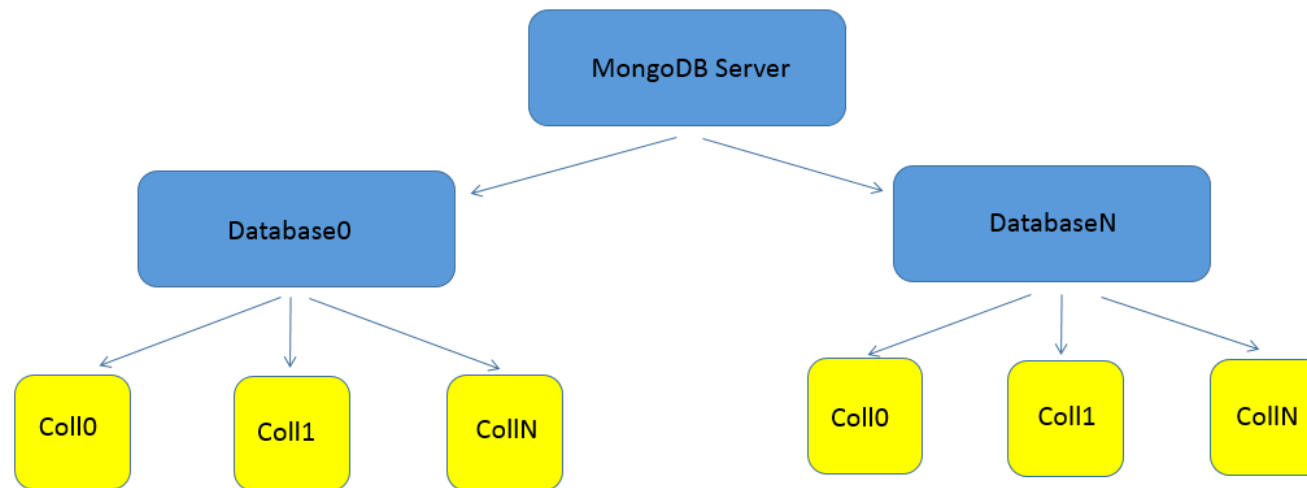
- Iniciar um servidor local na pasta de dados padrão
  - Executar comando mongod
  - Pasta padrão será o diretório “/data/db” (deve ser criado previamente)
- Iniciar um servidor local em uma nova base de dados “/data”
  - Executar comando mongod --dbpath=/data



```
C:\MongoDB\Server\3.4\bin>mongod
2017-05-07T15:24:27.329-0300 I CONTROL [initandlisten] MongoDB starting : pid=12124 port=27017 dbpath=C:\data\db\ 64-bit host=Julio-Notebook
2017-05-07T15:24:27.330-0300 I CONTROL [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2017-05-07T15:24:27.330-0300 I CONTROL [initandlisten] db version v3.4.4
2017-05-07T15:24:27.330-0300 I CONTROL [initandlisten] git version: 888390515874a9debd1b6c5d36559ca86b44babd
2017-05-07T15:24:27.330-0300 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.1u-fips 22 Sep 2016
2017-05-07T15:24:27.331-0300 I CONTROL [initandlisten] allocator: tcmalloc
2017-05-07T15:24:27.331-0300 I CONTROL [initandlisten] modules: none
2017-05-07T15:24:27.331-0300 I CONTROL [initandlisten] build environment:
```

# MongoDB – Estrutura de Dados

- Modelo mais utilizado é o de documentos
- Documentos que compartilham uma estrutura semelhante são organizados em coleções
- Uma base de dados é uma coleção de documentos



# MongoDB – Estrutura de Dados

- Documentos são compostos de pares campo-valor
  - Nomes de campos são strings
  - Campo `_id` é reservado e utilizado como identificador
    - Tipo `ObjectId` é o padrão quando um documento é inserido sem um id explícito
- O valor de um campo pode ser qualquer tipo BSON
  - Ver documentação <https://docs.mongodb.com/manual/reference/bson-types/>

```
{
  _id: ObjectId("5099803df3f4948bd2f98391"),
  name: { first: "Alan", last: "Turing" },
  birth: new Date('Jun 23, 1912'),
  death: new Date('Jun 07, 1954'),
  contribs: [ "Turing machine", "Turing test", "Turingery" ],
  views : NumberLong(1250000)
}
```

