



Persistência de Dados, Versionamento e Implantação

Bruno Augusto Teixeira

2021

Persistência de Dados, Versionamento e Implantação

Bruno Augusto Teixeira

© Copyright do Instituto de Gestão e Tecnologia da Informação.

Todos os direitos reservados.

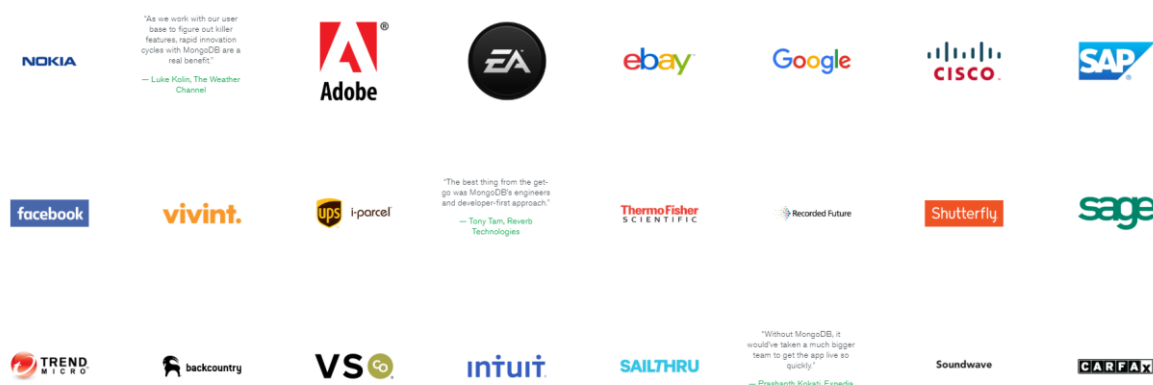
Sumário

Capítulo 1. Introdução ao MongoDB	4
1.1. Instalação.....	8
1.2. Banco de dados	15
1.3. Coleções	16
1.4. Inserir documentos (Create)	18
1.5. Consultar documentos (retrieve)	20
1.6. Atualizar documentos (update)	25
1.7. Exclusão de documentos (delete)	28
1.8. Comandos em massa (BulkWrite)	29
1.9. Índices.....	30
1.10. Modelagem	33
1.11. Agregações.....	34
Capítulo 2. MongoDB Atlas	37
2.1. O serviço MongoDB Atlas	37
2.1. Utilização do serviço	37
2.1. Integração ao MongoDB	41
Capítulo 3. Git e GitHub.....	44
3.1. Introdução ao Git	44
3.2. Instalação e configuração do Git.....	45
3.3. Principais comandos do Git	49
3.4. GitHub.....	57
Capítulo 4. Heroku.....	65
4.1. Introdução ao Heroku.....	65
4.2. Instalação do Heroku CLI.....	66
4.3. Implantação no Heroku	67
Referências.....	71

Capítulo 1. Introdução ao MongoDB

A utilização de bancos não relacionais (NoSQL) tem se tornado cada vez mais predominante no desenvolvimento de softwares modernos nos últimos anos. O MongoDB é um banco de dados NoSQL orientado à objetos que vem sendo adotado tanto em startups quanto em pequenas e grandes corporações, conforme Figura 1.

Figura 1 – Clientes do MongoDB.



Fonte: MongoDB Customers (2020).

Seu crescimento nos últimos anos o define como um dos principais dentro das opções para os bancos NoSQL, conforme Figura 2.

Figura 2 – Ranking geral de DBMS.

Rank	Rank			DBMS	Database Model	Score		
	Apr 2020	Mar 2020	Apr 2019			Apr 2020	Mar 2020	Apr 2019
1.	1.	1.	1.	Oracle +	Relational, Multi-model	1345.42	+4.78	+65.48
2.	2.	2.	2.	MySQL +	Relational, Multi-model	1268.35	+8.62	+53.21
3.	3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model	1083.43	-14.43	+23.47
4.	4.	4.	4.	PostgreSQL +	Relational, Multi-model	509.86	-4.06	+31.14
5.	5.	5.	5.	MongoDB +	Document, Multi-model	438.43	+0.82	+36.45
6.	6.	6.	6.	IBM Db2 +	Relational, Multi-model	165.63	+3.07	-10.42
7.	7.	8.	8.	Elasticsearch +	Search engine, Multi-model	148.91	-0.26	+2.91
8.	8.	7.	7.	Redis +	Key-value, Multi-model	144.81	-2.77	-1.57
9.	10.	10.	10.	SQLite +	Relational	122.19	+0.24	-2.02
10.	9.	9.	9.	Microsoft Access	Relational	121.92	-3.22	-22.73
11.	11.	11.	11.	Cassandra +	Wide column	120.07	-0.88	-3.54
12.	13.	12.	12.	MariaDB +	Relational, Multi-model	89.90	+1.55	+4.67
13.	12.	13.	13.	Splunk	Search engine	88.08	-0.44	+4.99
14.	14.	15.	15.	Hive	Relational	84.05	-1.32	+9.34

Fonte: DB-Engines (2020).

Embora o MongoDB possua toda a relevância supracitada, é importante compreender as principais diferenças entre bancos de dados *SQL* e *NoSQL* para possibilitar a melhor definição da solução a ser utilizada na arquitetura da aplicação a ser desenvolvida.

Bancos de dados SQL:

Conhecidos como bancos de dados relacionais (RDBMS), os bancos SQL definem e manipulam dados com base na linguagem de consulta estruturada (SQL). Eles são mais populares e úteis para manipular dados estruturados, e possui uma organização e padronização de como os elementos de dados relacionam entre si mesmo com diversas propriedades.

Bancos de dados NoSQL:

Os bancos NoSQL são conhecidos como bancos de dados não relacionais, que permitem armazenar e recuperar dados não estruturados usando um esquema dinâmico. O NoSQL é popularmente usado por sua capacidade flexível de criar uma estrutura exclusiva e dinâmica com uma diversidade de dados dentro de uma mesma estrutura.

O MongoDB é um banco de dados orientado à objetos, comumente chamados de documentos, criado em 2009, de multiplataforma e de código fonte aberto, sendo uma das bases da atualidade que segue o conceito *NoSQL*. Suas características evitam estruturas (schemas) de tabelas organizadas com linhas e colunas, como ocorrem nos bancos relacionais. O formato dos arquivos armazenados, denominado *BSON*, é uma versão binária do formato *JSON* (Java Script Object Notation), que é utilizado em diversos protocolos de comunicação. Isso torna a integração de dados para certos tipos de aplicativos mais rápida e fácil.

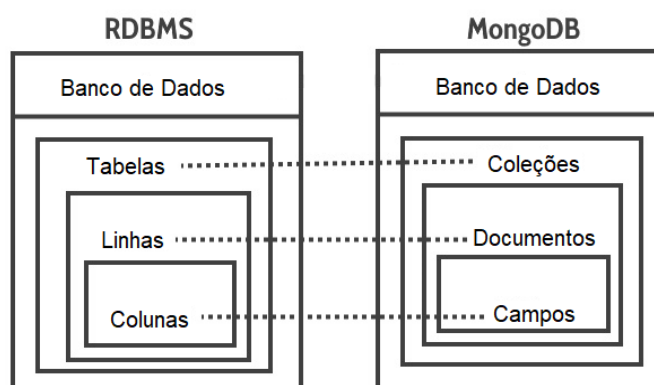
A organização dos dados no MongoDB é definida conforme a hierarquia abaixo:

- **Banco de dados:** pode conter uma ou mais coleções.

- **Coleções:** pode conter diferentes tipos de documentos.
- **Documentos:** pode conter tuplas de chave e valor em lista, vetor ou uma referência à um documento.

A Figura 3 apresenta a relação entre as estruturas de um banco de dados SQL com um banco NoSQL (MongoDB).

Figura 3 – Comparação de estrutura entre RDBMS e MongoDB.



Fonte: Adaptado de Google Images.

JSON e BSON:

O formato *JSON* é organizado com campos de chave-valor, no qual possui alguns pontos característicos, conforme exemplificado na Figura 4:

- As chaves são do tipo “String”.
- O delimitador entre chave-valor é o “.” (dois pontos).
- Cada entrada “chave-valor” é separada por “,” (vírgula).
- Cada objeto JSON é delimitado por “{}” (Chaves).

Figura 4 – Estrutura do formato JSON.

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

← field: value
 ← field: value
 ← field: value
 ← field: value

Fonte: MongoDB (2020).

O *BSON* foi criado para armazenar ou transferir *JSON*, de modo a evitar perda de caracteres, conforme definido em sua especificação (*BSON*). Os três principais pilares desse formato são:

1. **Ser leve:** manter a sobrecarga no espaço mínimo é importante para qualquer formato de representação de dados, especialmente quando usado na rede.
2. **Transportável:** ele foi desenhado para ser facilmente transportável.
3. **Eficiência:** a codificação de dados para *BSON* e a decodificação de *BSON* podem ser realizadas rapidamente na maioria das linguagens, devido ao uso de tipos de dados C.

Existem diversos tipos de dados que podem ser armazenados no formato *BSON*, os principais e mais utilizados são destacados na Figura 4, e os complementares (expressão regular, código Java Script, TimeStamp, Long, Binary Data, Min Key e Max Key) podem ser consultados na documentação do MongoDB.

Figura 4 – Tipo de dados do BSON.