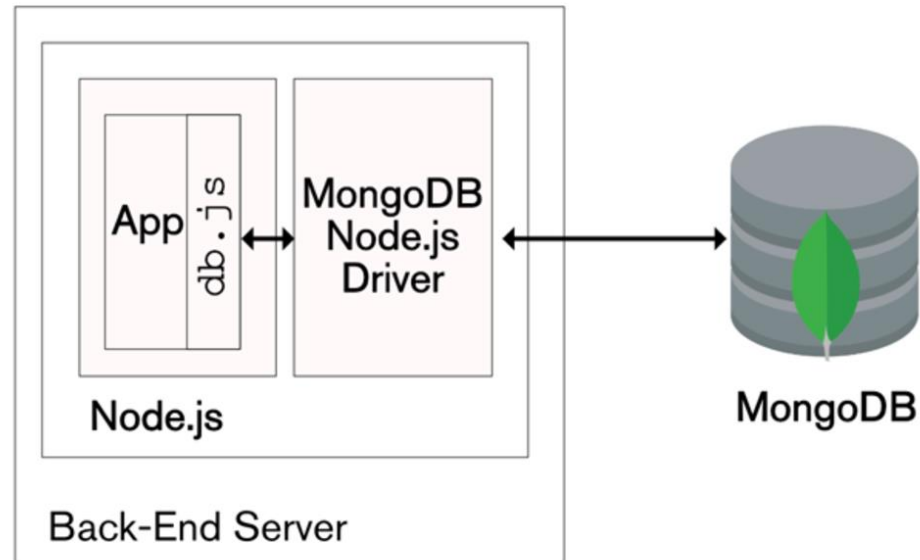
# MongoDB – Estrutura de Dados

- MongoDB possui a característica de esquema dinâmico
  - Um novo campo pode ser adicionado a um documento sem afetar os demais
- A estrutura dos documentos podem variar
  - Exemplo, nem todos os documentos que descrevem usuários precisam conter um campo userid ou um campo com a data do seu último login
- Ferramentas podem ser utilizadas para editar e visualizar os dados
  - MongoDB Compass – opção “oficial”
  - Studio3T - <https://studio3t.com/>

# MongoDB - Drivers

- MongoDB fornece drivers nativos para diferentes linguagens
  - Java, JavaScript, .NET, Python, etc
  - Instalar via comando `npm install mongodb --save`
  - Instalar tipos TypeScript via comando `npm install @types/mongodb --save-dev`
  - Operações suportam promises

<http://mongodb.github.io/node-mongodb-native/>



# MongoDB - Conexão

- A primeira etapa é configurar uma conexão com o servidor do MongoDB através do método **MongoClient.connect**
  - Retorna um objeto *MongoClient*
- O servidor a ser utilizado é feito via uma string de conexão
  - Ver documentação em <https://docs.mongodb.com/manual/reference/connection-string/>

# MongoDB - Conexão

- Conexão via promises com async await

```
import {MongoClient, Db} from 'mongodb';  
let url = 'mongodb://localhost:27017/teste';  
let cliente = await MongoClient.connect(url, {useNewUrlParser: true});  
let database = cliente.db('teste');
```

# MongoDB - Pool de Conexões

- Um *pool* de conexões é uma cache de conexões com a base de dados que podem ser reutilizadas a fim de melhorar a performance
- O *driver* de acesso ao MongoDB possui um *pool* de conexões
  - Tamanho padrão é de cinco conexões
- IMPORTANTE: deve-se estabelecer o acesso através de uma chamada única ao *MongoClient.connect* e reutilizar a variável *database* em todos os acessos realizados



# MongoDB - CRUD

- MongoDB driver fornece as operações de criação, leitura, alteração e remoção de documentos

```
db.users.insertOne(  ← collection
{
  name: "sue",        ← field: value
  age: 26,             ← field: value
  status: "pending"   ← field: value
}
)                    } document
```

# MongoDB - CRUD

- MongoDB driver fornece as operações de criação, leitura, alteração e remoção de documentos

```
db.users.find(  
  { age: { $gt: 18 } },  
  { name: 1, address: 1 }  
) .limit(5)
```

← collection  
← query criteria  
← projection  
← cursor modifier

# MongoDB - CRUD

- MongoDB driver fornece as operações de criação, leitura, alteração e remoção de documentos


```
db.users.updateMany(  
  { age: { $lt: 18 } },  
  { $set: { status: "reject" } }  
)
```

← collection  
← update filter  
← update action

# MongoDB - CRUD

- MongoDB driver fornece as operações de criação, leitura, alteração e remoção de documentos

```
db.users.deleteMany(  
  { status: "reject" }  
)
```



collection

delete filter

# MongoDB - Inserção

- Métodos do objeto **Collection** para inserir novos documentos em uma coleção:
  - `insertOne()` – insere um documento
  - `insertMany()` – insere vários documentos
- Se a coleção não existe, ela é criada automaticamente
- Se um campo `_id` não é informado, o driver automaticamente gera um campo `_id` do tipo `ObjectId`
- As operações de inserção são atômicas no nível de um único documento sobre uma coleção
- Resultado da inserção é um objeto `insertWriteOpResult` contendo as propriedades:
  - `result` – objeto com o resultado da operação
  - `insertedCount` – número de documentos inseridos
- As operações retornam *promises* se não fornecer um *callback*

# MongoDB - Inserção

```
let url = 'mongodb://localhost:27017/teste';  
let cliente = await MongoClient.connect(url, {useNewUrlParser: true});  
let database = cliente.db('teste');  
let resultado = await bd.collection('teste').insertOne({nome:'teste'});
```

# MongoDB - Inserção

MongoDB Compass - localhost:27017/teste.colecaoteste

Connect View Collection Help

localhost:27017  
Community version 3.4.4

3 DBS 2 COLLECTIONS

filter

admin >

local >

teste v

colecaoteste

teste.colecaoteste

DOCUMENTS	1	TOTAL SIZE	38B	AVG. SIZE	38B
INDEXES	1	TOTAL SIZE	16.0KB	AVG. SIZE	16.0KB

SCHEMA DOCUMENTS INDEXES EXPLAIN PLAN VALIDATION

{ "filter" : "example" }

APPLY

Query returned 1 document.

REFRESH

INSERT DOCUMENT

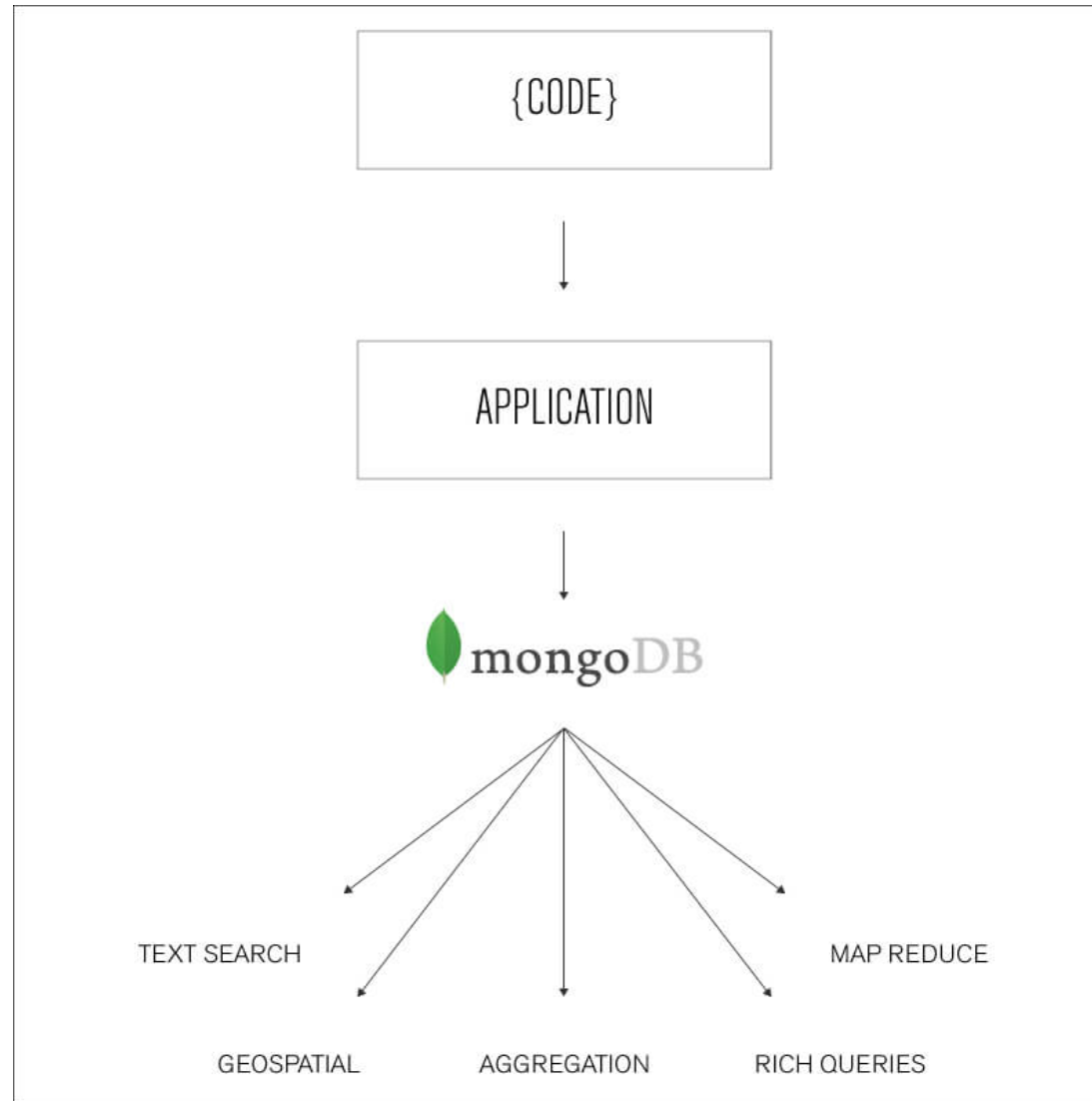
```
{
  "_id": ObjectId('590f85ae2a51e42098c46ad9'),
  "nome": "teste"
}
```