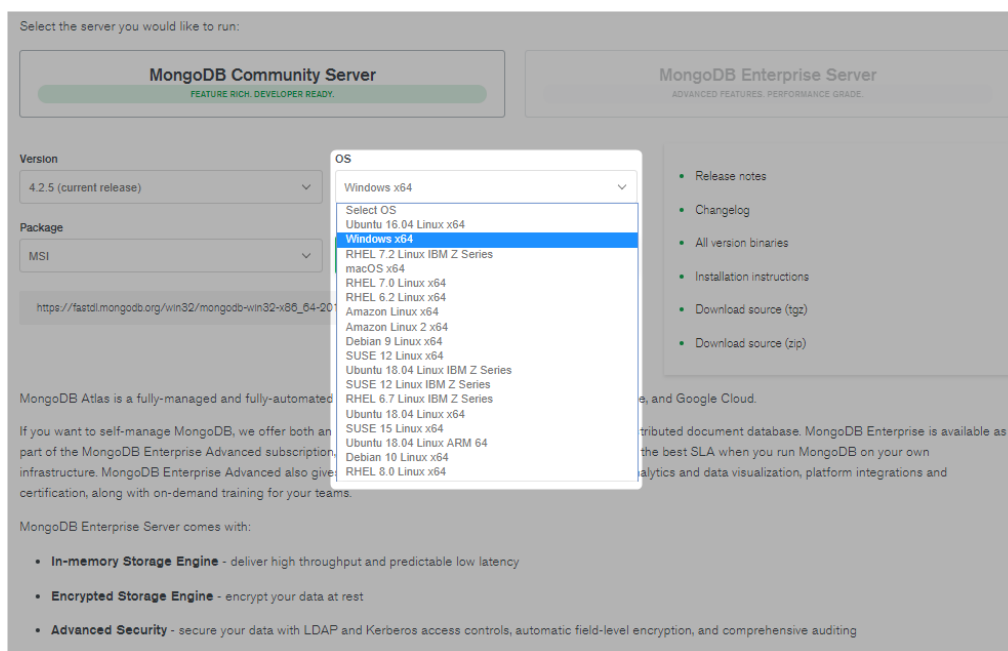
```
{
  string: "Algum texto",
  int: 5,
  double: 5.5,
  boolean: true,
  array: [1, 2, 3],
  object: {campo1: "valor", campo2: "valor"},
  date: new Date("<YYYY-mm-dd>"),
  object_id: <ObjectId>,
  nulo: null
}
```

As próximas seções deste capítulo apresentarão o processo para instalação, as operações CRUD e a administração do banco.

1.1. Instalação

O MongoDB possui duas versões de instalação on premise, a Community Server e a Enterprise Server, sendo uma gratuita e a outra precisando de licença, respectivamente. A versão Enterprise possui alguns recursos avançados que a necessidade para o negócio precisa ser avaliada. Portanto, para esse curso será adotada a versão Community Server 4.2.5, que pode ser instalada em várias plataformas, conforme Figura 5. Os comandos e funcionalidades das versões anteriores que foram descontinuados não serão apresentados neste curso.

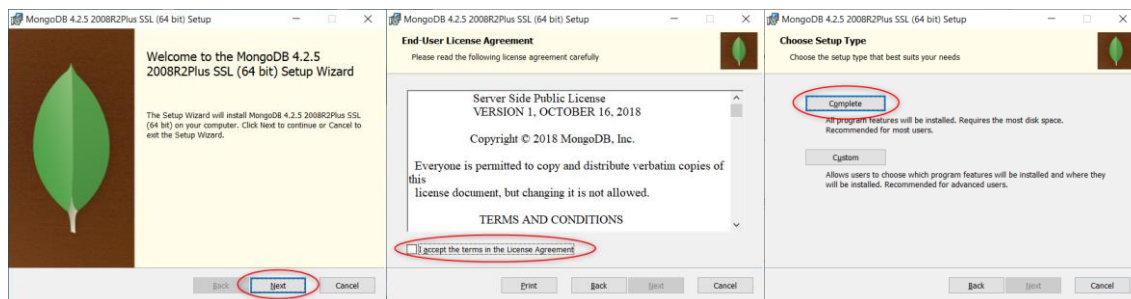
Figura 5 – Plataformas Instalação MongoDB Community Server.



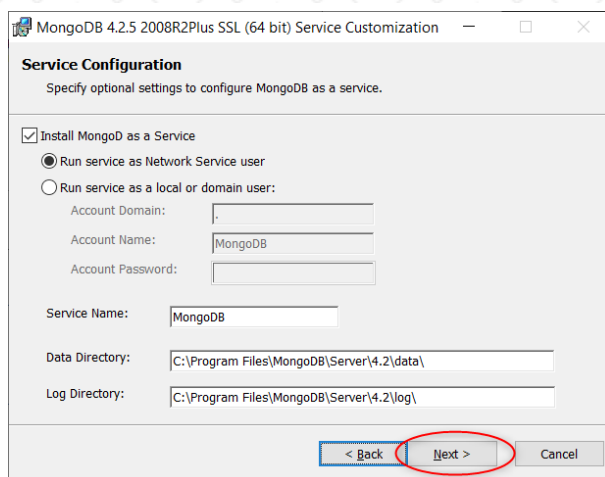
Fonte: MongoDB (2020).

Instalação Windows x64:

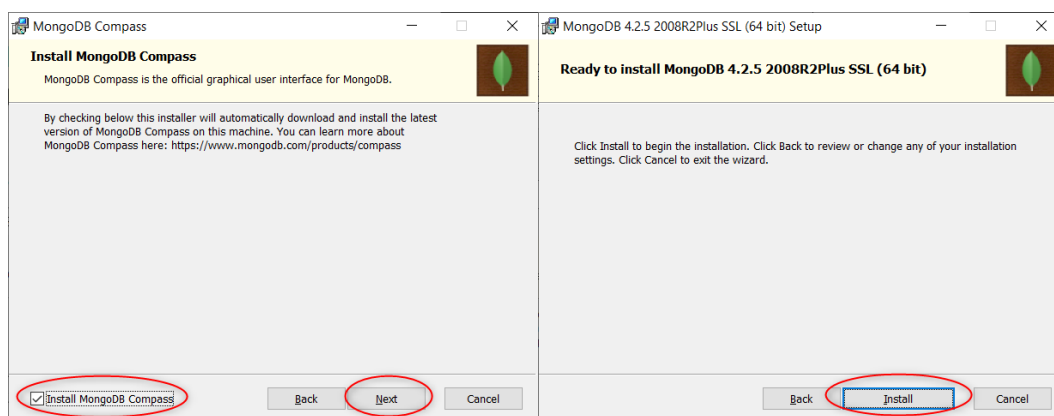
Após o download do arquivo .msi, basta seguir os passos abaixo:



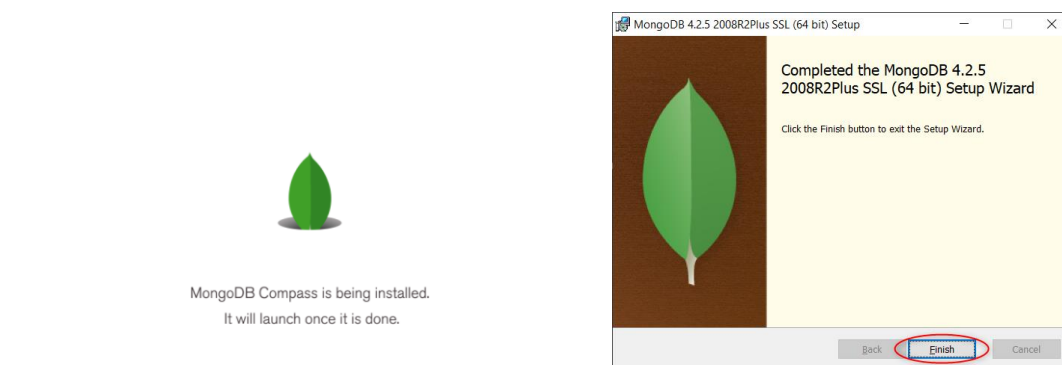
Caso o serviço do MongoDB deva ser executado por um usuário específico, a opção *“Run service as a local or domain user”* deve ser selecionada.



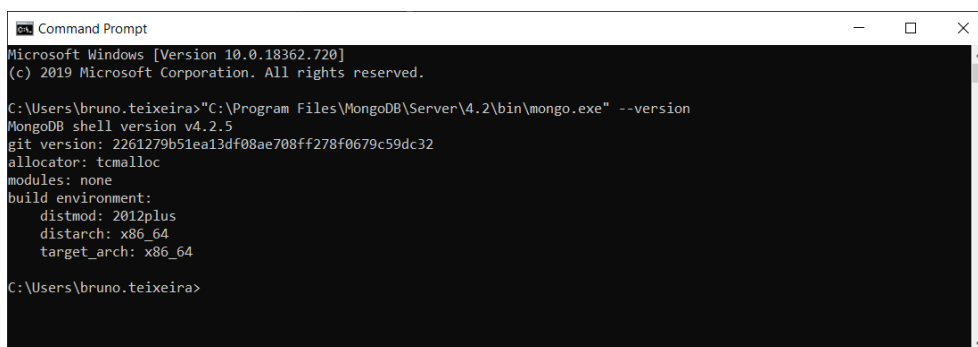
O MongoDB Compass é um GUI que permite a visualização das estruturas dos documentos, execução de consultas, criação de indexes etc. O *Compass* é um acelerador para análises visuais e a instalação será realizada para fins didáticos, embora seja optativa.