# MongoDB - Leitura

- Métodos do objeto **Collection** para realizar consultas:
  - find() – buscar o(s) documento(s) de acordo com o critério de busca
  - findOne() – buscar o primeiro documento de acordo com o critério de busca
- Exemplos de critérios de busca:
  - {} – todos os documentos
  - {nome:valor} – documentos que possuem o par "nome:valor"
  - { \$or: [{ nome1: valor1 }, { nome2: valor2 }]} – documentos que possuem o par "nome1:valor" ou "nome2:valor2"
  - Ver documentação <https://docs.mongodb.com/manual/reference/operator/query/#query-selectors>
- As operações retornam um objeto Cursor
  - toArray() - constrói uma representação de array da coleção de documentos obtida
  - hasNext() – verifica se existe um novo elemento a ser retornado
  - next() – itera sobre os documentos retornados pelo cursor



# MongoDB - Leitura



# MongoDB - Leitura

```
let url = 'mongodb://localhost:27017/teste';  
let cliente = await MongoClient.connect(url, {useNewUrlParser: true});  
let database = cliente.db('teste');  
let resultado = await bd.collection('teste').find({}).toArray();
```

# MongoDB - Leitura

```
let url = 'mongodb://localhost:27017/teste';  
let cliente = await MongoClient.connect(url, {useNewUrlParser: true});  
let database = cliente.db('teste');  
let resultado = await bd.collection('teste').find({nome:'teste'}).toArray();
```

# MongoDB - Leitura

```
let url = 'mongodb://localhost:27017/teste';  
let cliente = await MongoClient.connect(url, {useNewUrlParser: true});  
let database = cliente.db('teste');  
let resultado = await bd.collection('teste').find({ $or: [{ nome: 'teste' }, {  
nome: 'outro' }]}).toArray();
```

# MongoDB - Alteração

- Métodos do objeto **Collection** para realizar alterações:
  - `updateOne()` – atualiza um único documento
  - `updateMany()` – atualiza múltiplos documentos
  - `replaceOne()` – substitui um documento por outro
- As operações utilizam o mesmo formato de critérios de busca que as operações de leitura para selecionar os documentos a serem modificados
- As alterações são definidas via operadores de alteração
  - Ver documentação em <https://docs.mongodb.com/manual/reference/operator/update/>
- Resultado da alteração é um objeto `updateWriteOpResult` contendo as propriedades:
  - `result` – objeto com o resultado da operação
  - `matchedCount` – número de documentos consultados
  - `modifiedCount` – número de documentos alterados
- As operações retornam *promises* se não fornecer um *callback*

# MongoDB - Alteração

```
let url = 'mongodb://localhost:27017/teste';  
let cliente = await MongoClient.connect(url, {useNewUrlParser: true});  
let database = cliente.db('teste');  
let resultado = await  
bd.collection('teste').updateOne({nome:'outro'},{$set:{idade:22}});
```

# MongoDB - Alteração

MongoDB Compass - localhost:27017/teste.colecaoteste

Connect View Collection Help

localhost:27017  
Community version 3.4.4

3 DBS 2 COLLECTIONS

filter

admin

local

teste

colecaoteste

teste.colecaoteste

DOCUMENTS 2 TOTAL SIZE 76B AVG. SIZE 38B INDEXES 1 TOTAL SIZE 32.0KB AVG. SIZE 32.0KB

SCHEMA DOCUMENTS INDEXES EXPLAIN PLAN VALIDATION

{ "filter" : "example" }

APPLY

Query returned 2 documents.

REFRESH

INSERT DOCUMENT

```
{ "_id": ObjectId('590f85ae2a51e42098c46ad9'), "nome": "teste" }
```

```
{ "_id": ObjectId('590f9cc2d386102e40ab24f7'), "nome": "outro" }
```

# MongoDB - Alteração

MongoDB Compass - localhost:27017/teste.colecaoteste

Connect View Collection Help

localhost:27017  
Community version 3.4.4

3 DBS 2 COLLECTIONS

filter

admin >

local >

teste v

colecaoteste

teste.colecaoteste

DOCUMENTS 2 TOTAL SIZE 76B AVG. SIZE 38B | INDEXES 1 TOTAL SIZE 32.0KB AVG. SIZE 32.0KB

SCHEMA

DOCUMENTS

INDEXES

EXPLAIN PLAN

VALIDATION

{ "filter" : "example" }

APPLY

Query returned 2 documents.

REFRESH

INSERT DOCUMENT

```
_id: ObjectId('590f85ae2a51e42098c46ad9')
nome: "teste"
```

```
_id: ObjectId('590f9cc2d386102e40ab24f7')
nome: "outro"
idade: 22
```



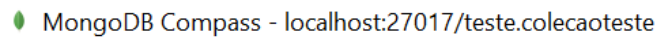
# MongoDB - Remoção

- Métodos do objeto **Collection** para realizar remoções:
  - `deleteOne()` – remove um único documento (o primeiro encontrado)
  - `deleteMany()` – remove múltiplos documentos
- As operações utilizam o mesmo formato de critérios de busca que as operações de leitura para selecionar os documentos a serem removidos
- Resultado da remoção é um objeto `deleteWriteOpCallback` contendo as propriedades:
  - `result` – objeto com o resultado da operação
  - `deletedCount` – número de documentos removidos
- As operações retornam *promises* se não fornecer um *callback*

# MongoDB - Remoção

```
let url = 'mongodb://localhost:27017/teste';  
let cliente = await MongoClient.connect(url, {useNewUrlParser: true});  
let database = cliente.db('teste');  
let resultado = await bd.collection('teste').deleteOne({nome:'outro'});
```

# MongoDB - Remoção



[Connect](#) [View](#) [Collection](#) [Help](#)

localhost:27017  
Community version 3.4.4

3 DBS 2 COLLECTIONS

Q filter

 admin >

 local >

 teste 

colecaoteste

teste.colecaoteste

DOCUMENTS	2	TOTAL SIZE	76B	AVG. SIZE	38B	INDEXES	1	TOTAL SIZE	32.0KB	AVG. SIZE	32.0KB
-----------	---	------------	-----	-----------	-----	---------	---	------------	--------	-----------	--------

## SCHEMA

## DOCUMENTS

## INDEXES

### EXPLAIN PLAN

## VALIDATION

```
i { "filter" : "example" }
```

## APPLY

Query returned **2** documents.

 REFRESH

INSERT DOCUMENT

```
_id: ObjectId('590f85ae2a51e42098c46ad9')
nome: "teste"
```

```
_id: ObjectId('590f9cc2d386102e40ab24f7')
nome: "outro"
idade: 22
```

# MongoDB - Remoção

MongoDB Compass - localhost:27017/teste.colecaoteste

Connect View Collection Help

localhost:27017  
Community version 3.4.4

3 DBS 2 COLLECTIONS

filter

admin >

local >

teste v

colecaoteste

teste.colecaoteste

DOCUMENTS	TOTAL SIZE	AVG. SIZE
2	76B	38B

INDEXES	TOTAL SIZE	AVG. SIZE
1	32.0KB	32.0KB

SCHEMA DOCUMENTS INDEXES EXPLAIN PLAN VALIDATION

{ "filter" : "example" }

APPLY

Query returned 1 document.

REFRESH

INSERT DOCUMENT

```
{
  "_id": ObjectId('590f85ae2a51e42098c46ad9'),
  "nome": "teste"
}
```

# MongoDB – Projeção e Agregação

- Operações de projeção permitem restringir os campos retornados via consultas
  - Via método *project*
  - Ver exemplo <http://mongodb.github.io/node-mongodb-native/3.1/tutorials/projections/>
- Operações de agregação computam um valor com base nos dados consultados (contagem, agrupamento, somatório, etc)
  - Via método *aggregate*
  - Operações descritas em <https://docs.mongodb.com/manual/core/aggregation-pipeline/>
  - Ver exemplo <http://mongodb.github.io/node-mongodb-native/3.2/tutorials/aggregation/>