A instalação exibirá o pop-up do progresso, além da tela a seguir:



A instalação do MongoDB foi realizada e a execução dos comandos via shell script pode ser realizada utilizando o executável **mongo.exe**, que comumente está localizado dentro da pasta bin: “C:\Program Files\MongoDB\Server\4.2\bin\mongodb.exe”, conforme a figura abaixo:



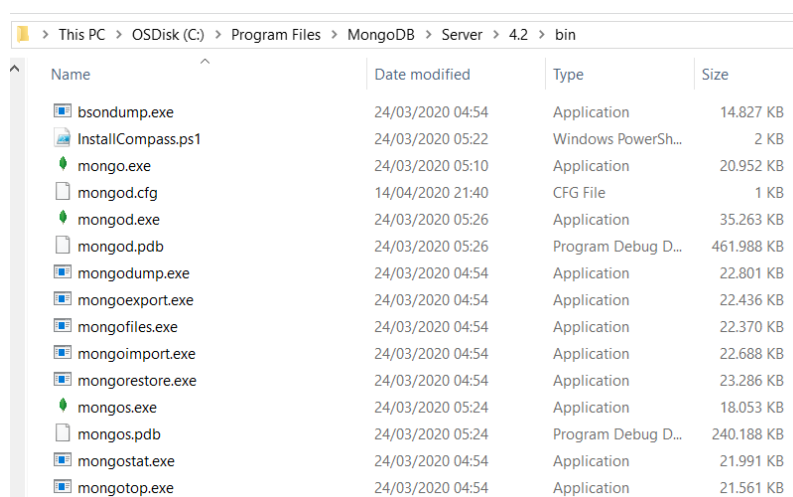
```

Command Prompt
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\bruno.teixeira>"C:\Program Files\MongoDB\Server\4.2\bin\mongo.exe" --version
MongoDB shell version v4.2.5
git version: 2261279b51ea13df08ae708ff278f0679c59dc32
allocator: tcmalloc
modules: none
build environment:
  distmod: 2012plus
  distarch: x86_64
  target_arch: x86_64

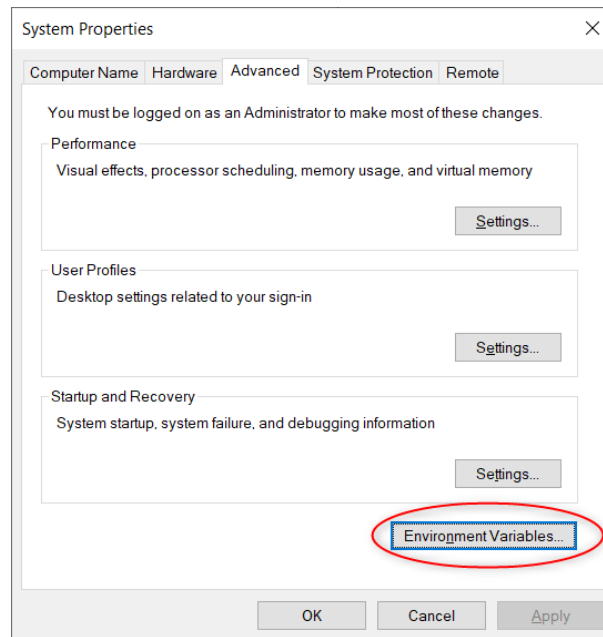
C:\Users\bruno.teixeira>
  
```

Localização dos arquivos instalados do MongoDB:

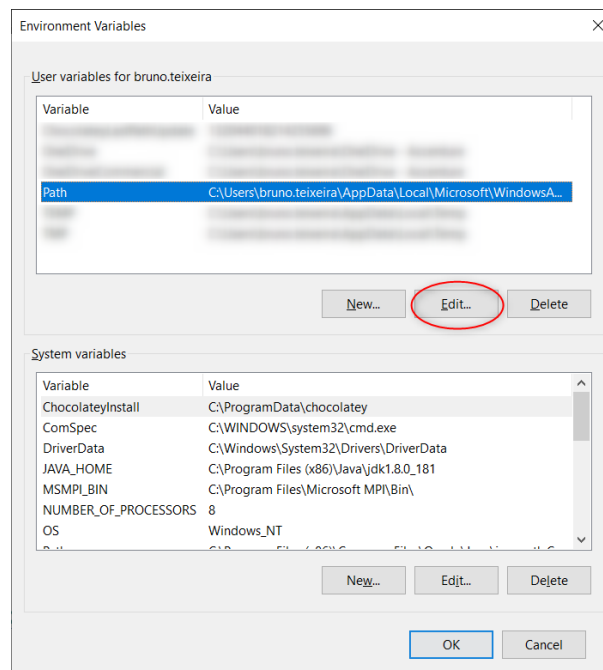


Name	Date modified	Type	Size
bsondump.exe	24/03/2020 04:54	Application	14.827 KB
InstallCompass.ps1	24/03/2020 05:22	Windows PowerSh...	2 KB
mongo.exe	24/03/2020 05:10	Application	20.952 KB
mongod.cfg	14/04/2020 21:40	CFG File	1 KB
mongod.exe	24/03/2020 05:26	Application	35.263 KB
mongod.pdb	24/03/2020 05:26	Program Debug D...	461.988 KB
mongodump.exe	24/03/2020 04:54	Application	22.801 KB
mongoexport.exe	24/03/2020 04:54	Application	22.436 KB
mongoimport.exe	24/03/2020 04:54	Application	22.688 KB
mongorestore.exe	24/03/2020 04:54	Application	23.286 KB
mongos.exe	24/03/2020 05:24	Application	18.053 KB
mongos.pdb	24/03/2020 05:24	Program Debug D...	240.188 KB
mongostat.exe	24/03/2020 04:54	Application	21.991 KB
mongotop.exe	24/03/2020 04:54	Application	21.561 KB

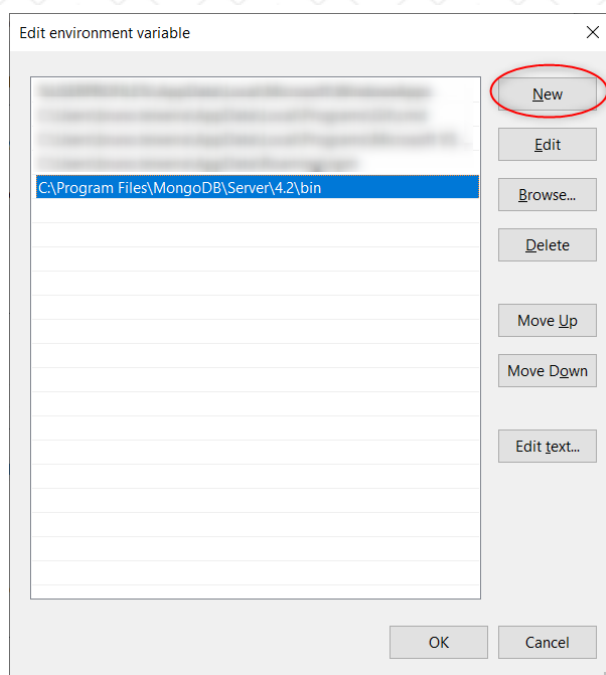
A fim de evitar o acesso pela linha de comando com o caminho extenso, o próximo passo definirá o executável no Path das Variáveis de Ambiente do Windows. Desta forma, os passos para inclusão no Path estão detalhados a seguir:



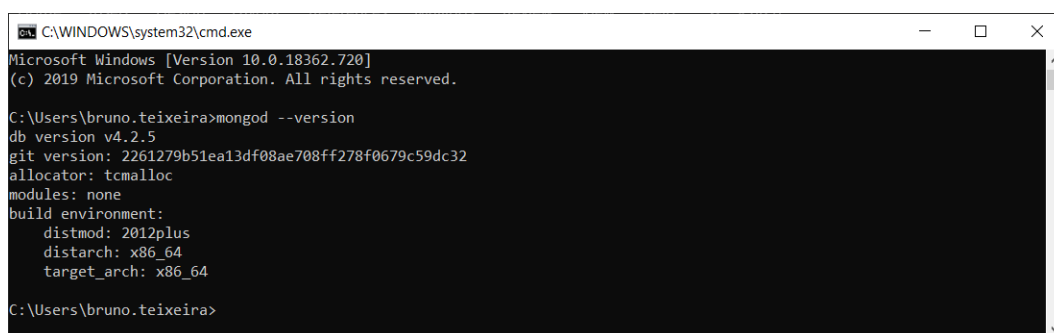
Acessar a variável Path e Editar:



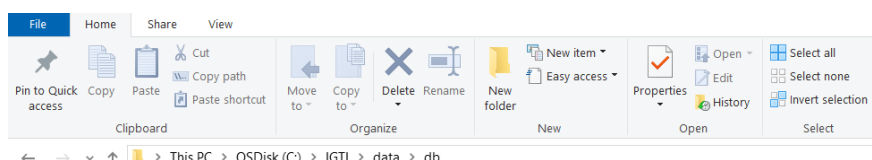
Adicionar uma nova entrada com o caminho da pasta bin do MongoDB:



Finalizar e validar que o acesso aos comandos do banco já pode ser acessado sem a necessidade de ter o caminho completo da pasta bin, conforme figura abaixo:



O MongoDB possui um diretório default para os dados conforme definido na instalação, porém nesse curso iremos definir um diretório que irá armazenar os dados. Para isso, basta criar um novo diretório



Digitar o seguinte comando na linha de comando: `mongod -dbpath="CAMINHO_CRIADO_CONFORME_PASSO_ANTERIOR"`.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\bruno.teixeira>mongod --version
db version v4.2.5
git version: 2261279b51ea13df08ae708ff278f0679c59dc32
allocator: tcmalloc
modules: none
build environment:
  distmod: 2012plus
  distarch: x86_64
  target_arch: x86_64

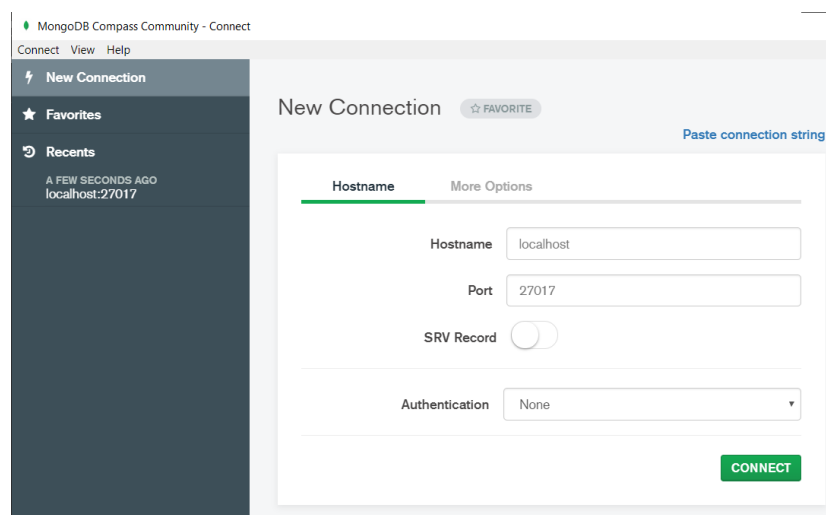
C:\Users\bruno.teixeira>mongod --dbpath="C:\IGTI\data\db"
```

O acesso ao banco via linha de comando para execução dos comandos nos dados, é realizado pelo comando: `mongo -host localhost:27017`.