# MongoDB - Validação

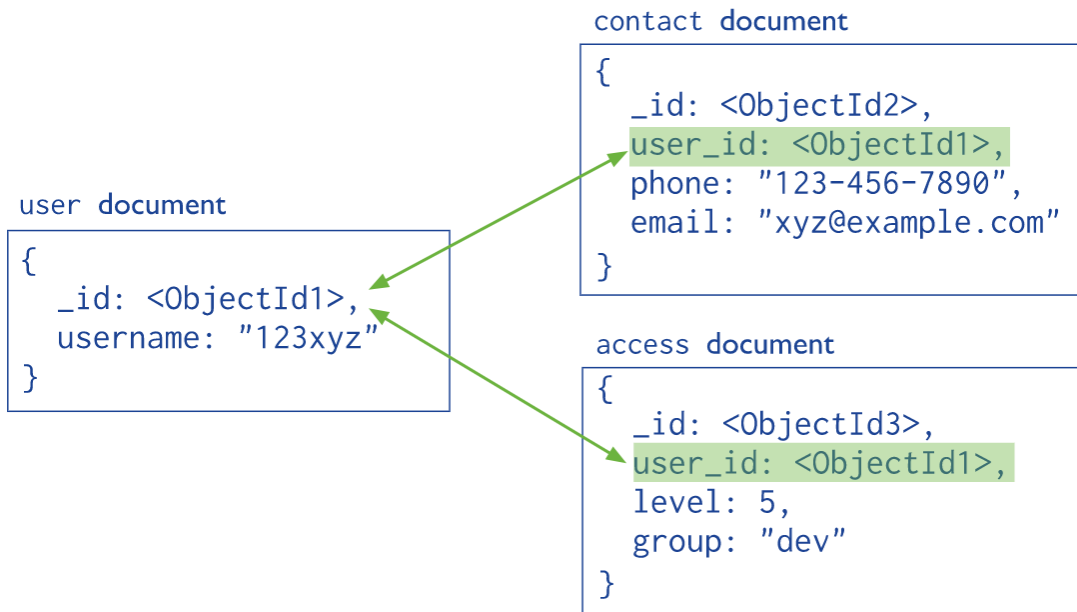
- MongoDB permite validar o esquema dos documentos ao realizar operações de inserção e alteração de dados
- Validações são definidas através de *validators* associados a uma coleção
- Para especificar as validações utiliza-se o padrão "JSON Schema"
  - Ver <http://json-schema.org/>
  - Documentação <https://docs.mongodb.com/manual/reference/operator/query/jsonSchema/>

# MongoDB - Validação

```
db.createCollection("students", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: [ "name", "year", "major", "gpa" ],
      properties: {
        name: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        gender: {
          bsonType: "string",
          description: "must be a string and is not required"
        },
        year: {
          bsonType: "int",
          minimum: 2017,
          maximum: 3017,
          exclusiveMaximum: false,
          description: "must be an integer in [ 2017, 3017 ] and is required"
        },
        major: {
          enum: [ "Math", "English", "Computer Science", "History", null ],
          description: "can only be one of the enum values and is required"
        },
        gpa: {
          bsonType: [ "double" ],
          description: "must be a double and is required"
        }
      }
    }
  }
})
```

# MongoDB – Estrutura de Documentos

- Duas funcionalidades são utilizadas na estruturação de documentos complexos:
  - Referências
  - Documentos embutidos





# MongoDB - Referências

- Referências representam relações entre documentos
  - Ver <https://docs.mongodb.com/manual/reference/database-references/>
- Uma aplicação precisa resolver a referência ao navegar de um documento para outro
  - Exemplos:
    - <https://docs.mongodb.com/manual/tutorial/model-referenced-one-to-many-relationships-between-documents/>

# MongoDB – Documento Embutido

- Documentos embutidos capturam a relação entre dados através do armazenamento dos dados relacionados dentro de uma única estrutura de documento
- Exemplos:
  - <https://docs.mongodb.com/manual/tutorial/model-embedded-one-to-one-relationships-between-documents/>
  - <https://docs.mongodb.com/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/>

# Mongoose



# Mongoose

- Mongoose é um *Object Data Mapper* para o MongoDB
  - Utiliza *models* que são responsáveis pelo mapeamento de objetos JavaScript aos documentos do MongoDB
- É uma alternativa ao uso do driver nativo no Node.JS
  - Instalar via comando `npm install mongoose --save`
  - Instalar tipos TypeScript via comando `npm install @types/mongoose --save-dev`

<http://mongoosejs.com/>

# Mongoose - Conexão

- A primeira etapa é configurar uma conexão com o servidor do MongoDB
- O servidor a ser utilizado é feito via uma string de conexão
  - Ver documentação em <http://docs.mongodb.org/manual/reference/connection-string/>

```
import * as mongoose from 'mongoose';  
const url = 'mongodb://localhost:27017/testemongoose';  
const cliente = await mongoose.connect(url, { useNewUrlParser: true });
```

# Mongoose – Esquemas e Modelos

- Um **Schema** define a estrutura e as propriedades dos documentos
  - Ver tipos suportados em <http://mongoosejs.com/docs/schematypes.html>
  - Ao definir o esquema é possível definir regras de validação
    - Ver documentação em <http://mongoosejs.com/docs/validation.html>
- Um **Model** é o resultado da compilação de um schema
  - O modelo é o objeto a ser utilizado para a criação de documentos
  - Cada modelo mapeia para uma coleção no MongoDB
  - Ver documentação em <http://mongoosejs.com/docs/models.html>
- Um documento é uma instância de um modelo
  - Representam um mapeamento um-para-um com o MongoDB
  - Possuem os métodos para operações CRUD

# Mongoose – Esquemas e Modelos

```
...  
const pessoaEsquema = new mongoose.Schema({  
  nome: String,  
  idade: Number  
});  
  
const pessoaModelo = mongoose.model('Pessoa', pessoaEsquema, 'pessoas');  
  
let documento = new pessoaModelo({ nome: 'John Doe', idade: 22 });
```

# Mongoose - Validação

- Mongoose oferece validadores padrão e a possibilidade de criar os próprios validadores para os esquemas
  - Ver documentação em <http://mongoosejs.com/docs/validation.html>
- Exemplo de validadores padrão:
  - Todos os tipo possuem validador *required* para tornar um campo obrigatório
  - *Number* possui validadores *min* e *max* para definição de intervalos
  - *String* possui validadores *enum* (para conjunto de valores aceitos), *match* (para expressão regular), *minlength* e *maxlength* (para tamanho)



# Mongoose - Validação

```
const cafedamanhaSchema = new Schema({
  ovos: {
    type: Number,
    min: [2, 'Poucos ovos'],
    max: 12
    required: [true, 'Ovos obrigatórios']
  },
  bebida: {
    type: String,
    enum: ['Café', 'Chá', 'Água',]
  }
});
```

# Mongoose - Inserção

- Para inserir documentos utiliza-se a função **save()**

```
...  
const pessoaEsquema = new mongoose.Schema({  
  nome: String,  
  idade: Number  
});  
const pessoaModelo = mongoose.model('Pessoa', pessoaEsquema, 'pessoas');  
let documento = new pessoaModelo({ nome: 'John Doe', idade: 22 });  
let resultado = await documento.save();
```

# Mongoose - Inserção

MongoDB Compass - localhost:27017/testmongoose.pessoas

Connect View Collection Help

localhost:27017  
Community version 3.4.4

4 DBS 3 COLLECTIONS

filter

admin >

local >

teste >

testmongoose >

pessoas

testmongoose.pessoas

DOCUMENTS 1 TOTAL SIZE 61B AVG. SIZE 61B INDEXES 1 TOTAL SIZE 16.0KB AVG. SIZE 16.0KB

SCHEMA DOCUMENTS INDEXES EXPLAIN PLAN VALIDATION

{ "filter" : "example" }

APPLY

Query returned 1 document.

REFRESH

INSERT DOCUMENT

```
{
  "_id": ObjectId('590fc14aa8e8220fc4033b33'),
  "nome": "John Doe",
  "idade": 22,
  "__v": 0
}
```

# Mongoose - Leitura

- Métodos do objeto **Model** para realizar consultas:
  - find() – retorna o(s) documento(s) de acordo com o critério de busca
  - findOne() – retorna o primeiro documento de acordo com o critério de busca
  - findById() – retorna um único documento
- Critérios de busca:
  - Ver documentação <https://docs.mongodb.com/manual/reference/operator/query/#query-selectors>
- As operações retornam um objeto **Query** quando não se fornece um *callback*
  - Permite construir uma consulta pela composição de métodos where(), select(), sort(), etc
  - Ver documentação em <http://docs.mongodb.org/manual/reference/operator/query/>
  - A consulta é finalmente executada pelo método exec(), o qual retorna uma *promise*

# Mongoose - Leitura

```
...  
let consulta = pessoaModelo.findOne({nome: 'John Doe'});  
let resultado = await consulta.exec();
```

```
...  
let consulta = pessoaModelo.find().where('nome').equals('John Doe');  
let resultado = await consulta.exec();
```