```
Command Prompt
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\bruno.teixeira>mongo --host localhost:27017
```

O mesmo acesso pode ser realizado pelo Compass:



1.2. Banco de dados

O MongoDB não possui diretiva DDL para criação de um banco de dados antes da inserção dos documentos, como é realizado tradicionalmente nos bancos de dados SQL. Portanto, não é necessária a criação de um banco de dados manualmente, pois o MongoDB o criará automaticamente quando um documento for salvo numa de suas coleções.

Criação:

O comando `use [NOME_DO_BANCO]` é utilizado para acessar um banco de dados no MongoDB. Caso o banco não exista, automaticamente o banco é criado. A figura abaixo apresenta a criação do banco **IGTI_DATABASE**, que não existia no servidor e foi criado.

```
> use IGTI_DATABASE
switched to db IGTI_DATABASE
```

Consultar:

O comando `show dbs` é utilizado para consultar as bases existentes no servidor do banco de dados. A Figura abaixo apresenta os bancos padrões já existentes em decorrência da instalação do MongoDB.

```
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
```

Note que se executarmos o comando para consultar os bancos existentes no servidor, o banco recentemente criado não será listado, pois o MongoDB só o cria após a inserção de um documento no banco, conforme abaixo:

```
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
```

Caso um documento seja inserido, o banco será listado conforme abaixo:

```
> db.medicamentos.insert({"produto": "DIPIRONA"})
WriteResult({ "nInserted" : 1 })
> show dbs
IGTI_DATABASE 0.000GB
admin         0.000GB
config        0.000GB
local         0.000GB
```

O comando `db` é utilizado para consultar o banco de dados corrente, que está sendo manipulado, conforme abaixo:

```
> db
IGTI_DATABASE
```

Apagar (drop):

O comando `db.dropDatabase()` é utilizado para apagar uma base de dados no servidor do MongoDB, após a sua execução podemos consultar as bases existentes com o comando `show dbs`.

```
> db.dropDatabase()
{ "dropped" : "IGTI_DATABASE", "ok" : 1 }
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
```

1.3. Coleções

As coleções possuem a mesma característica dos bancos de dados, o MongoDB automaticamente cria uma coleção com a inserção direta de um documento, portanto, não é necessária a criação explícita de uma coleção *a priori* para se inserir um documento.

Criação:

A sintaxe para criação de coleções é pelo comando `db.createCollection(name, options)`, onde `name` define o nome da coleção e `options` define algumas configurações para a coleção. A Tabela apresenta as principais opções que podem ser utilizadas opcionalmente:

Campo	Tipo	Descrição
capped	boolean	Utilizado para criar uma coleção limitada “ <i>capped collection</i> ” se especificar como <code>true</code> . Se especificar como <code>true</code> , o campo <code>size</code> deve ser definido.
size	number	Tamanho em <code>bytes</code> para definir o limite de uma coleção limitada “ <i>capped collection</i> ”. Parâmetro obrigatório caso o campo <code>capped</code> seja <code>true</code> .
max	number	Determina o limite máximo de documentos de uma coleção limitada.
validator	document	Documento que define regras ou expressões de validação para os documentos. A validação ocorre durante as inserções (<code>insert</code>) e atualizações (<code>update</code>) de documentos.
validationLevel	string	Determina com que rigor as regras de validação são aplicadas aos documentos existentes durante uma atualização.
validationAction	string	Determina se acusa erro em documentos inválidos ou apenas alerta sobre as violações, mas permite que documentos inválidos sejam inseridos.

A Figura abaixo apresenta a criação de uma coleção explicitamente sem os parâmetros opcionais:

```
> use IGTI_DATABASE
switched to db IGTI_DATABASE
> show collections
> db.createCollection("medicamentos")
{ "ok" : 1 }
```

A Figura abaixo apresenta a criação de uma *capped collection* (coleção limitada) com limite máximo de 10000 documentos ou ~6MB.

```
> db.createCollection("medicamentos", { capped : true, size : 6142800, max : 10000 } )
{ "ok" : 1 }
```

Consultar:

O comando `show collections` é utilizado para listar as coleções existentes no banco de dados, conforme a figura abaixo:

```
> show collections
medicamentos
```

Apagar (drop):

O comando `db.COLLECTION.drop()` é utilizado para apagar uma coleção de uma base de dados, após a sua execução podemos consultar as coleções existentes com o comando `show collections`.

```
> db.medicamentos.drop()
true
> show collections
```

1.4. Inserir documentos (Create)

O MongoDB possui os seguintes métodos de inserção:

- `db.collection.insertOne()` – Insere um único documento na coleção.
- `db.collection.insertMany()` – Insere múltiplos documentos na coleção.
- `db.collection.insert()` – Insere um ou mais documentos na coleção.

Os métodos `insertOne()` e `insertMany()` retornam um `document` com o retorno da inserção e o(s) `id(s)` dos objetos inseridos; ao passo que o método `insert()` retorna um `WriteResult`, para a inserção de um único documento, e um `BulkWriteResult`, quando é feita a inserção de múltiplos documentos. Além disso, os métodos `insertOne()` e `insertMany()` não são compatíveis com o método `db.collection.explain()` para avaliação do plano de inserção do documento na coleção. Embora o método `insert()` possua um comportamento genérico para

inserção de um ou mais documentos, preferencialmente os métodos `insertOne()` e `insertMany()` são mais utilizados na prática.

A Figura abaixo apresenta o comportamento para a inserção de um registro pelo método `insert()`.

```
> db.medicamentos.insert({produto:"DIPIRONA", tarja: "Venda Livre", tipo: "Genérico"})
WriteResult({ "nInserted" : 1 })
```

A Figura abaixo apresenta o comportamento para a inserção de um registro pelo método `insertOne()`. Conforme apresentado na figura, o MongoDB retorna o ID do documento criado caso este valor não seja definido, explicitamente o MongoDB cria um identificador único.

```
> db.medicamentos.insertOne({produto:"NOVALGINA", tarja: "Venda Livre", tipo: "Novo"})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5e98abff0193f51199bb8ce5")
}
```

O campo `_id` é reservado para a identificação dos documentos e não pode ser definido como outro tipo de campo, para os documentos, seu valor é automaticamente definido pelo MongoDB quando não é definido na criação do documento. No exemplo da Figura abaixo, o campo `_id` foi definido manualmente:

```
> db.medicamentos.insertOne({_id: 3, produto:"NEOSALDINA", tarja: "Tarja Vermelha", tipo: "Novo", preco: 11.52})
{ "acknowledged" : true, "insertedId" : 3 }
```

O MongoDB possui uma flexibilidade para a estrutura de sua base, ou seja, não possui um schema rígido no qual todos os campos devem ser definidos e novos campos podem ser criados sem terem sido criados previamente. Na Figura acima, podemos observar que o campo `preco` foi incluído mesmo após a inserção de outros documentos na coleção.

A Figura abaixo apresenta a utilização do método `insertMany()` com novos campos para cada documento inserido:

```
> db.medicamentos.insertMany([{produto:"BENEGRIPI", tarja: "Tarja Vermelha", tipo: "Novo", preco: 9.36},{produto:"NIMESULIDA", tarja: "Tarja Vermelha", tipo: "Genérico", substancia: "Nimesulida"},{produto: "NOEX", tarja: "Tarja Preta", substancia:"BUDESONIDA", preco: 33.93, laboratorio: "EUROFARMA LABORATÓRIOS S.A."}])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5e98b2402feb1bbca3923292"),
    ObjectId("5e98b2402feb1bbca3923293"),
    ObjectId("5e98b2402feb1bbca3923294")
  ]
}
```

1.5. Consultar documentos (retrieve)

O método `db.COLLECTION.find(query, projection)` é utilizado para consultar os documentos no banco. Este método projeta (`projection`) os campos especificados dos documentos que atendem os critérios de uma `query` definida. A Figura abaixo apresenta o retorno do método `find()`.