# Mongoose - Alteração

- A maneira mais direta de alterar um documento é consultá-lo primeiro, alterar qualquer uma de suas propriedades e executar o método `save()`
- Métodos do objeto **Model** para realizar alterações:
  - `updateOne()` – atualiza documento
  - `updateMany()` – atualiza vários documentos
  - `findOneAndUpdate()` – atualiza um único documento e o retorna
  - `findByIdAndUpdate()` – atualiza um único documento e o retorna
  - `replaceOne()` – substitui um documento por outro
- As operações utilizam o mesmo formato de critérios de busca que as operações de leitura para selecionar os documentos a serem modificados
- As alterações são definidas via operadores de alteração
  - Ver documentação em <https://docs.mongodb.com/manual/reference/operator/update/>

# Mongoose - Alteração

```
...  
await pessoaModel.updateOne({ nome: 'John Doe' }, { $set: { idade: 33 } }).exec();
```

# Mongoose - Remoção

- Métodos do objeto **Model** para realizar remoções:
  - `removeOne()` – remove um documento de acordo com o critério
  - `removeMany()` – remove vários documentos de acordo com o critério
  - `findOneAndRemove()` – remove um único documento
  - `findByIdAndRemove()` – remove um único documento
- As operações utilizam o mesmo formato de critérios de busca que as operações de leitura para selecionar os documentos a serem removidos



# Mongoose - Remoção

```
...  
await pessoaModel.removeOne({ nome: 'teste' }).exec();
```

# Mongoose – Relacionamento Entre Documentos

- MongoDB permite que documentos apresentem referências entre si
  - Relacionamento para um via campo do tipo ObjectId
  - Relacionamento para vários via campo do tipo array de ObjectId
- Ao realizar consultas, utiliza-se o método **populate()** para realizar a busca do documento associado
  - Ver documentação em <http://mongoosejs.com/docs/populate.html>

# Mongoose – Relacionamento Entre Documentos

```
...  
  
const authorSchema = Mongoose.Schema({  
  name: String,  
  stories: [{ type: Mongoose.Schema.Types.ObjectId, ref: 'Story' }]  
});  
  
const storySchema = Mongoose.Schema({  
  author: { type: Mongoose.Schema.Types.ObjectId, ref: 'Author' },  
  title: String  
});  
  
const Story = Mongoose.model('Story', storySchema);  
const Author = Mongoose.model('Author', authorSchema);
```

# Mongoose – Relacionamento Entre Documentos

```
...  
let bob = new Author({ name: 'Bob Smith' });  
await bob.save();  
  
let story = new Story({  
  title: "Bob goes sledding",  
  author: bob._id  
});  
await story.save();
```

# Mongoose – Relacionamento Entre Documentos

MongoDB Compass - localhost:27017/testmongoose.authors

Connect View Collection Help

localhost:27017  
Community version 3.4.4

4 DBS 5 COLLECTIONS

filter

admin >

local >

teste >

testmongoose >

authors

pessoas

stories

testmongoose.authors

DOCUMENTS	1	TOTAL SIZE	65B	AVG. SIZE	65B
INDEXES	1	TOTAL SIZE	16.0KB	AVG. SIZE	16.0KB

SCHEMA DOCUMENTS INDEXES EXPLAIN PLAN VALIDATION

{ "filter" : "example" }

APPLY

Query returned 1 document.

REFRESH

INSERT DOCUMENT

```
{
  "_id": ObjectId('590fd31aa1d5fc1b8cb43566'),
  "name": "Bob Smith",
  "stories": Array,
  "__v": 0
}
```

# Mongoose – Relacionamento Entre Documentos

MongoDB Compass - localhost:27017/testmongoose.stories

Connect View Collection Help

localhost:27017  
Community version 3.4.4

4 DBS 5 COLLECTIONS

filter

admin >

local >

teste >

testmongoose ▾

authors

pessoas

stories

testmongoose.stories

DOCUMENTS 1 TOTAL SIZE 80B AVG. SIZE 80B INDEXES 1 TOTAL SIZE 16.0KB AVG. SIZE 16.0KB

SCHEMA DOCUMENTS INDEXES EXPLAIN PLAN VALIDATION

{ "filter" : "example" }

APPLY

Query returned 1 document.

REFRESH

INSERT DOCUMENT

```
_id: ObjectId('590fd31ba1d5fc1b8cb43567')
title: "Bob goes sledding"
author: ObjectId('590fd31aa1d5fc1b8cb43566')
__v: 0
```

# Mongoose – Relacionamento Entre Documentos

```
...  
await Story  
  .findOne({ title: 'Bob goes sledding' })  
  .populate('author')  
  .exec();
```

# Laboratório

- Abra as instruções do arquivo Lab05\_MongoDB