```
> db.medicamentos.find()
{ "_id" : ObjectId("5e98ab600193f51199bb8ce4"), "produto" : "DIPIRONA", "tarja" : "Venda Livre", "tipo" : "Genérico" }
{ "_id" : ObjectId("5e98abff0193f51199bb8ce5"), "produto" : "NOVALGINA", "tarja" : "Venda Livre", "tipo" : "Novo" }
{ "_id" : 3, "produto" : "NEOSALDINA", "tarja" : "Tarja Vermelha", "tipo" : "Novo", "preco" : 11.52 }
{ "_id" : ObjectId("5e98b2fcfd589f9f565694a9"), "produto" : "BENEGRIPI", "tarja" : "Tarja Vermelha", "tipo" : "Novo", "preco" : 9.36 }
{ "_id" : ObjectId("5e98b2fcfd589f9f565694aa"), "produto" : "NIMESULIDA", "tarja" : "Tarja Vermelha", "tipo" : "Genérico", "substancia" : "Nimesulida" }
{ "_id" : ObjectId("5e98b2fcfd589f9f565694ab"), "produto" : "NOEX", "tarja" : "Tarja Preta", "substancia" : "BUDESONIDA", "preco" : 33.93, "laboratorio" : "EUROFARMA LABORATÓRIOS S.A." }
```

O comando `find()` retorna os dados linha a linha, podemos melhorar a visualização dos registros com a chamada do método `pretty()`, conforme Figura abaixo.

```
> db.medicamentos.find().pretty()
{
  "_id" : ObjectId("5e98ab600193f51199bb8ce4"),
  "produto" : "DIPIRONA",
  "tarja" : "Venda Livre",
  "tipo" : "Genérico"
}
{
  "_id" : ObjectId("5e98abff0193f51199bb8ce5"),
  "produto" : "NOVALGINA",
  "tarja" : "Venda Livre",
  "tipo" : "Novo"
}
```

Existe também o `db.COLLECTION.findOne(query, projection)`, que retorna o primeiro registro da ordenação preestabelecida.

```
> db.medicamentos.findOne()
{
  "_id" : ObjectId("5e98ab600193f51199bb8ce4"),
  "produto" : "DIPIRONA",
  "tarja" : "Venda Livre",
  "tipo" : "Genérico"
}
```

Projeção:

A projeção (*projection*) no método `find()` significa selecionar apenas os campos necessários, em vez de selecionar todos os campos de um documento. Se um documento conter 7 campos e é necessário exibir apenas 3, selecione apenas 3 campos. Para carregar os campos solicitados, no segundo parâmetro do método `find` é necessário definir a lista de campos com o valor 1 ou 0, onde 1 é usado para mostrar o campo enquanto 0 é usado para ocultar os campos. Vejamos no exemplo da figura abaixo a busca por todos os valores sem carregar o campo `_id`.

```
> db.medicamentos.find({}, {_id: 0})
{ "produto" : "DIPIRONA", "tarja" : "Venda Livre", "tipo" : "Genérico" }
{ "produto" : "NOVALGINA", "tarja" : "Venda Livre", "tipo" : "Novo" }
{ "produto" : "NEOSALDINA", "tarja" : "Tarja Vermelha", "tipo" : "Novo", "preco" : 11.52 }
{ "produto" : "BENEGRIP", "tarja" : "Tarja Vermelha", "tipo" : "Novo", "preco" : 9.36 }
{ "produto" : "NIMESULIDA", "tarja" : "Tarja Vermelha", "tipo" : "Genérico", "substancia" : "Nimesulida" }
{ "produto" : "NOEX", "tarja" : "Tarja Preta", "substancia" : "BUDESONIDA", "preco" : 33.93, "laboratorio" : "EUROFARMA LABORATÓRIOS S.A." }
```

O campo `_id` sempre é retornado caso não seja especificado, vejamos um outro exemplo para carregar apenas o campo `produto` dos documentos da Coleção.

```
> db.medicamentos.find({}, {_id: 0, produto: 1})
{ "produto" : "DIPIRONA" }
{ "produto" : "NOVALGINA" }
{ "produto" : "NEOSALDINA" }
{ "produto" : "BENEGRIP" }
{ "produto" : "NIMESULIDA" }
{ "produto" : "NOEX" }
```

Limit (Top):

Para limitar os registros retornados do método `find()` pode-se utilizar método `limit()` que possui a sintaxe `db.COLLECTION.find().limit(NUMBER)`, o argumento numérico `NUMBER` define o número de documentos a ser exibido. A Figura abaixo apresenta o resultado desta operação.

```
> db.medicamentos.find({}, {_id: 0, produto: 1}).limit(3)
{ "produto" : "DIPIRONA" }
{ "produto" : "NOVALGINA" }
{ "produto" : "NEOSALDINA" }
```

O método `Limit()` retorna os registros de uma ordenação natural ou definida, entretanto, é possível pular um número de documentos a ser retornado. Para esse salto, pode-se utilizar o método `skip()`, que possui a sintaxe `db.COLLECTION.find().limit(NUMBER).skip(NUMBER)`. A figura abaixo apresenta a execução deste método:

```
> db.medicamentos.find({}, {_id: 0, produto: 1}).limit(3).skip(2)
{ "produto" : "NEOSALDINA" }
{ "produto" : "BENEGRIP" }
{ "produto" : "NIMESULIDA" }
```

Sorting (Ordenação):

O MongoDB retorna os documentos por meio de uma ordenação pré-existente. O método `sort()` permite definir a ordenação dos documentos retornados com um documento contendo uma lista de campos, junto com sua ordem de classificação no parâmetro do método. A ordem de classificação dos campos é definida pelos valores 1 e -1, onde 1 define em ordem crescente, enquanto -1 em ordem decrescente. A sintaxe do método `sort()` é definida por `db.COLLECTION.find().sort({CAMPO_1:1, CAMPO_2:-1})`. Na figura abaixo temos o retorno em ordem decrescente dos documentos pelo campo `produto`.

```
> db.medicamentos.find({}, {_id: 0, produto: 1}).sort({produto:-1})
{ "produto" : "NOVALGINA" }
{ "produto" : "NOEX" }
{ "produto" : "NIMESULIDA" }
{ "produto" : "NEOSALDINA" }
{ "produto" : "DIPIRONA" }
{ "produto" : "BENEGRIP" }
```

Operadores:

O parâmetro `query` do método `find()` permite a utilização dos operadores lógicos, operadores de comparação, entre outros. Para os operadores lógicos a tabela a seguir define a sintaxe para utilização de cada um deles

Operador	Descrição
\$and	Retorna todos os documentos que atendem todas as condições especificadas.
\$not	Inverte o critério especificado pela query retornando todos os valores que não atendem o critério da query.
\$nor	Retorna todos os documentos que não atendem os critérios especificados em ambos os casos.
\$or	Retorna todos os documentos que atendem uma das condições especificadas

- **\$and** - Para utilização do operador AND, a sintaxe na query necessita do operador \$and com um vetor dos campos a serem considerados, \$and[{campo1:valor},{campo2:valor},...], conforme apresentado na Figura abaixo onde é selecionado todos os documentos cujo campo produto seja NOVALGINA e o campo tarja seja Tarja Vermelha.

```
> db.medicamentos.find({$and:[{produto: "NOVALGINA"}, {tarja: "Tarja Vermelha"}]}).pretty()
{
  "_id" : ObjectId("5e98abff0193f51199bb8ce5"),
  "produto" : "NOVALGINA",
  "tarja" : "Tarja Vermelha",
  "tipo" : "Novo"
}
```

- **\$or** – Para utilização da condição OR, a sintaxe na query necessita do operador \$or com um vetor dos campos a serem considerados, \$or[{campo1:valor},{campo2:valor},...]. No exemplo da Figura abaixo é selecionado todos os documentos que tenham o campo produto com o valor NOVALGINA ou o campo tarja com o valor Tarja Preta.

```
> db.medicamentos.find({$or: [{produto: "NOVALGINA"}, {tarja: "Tarja Preta"}]}).pretty();
{
  "_id" : ObjectId("5e98abff0193f51199bb8ce5"),
  "produto" : "NOVALGINA",
  "tarja" : "Tarja Vermelha",
  "tipo" : "Novo"
}
{
  "_id" : ObjectId("5e98fdbd6412a457b703783c"),
  "produto" : "NOEX",
  "tarja" : "Tarja Preta",
  "preco" : 35.21
}
```

Os operadores de comparação são definidos na tabela abaixo.

Operador	Descrição
\$eq	Compara com os valores dos campos que possuem o valor especificado.
\$gt	Compara com os valores dos campos que possuem o valor maior que o especificado.
\$gte	Compara com os valores dos campos que possuem o valor maior ou igual ao especificado.
\$in	Compara com os valores dos campos que possuem um dos valores especificados dentro do vetor.
\$lt	Compara com os valores dos campos que possuem o valor menor que o especificado.
\$lte	Compara com os valores dos campos que possuem o valor menor ou igual ao especificado.
\$ne	Compara com os valores dos campos que não possui o valor especificado.
@nin	Compara com os valores dos campos que não possui um dos valores especificados dentro do vetor.

- **\$gt** – O operador \$gt Maior que (greater than), pode ser utilizado conforme exemplo da figura abaixo, onde foi consultado todos os documentos cujo campo `preco` é maior que 20.

```
> db.medicamentos.find({preco: {$gt:20}}).pretty()
{
  "_id" : ObjectId("5e98fdbd6412a457b703783c"),
  "produto" : "NOEX",
  "tarja" : "Tarja Preta",
  "preco" : 35.21
}
```

- **\$in** – O operador \$in é utilizado para buscar todos os documentos que possuem no campo especificado um dos valores que estão no vetor do \$in conforme exemplo da figura abaixo, onde foi consultado todos os documentos cujo campo `tarja` tenha um dos valores “Tarja Preta”, “Tarja Vermelha”.

```
> db.medicamentos.find({tarja: {$in:["Tarja Vermelha", "Tarja Preta"]}}).pretty()
{
  "_id" : ObjectId("5e98abff0193f51199bb8ce5"),
  "produto" : "NOVALGINA",
  "tarja" : "Tarja Vermelha",
  "tipo" : "Novo"
}
{
  "_id" : ObjectId("5e98fdbd6412a457b703783c"),
  "produto" : "NOEX",
  "tarja" : "Tarja Preta",
  "preco" : 35.21
}
```

1.6. Atualizar documentos (update)

O MongoDB possui os seguintes métodos para atualização dos dados de documentos:

- `db.COLLECTION.updateOne()` – Atualiza um único documento na coleção.
- `db.COLLECTION.updateMany()` – Atualiza múltiplos documentos na coleção.
- `db.COLLECTION.replaceOne()` – Insere um ou mais documentos na coleção.

Atualizar um único documento:

O método `updateOne()` atualiza o primeiro documento com o critério utilizado no parâmetro `<filter>`. O método possui a seguinte sintaxe: `db.collection.updateOne(<filter>, <update>, <options>)`, onde `<filter>` representa o critério para aplicar o update, semelhante às consultas realizadas no método `find()`, e `<update>` é um documento com os operadores de atualização `<options>` que representa algumas opções para a atualização.

Existem diversos operadores de atualização, a Tabela abaixo apresenta os principais operadores.

Operador	Descrição
<code>\$inc</code>	Utilizado para incrementar o valor de um campo com o valor do parâmetro.
<code>\$min</code>	Utilizado para atualizar os valores dos campos com o valor do parâmetro, cujo valor atual seja maior que o valor do parâmetro.
<code>\$max</code>	Utilizado para atualizar os valores dos campos com o valor do parâmetro, cujo valor atual seja menor que o valor do parâmetro.
<code>\$set</code>	Utilizado para atualizar o valor de um campo ou criar um novo campo.
<code>\$unset</code>	Utilizado para remover o(s) campo(s) dos documentos.
<code>\$mul</code>	Utilizado para multiplicar o valor de um campo com o valor do parâmetro.
<code>\$currentDate</code>	Atualiza o campo data com a data e hora atual.

Na figura abaixo foi realizada a atualização do campo `tarja` (`$set: {tarja: "Tarja Preta"}`), onde a `tarja` é "Tarja Vermelha". Como haviam vários documentos com o campo `tarja` com valor "Tarja Vermelha", o método atualizou a primeira ocorrência que trata-se do produto "NEOSALDINA".

```
> db.medicamentos.find({}, {_id:0})
{ "produto": "DIPIRONA", "tarja": "Venda Livre", "tipo": "Genérico" }
{ "produto": "NOVALGINA", "tarja": "Venda Livre", "tipo": "Novo" }
{ "produto": "NEOSALDINA", "tarja": "Tarja Vermelha", "tipo": "Novo", "preco": 11.52 }
{ "produto": "BENEGRIP", "tarja": "Tarja Vermelha", "tipo": "Novo", "preco": 9.36 }
{ "produto": "NIMESULIDA", "tarja": "Tarja Vermelha", "tipo": "Genérico", "substancia": "Nimesulida" }
{ "produto": "NOEX", "tarja": "Tarja Preta", "substancia": "BUDESONIDA", "preco": 35.21, "laboratorio": "EUROFARMA LABORATÓRIOS S.A." }
> db.medicamentos.updateOne({tarja:"Tarja Vermelha"},{$set: {tarja:"Tarja Preta"}})
{ "acknowledged": true, "matchedCount": 1, "modifiedCount": 1 }
> db.medicamentos.find({}, {_id:0})
{ "produto": "DIPIRONA", "tarja": "Venda Livre", "tipo": "Genérico" }
{ "produto": "NOVALGINA", "tarja": "Venda Livre", "tipo": "Novo" }
{ "produto": "NEOSALDINA", "tarja": "Tarja Preta", "tipo": "Novo", "preco": 11.52 }
{ "produto": "BENEGRIP", "tarja": "Tarja Vermelha", "tipo": "Novo", "preco": 9.36 }
{ "produto": "NIMESULIDA", "tarja": "Tarja Vermelha", "tipo": "Genérico", "substancia": "Nimesulida" }
{ "produto": "NOEX", "tarja": "Tarja Preta", "substancia": "BUDESONIDA", "preco": 35.21, "laboratorio": "EUROFARMA LABORATÓRIOS S.A." }
```

Atualizar vários documentos:

O método `updateMany()` possui os mesmos parâmetros do método `updateOne()`, porém seu comportamento atualiza todos os documentos do banco que possuem o critério do filtro. A Figura abaixo apresenta o resultado da execução deste método para atualizar todos os documentos cujo campo `tarja` tenha o valor Tarja Livre e seja alterado para Tarja Vermelha (`$set: {tarja: "Tarja Vermelha"}`).

```
> db.medicamentos.find({}, {_id:0})
{ "produto": "DIPIRONA", "tarja": "Venda Livre", "tipo": "Genérico" }
{ "produto": "NOVALGINA", "tarja": "Venda Livre", "tipo": "Novo" }
{ "produto": "NEOSALDINA", "tarja": "Tarja Preta", "tipo": "Novo", "preco": 11.52 }
{ "produto": "BENEGRIP", "tarja": "Tarja Vermelha", "tipo": "Novo", "preco": 9.36 }
{ "produto": "NIMESULIDA", "tarja": "Tarja Vermelha", "tipo": "Genérico", "substancia": "Nimesulida" }
{ "produto": "NOEX", "tarja": "Tarja Preta", "substancia": "BUDESONIDA", "preco": 35.21, "laboratorio": "EUROFARMA LABORATÓRIOS S.A." }
> db.medicamentos.updateMany({tarja:"Venda Livre"},{$set: {tarja:"Tarja Vermelha"}})
{ "acknowledged": true, "matchedCount": 2, "modifiedCount": 2 }
> db.medicamentos.find({}, {_id:0})
{ "produto": "DIPIRONA", "tarja": "Tarja Vermelha", "tipo": "Genérico" }
{ "produto": "NOVALGINA", "tarja": "Tarja Vermelha", "tipo": "Novo" }
{ "produto": "NEOSALDINA", "tarja": "Tarja Preta", "tipo": "Novo", "preco": 11.52 }
{ "produto": "BENEGRIP", "tarja": "Tarja Vermelha", "tipo": "Novo", "preco": 9.36 }
{ "produto": "NIMESULIDA", "tarja": "Tarja Vermelha", "tipo": "Genérico", "substancia": "Nimesulida" }
{ "produto": "NOEX", "tarja": "Tarja Preta", "substancia": "BUDESONIDA", "preco": 35.21, "laboratorio": "EUROFARMA LABORATÓRIOS S.A." }
```

Além da atualização dos campos de um documento, é possível substituir um documento por outro com exceção do campo `_id`. Essa substituição pode ser

realizada pelo método `replaceOne()`, que possui sintaxe semelhante à dos `updates` `db.COLLECTION.replaceOne(<filter>, <replacement>, <options>)`, a diferença é que `<replacement>` possui o documento a ser substituído. A figura abaixo apresenta a execução deste comando.

```
> db.medicamentos.find({}, { _id: 0 })
{ "produto": "DIPIRONA", "tarja": "Tarja Vermelha", "tipo": "Genérico" }
{ "produto": "NOVALGINA", "tarja": "Tarja Vermelha", "tipo": "Novo" }
{ "produto": "NEOSALDINA", "tarja": "Tarja Preta", "tipo": "Novo", "preco": 11.52 }
{ "produto": "BENEGRIP", "tarja": "Tarja Vermelha", "tipo": "Novo", "preco": 9.36 }
{ "produto": "NIMESULIDA", "tarja": "Tarja Vermelha", "tipo": "Genérico", "substancia": "Nimesulida" }
{ "produto": "NOEX", "tarja": "Tarja Preta", "substancia": "BUDESONIDA", "preco": 35.21, "laboratorio": "EUROFARMA LABORATÓRIOS S.A." }
> db.medicamentos.replaceOne({produto:"DIPIRONA"},{produto:"DIPIRONA SODICA", tarja: "Venda Livre", tipo: "Genérico", preco: 12.29})
{ "acknowledged": true, "matchedCount": 1, "modifiedCount": 1 }
> db.medicamentos.find({}, { _id: 0 })
{ "produto": "DIPIRONA SODICA", "tarja": "Venda Livre", "tipo": "Genérico", "preco": 12.29 }
{ "produto": "NOVALGINA", "tarja": "Tarja Vermelha", "tipo": "Novo" }
{ "produto": "NEOSALDINA", "tarja": "Tarja Preta", "tipo": "Novo", "preco": 11.52 }
{ "produto": "BENEGRIP", "tarja": "Tarja Vermelha", "tipo": "Novo", "preco": 9.36 }
{ "produto": "NIMESULIDA", "tarja": "Tarja Vermelha", "tipo": "Genérico", "substancia": "Nimesulida" }
{ "produto": "NOEX", "tarja": "Tarja Preta", "substancia": "BUDESONIDA", "preco": 35.21, "laboratorio": "EUROFARMA LABORATÓRIOS S.A." }
```

1.7. Exclusão de documentos (delete)

O MongoDB possui os seguintes métodos para exclusão de documentos:

- `db.COLLECTION.deleteOne(<filter>)` – Deleta um único documento na coleção.
- `db.COLLECTION.deleteMany(<filter>)` – Deleta múltiplos documentos na coleção.

Os métodos possuem o mesmo comportamento, porém o `deleteOne()` apaga a primeira ocorrência de documento que atende ao critério do filtro e `deleteMany()` apaga todos os documentos que atendem o critério do filtro. A figura abaixo apresenta a exclusão de um único documento, cujo campo `produto` seja Dipirona Sódica.

```
> db.medicamentos.find({}, { _id: 0 })
{ "produto" : "DIPIRONA SODICA", "tarja" : "Venda Livre", "tipo" : "Genérico", "preco" : 12.29 }
{ "produto" : "NOVALGINA", "tarja" : "Tarja Vermelha", "tipo" : "Novo" }
{ "produto" : "NEOSALDINA", "tarja" : "Tarja Preta", "tipo" : "Novo", "preco" : 11.52 }
{ "produto" : "BENEGRIP", "tarja" : "Tarja Vermelha", "tipo" : "Novo", "preco" : 9.36 }
{ "produto" : "NIMESULIDA", "tarja" : "Tarja Vermelha", "tipo" : "Genérico", "substancia" : "Nimesulida" }
{ "produto" : "NOEX", "tarja" : "Tarja Preta", "substancia" : "BUDESONIDA", "preco" : 35.21, "laboratorio" : "EUROFARMA LABORATÓRIOS S.A." }
> db.medicamentos.deleteOne({produto:"DIPIRONA SODICA"})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.medicamentos.find({}, { _id: 0 })
{ "produto" : "NOVALGINA", "tarja" : "Tarja Vermelha", "tipo" : "Novo" }
{ "produto" : "NEOSALDINA", "tarja" : "Tarja Preta", "tipo" : "Novo", "preco" : 11.52 }
{ "produto" : "BENEGRIP", "tarja" : "Tarja Vermelha", "tipo" : "Novo", "preco" : 9.36 }
{ "produto" : "NIMESULIDA", "tarja" : "Tarja Vermelha", "tipo" : "Genérico", "substancia" : "Nimesulida" }
{ "produto" : "NOEX", "tarja" : "Tarja Preta", "substancia" : "BUDESONIDA", "preco" : 35.21, "laboratorio" : "EUROFARMA LABORATÓRIOS S.A." }
```

A figura abaixo apresenta a exclusão de todos os documentos cujo campo tarja seja Tarja Preta.

```
> db.medicamentos.find({}, { _id: 0 })
{ "produto" : "NOVALGINA", "tarja" : "Tarja Vermelha", "tipo" : "Novo" }
{ "produto" : "NEOSALDINA", "tarja" : "Tarja Preta", "tipo" : "Novo", "preco" : 11.52 }
{ "produto" : "BENEGRIP", "tarja" : "Tarja Vermelha", "tipo" : "Novo", "preco" : 9.36 }
{ "produto" : "NIMESULIDA", "tarja" : "Tarja Vermelha", "tipo" : "Genérico", "substancia" : "Nimesulida" }
{ "produto" : "NOEX", "tarja" : "Tarja Preta", "substancia" : "BUDESONIDA", "preco" : 35.21, "laboratorio" : "EUROFARMA LABORATÓRIOS S.A." }
> db.medicamentos.deleteMany({tarja:"Tarja Preta"})
{ "acknowledged" : true, "deletedCount" : 2 }
> db.medicamentos.find({}, { _id: 0 })
{ "produto" : "NOVALGINA", "tarja" : "Tarja Vermelha", "tipo" : "Novo" }
{ "produto" : "BENEGRIP", "tarja" : "Tarja Vermelha", "tipo" : "Novo", "preco" : 9.36 }
{ "produto" : "NIMESULIDA", "tarja" : "Tarja Vermelha", "tipo" : "Genérico", "substancia" : "Nimesulida" }
```

1.8. Comandos em massa (BulkWrite)

O MongoDB possui um comando no qual é possível executar comandos em massa na mesma coleção. Os comandos são executados por padrão em ordem, porém é possível alterar via parâmetro para que o MongoDB execute paralelamente as operações. O método possui a sintaxe `db.COLLECTION.bulkWrite([<operacao 1>, <operacao 2>, ...], options)`, onde no primeiro parâmetro são definidas as operações a serem aplicadas (`insertOne()`, `updateOne()`, `updateMany()`, `deleteOne()`, `deleteMany()` e `replaceOne()`) e no segundo define-se os comandos que devem ser executados paralelamente, pois por default é executado em ordem. A figura abaixo apresenta os documentos da coleção, a Figura 6 apresenta o comando `bulkWrite` e a Figura 7 apresenta o resultado da execução na coleção.

```
> db.medicamentos.find({}, {_id:0})
{ "produto" : "NOVALGINA", "tarja" : "Tarja Vermelha", "tipo" : "Novo" }
{ "produto" : "BENEGRIP", "tarja" : "Tarja Vermelha", "tipo" : "Novo", "preco" : 9.36 }
{ "produto" : "NIMESULIDA", "tarja" : "Tarja Vermelha", "tipo" : "Genérico", "substancia" : "Nimesulida" }
```

Figura 6 – Exemplo do comando bulkWrite.

```
db.medicamentos.bulkWrite(
[
  { insertOne :
    {
      "document" :
      {
        "produto" : "NOEX", "tarja" : "Traja Preta", "preco" : 35.21
      }
    }
  },
  { insertOne :
    {
      "document" :
      {
        "produto" : "NEOSALDINA", "tarja" : "Venda Livre"
      }
    }
  },
  { updateOne :
    {
      "filter" : { "produto" : "NEOSALDINA" },
      "update" : { $set : { "preco" : 11.23 } }
    }
  },
  { deleteOne :
    { "filter" : { "produto" : "BENEGRIP" } }
  },
  { replaceOne :
    {
      "filter" : { "produto" : "NIMESULIDA" },
      "replacement" : { "produto" : "NALDECON", "tarja" : "Venda Livre", "preco" : 9.89 }
    }
  }
]
);
```

Figura 7 – Resultado do bulkWrite.

```
> db.medicamentos.find({}, {_id:0})
{ "produto" : "NOVALGINA", "tarja" : "Tarja Vermelha", "tipo" : "Novo" }
{ "produto" : "NALDECON", "tarja" : "Venda Livre", "preco" : 9.89 }
{ "produto" : "NOEX", "tarja" : "Traja Preta", "preco" : 35.21 }
{ "produto" : "NEOSALDINA", "tarja" : "Venda Livre", "preco" : 11.23 }
```

1.9. Índices

Os índices são estruturas de dados especiais que armazenam o valor de um campo específico ou conjunto de campos, ordenado pelo valor dos campos nos quais o índice foi criado. A criação de índices pode melhorar a velocidade das operações de pesquisa no MongoDB porque, em vez de pesquisar em todo o documento, a

pesquisa é realizada na estrutura dos índices, que por sua vez contêm apenas alguns campos. Por outro lado, ter muitos índices pode prejudicar o desempenho das operações de inserção, atualização e exclusão, devido à gravação adicional e ao espaço de dados adicional utilizado pelos índices.

Criação de índices:

A criação de índices tem a seguinte sintaxe `db.COLLECTION_NAME.createIndex({CAMPO:1})`, onde deve ser definido o campo do documento e o tipo de índice: ascendente (1) ou descendente(-1).

```
> db.medicamentos.createIndex({produto:1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

Sucesso

Consultar índices:

Para consultar os índices existentes de uma coleção, o método utilizado é `getIndexes()`, cuja sintaxe é `db.COLLECTION_NAME.getIndexes()`. Conforme podemos observar, a coleção já possui um índice pré-criado para o campo `_id`.

```
> db.medicamentos.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "IGTI_DATABASE.medicamentos"
  },
  {
    "v" : 2,
    "key" : {
      "produto" : 1
    },
    "name" : "produto_1",
    "ns" : "IGTI_DATABASE.medicamentos"
  }
]
```

Índice ascendente para o campo _id

Índice ascendente para o campo produto

Exclusão de índices:

A exclusão de índices utiliza-se do método `db.collection.dropIndex()`, onde o parâmetro recebido pode ser o nome do índice

`db.collection.dropIndex("produto_1")` ou o campo do índice
`db.collection.dropIndex({"produto": 1}).`

```
> db.medicamentos.dropIndex({"produto": 1})
{ "nIndexesWas" : 2, "ok" : 1 }
> db.medicamentos.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "IGTI_DATABASE.medicamentos"
  }
]
```

A partir da versão 4.2 é possível excluir todos os índices (exceto o `_id`), com o método: `db.collection.dropIndex("*").`

Índices textuais (text indexing):

Os índices textuais são um tipo especial de índice que permite que o MongoDB faça buscas de texto no conteúdo da string dos documentos.

Suponhamos a coleção de livros criada na base de dados, conforme Figura abaixo:

```
db.livros.insertMany([
  {"_id": 1, "titulo": "Bootcamp IGTI Full Stack"},
  {"_id": 2, "titulo": "MongoDB no Bootcamp"},
  {"_id": 3, "titulo": "Introducao ao MongoDB no Bootcamp do IGTI"},
  {"_id": 4, "titulo": "Criacao de Indexes no MongoDB"}])
```

A criação de um índice textual tem a seguinte sintaxe:
`db.COLLECTION.createIndex({campo1: "text", campo2: "text",...}).`
 Para o exemplo da figura acima, podemos criar um índice conforme a imagem a seguir:

```
> db.livros.createIndex({titulo: "text"})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```


Com o índice criado podemos realizar buscar textuais nos campos dos documentos conforme exemplo da figura abaixo, na qual foi realizada a busca com as palavras definidas dentro da operação `$search`, no índice textual que fora criado.

```
> db.livros.find({$text: {$search: "Bootcamp IGTI MongoDB"}})
{ "_id" : 2, "titulo" : "MongoDB no Bootcamp" }
{ "_id" : 1, "titulo" : "Bootcamp IGTI Full Stack" }
{ "_id" : 3, "titulo" : "Introducao ao MongoDB no Bootcamp do IGTI" }
{ "_id" : 4, "titulo" : "Criacao de Indexes no MongoDB" }
```

A busca textual possui um score de match das palavras nos campos indexados, ou seja, é possível obter estatisticamente o grau de proximidade da busca com os campos. Para visualizar esta informação, utiliza-se o operador `$meta`: "textScore" no `find()`. Por fim, podemos ranquear por grau de proximidade os documentos mais relevantes utilizando-se o `sort()` pelo score.

```
> db.livros.find({$text: {$search: "Bootcamp IGTI MongoDB"}}, {score: { $meta: "textScore"}}).sort({score:{$meta: "textScore"}})
{ "_id" : 3, "titulo" : "Introducao ao MongoDB no Bootcamp do IGTI", "score" : 1.7999999999999998 }
{ "_id" : 2, "titulo" : "MongoDB no Bootcamp", "score" : 1.5 }
{ "_id" : 1, "titulo" : "Bootcamp IGTI Full Stack", "score" : 1.25 }
{ "_id" : 4, "titulo" : "Criacao de Indexes no MongoDB", "score" : 0.625 }
```

1.10. Modelagem

A modelagem de dados no MongoDB é completamente flexível, diferentemente dos bancos SQL onde é necessário definir a estrutura da tabela para possibilitar a inserção dos dados, no MongoDB essa estrutura não é definida *a priori*. Apesar desta flexibilidade, no MongoDB temos dois tipos de estrutura para os documentos:

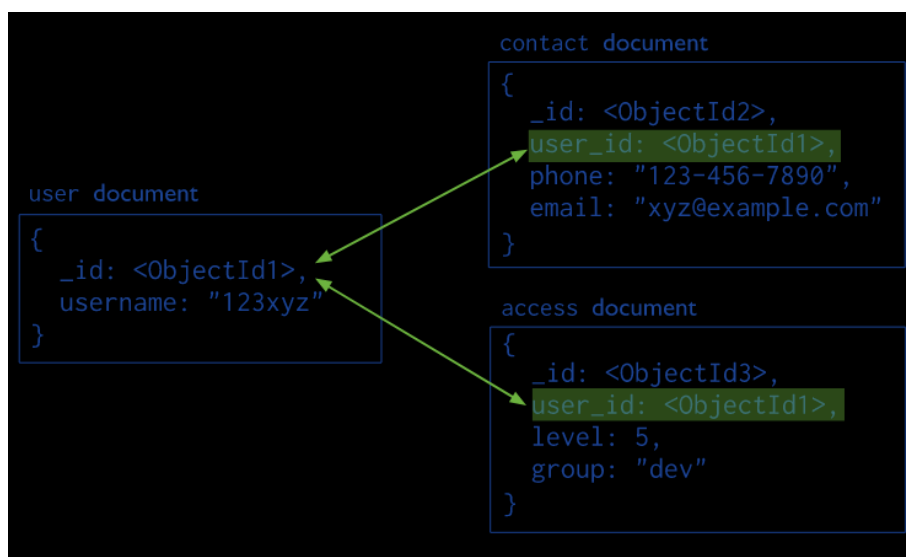
Modelos de dados incorporados (embedded data):

Neste tipo de modelo, também conhecido como **modelo de dados desnormalizados**, os dados estão todos na mesma estrutura do documento. Desta forma, um documento possui campo ou um vetor de dados. Esses modelos de dados possibilitam que as aplicações consultem e atualizem os dados relacionados numa única operação. A figura abaixo apresenta a estrutura de um documento com estruturas incorporadas.



Modelo de dados referenciados:

Neste tipo de modelagem, conhecida como **modelo de dados normalizados**, a estrutura do documento segue o princípio dos bancos relacionais no qual os dados são referenciados por chave primária/estrangeira. Na figura abaixo, temos uma referência do usuário em dois documentos diferentes do que foi estruturado no exemplo apresentado anteriormente na figura acima.



1.11. Agregações

Agregações é o processo que retorna os resultados computados de acordo com a operação. As operações de agregação podem agrupar valores de vários documentos e executar várias operações nos dados agrupados para retornar um

único resultado. A operação de group by do SQL é equivalente à uma agregação do MongoDB. O método `db.COLLECTION.aggregate()` realiza essas agregações e possui a sintaxe conforme apresentada na figura abaixo:

```
db.COLLECTION.aggregate([
  { $project: {} },
  { $match: { campo1: "X" } },
  { $group: { _id: "$campo2", total: { $sum: "$campo3" } } },
  { $sort: {} },
  { $skip: numero_inteiro },
  { $limit: numero_inteiro }
])
```

Este método possui algumas operações de agregação e as principais foram apresentadas na figura acima, sendo elas:

- `$project`: utilizada para definir os campos que serão carregados do documento.
- `$match`: utilizada para filtrar os documentos que serão utilizados na agregação.
- `$group`: utilizada para realizar a agregação com operações de agregação que serão vistas mais à frente (sum, abs, avg,...).
- `$sort`: utilizada para realizar a ordenação do resultado apresentado.
- `$skip`: utilizada para pular os documentos que não serão agregados.
- `$limit`: utilizada para limitar o número de documentos que serão agregados.

Na figura abaixo temos o exemplo da obtenção da média de preços dos produtos pelo campo `tarja`.

```
> db.medicamentos.aggregate([{$group: { _id: "$tarja", media: {$avg: "$preco"} } }])
{ "_id" : "Venda Livre", "media" : 10.56 }
{ "_id" : "Tarja Preta", "media" : 35.21 }
{ "_id" : "Tarja Vermelha", "media" : 8.45 }
```

A consulta em SQL correspondente seria:

```
SELECT tarja, AVG(preco) FROM medicamentos GROUP BY tarja
```

No MongoDB temos os métodos de agregação singulares, nos quais uma operação é realizada na coleção com uma determinada `query`.

Count:

O MongoDB possui o método `db.COLLECTION.count(<query>)`, que realiza a contabilização do número de documentos de uma coleção, que atendem um critério definido na query. A partir da versão 4.0.3 foi disponibilizado o método `db.COLLECTION.countDocuments(<query>)`, pois o método `count()` retorna o valor com base no metadados da coleção, ou seja, é um valor aproximado. Portanto, o método `countDocuments()` foi disponibilizado para possibilitar uma contagem de documentos mais acurada. A figura abaixo apresenta o conjunto de dados da coleção e a utilização do método `countDocuments()` para contabilizar o número de documentos que atendam o critério da query (`tarja: "Venda Livre"`).

```
> db.medicamentos.find()
{ "_id" : ObjectId("5e98abff0193f51199bb8ce5"), "produto" : "NOVALGINA", "tarja" : "Tarja Vermelha", "tipo" : "Novo" }
{ "_id" : ObjectId("5e98b2fcfd589f9f565694aa"), "produto" : "NALDECON", "tarja" : "Venda Livre", "preco" : 9.89 }
{ "_id" : ObjectId("5e98fdbd6412a457b703783c"), "produto" : "NOEX", "tarja" : "Tarja Preta", "preco" : 35.21 }
{ "_id" : ObjectId("5e98fdbd6412a457b703783d"), "produto" : "NEOSALDINA", "tarja" : "Venda Livre", "preco" : 11.23 }
> db.medicamentos.countDocuments({tarja:"Venda Livre"})
2
```

Distinct:

O método `db.COLLECTION.distinct(field,query)` é utilizado para identificar valores distintos na coleção. A figura abaixo apresenta uma base com vários documentos com o campo `produto` repetido. Para uma extração da lista distinta dos valores deste campo, foi utilizado o método `distinct()`.

```
> db.medicamentos.find()
{ "_id" : ObjectId("5e98abff0193f51199bb8ce5"), "produto" : "NOVALGINA", "tarja" : "Tarja Vermelha", "tipo" : "Novo" }
{ "_id" : ObjectId("5e98b2fcfd589f9f565694aa"), "produto" : "NALDECON", "tarja" : "Venda Livre", "preco" : 9.89 }
{ "_id" : ObjectId("5e98fdbd6412a457b703783c"), "produto" : "NOEX", "tarja" : "Tarja Preta", "preco" : 35.21 }
{ "_id" : ObjectId("5e98fdbd6412a457b703783d"), "produto" : "NEOSALDINA", "tarja" : "Venda Livre", "preco" : 11.23 }
{ "_id" : ObjectId("5e9f129a5c6f332f5d9ad3fd"), "produto" : "BENEGRIPI", "tarja" : "Tarja Vermelha", "tipo" : "Novo", "preco" : 9.36 }
{ "_id" : ObjectId("5e9f129a5c6f332f5d9ad3fe"), "produto" : "NIMESULIDA", "tarja" : "Tarja Vermelha", "tipo" : "Genérico", "substancia" : "Nimesulida" }
{ "_id" : ObjectId("5e9f130d5c6f332f5d9ad3ff"), "produto" : "NOEX", "tarja" : "Tarja Vermelho", "preco" : 7.54 }
{ "_id" : ObjectId("5e9f130d5c6f332f5d9ad400"), "produto" : "NIMESULIDA", "tarja" : "Tarja Preta", "substancia" : "Nimesulida" }
> db.medicamentos.distinct("produto")
[
  "BENEGRIPI",
  "NALDECON",
  "NEOSALDINA",
  "NIMESULIDA",
  "NOEX",
  "NOVALGINA"
]
```

Capítulo 2. MongoDB Atlas

2.1. O serviço MongoDB Atlas

O MongoDB Atlas é um *Database as a Service* (DaaS), anteriormente conhecido como MongoLab ou mLab, que provê um serviço de banco de dados na nuvem totalmente gerenciado, com provisionamento e dimensionamento automatizados de bancos de dados MongoDB, backup e recuperação, monitoramento e alerta 24/7, ferramentas de gerenciamento baseadas na web e suporte especializado. A plataforma de banco de dados como serviço do MongoDB Atlas está em diversos provedores, como AWS, Azure e Google, e permite que os desenvolvedores dediquem especialmente no desenvolvimento de produtos, e não nas operações (mLab, 2020).

Atualmente existem três tipos de planos de serviços oferecidos pela plataforma: **Sandbox** que é gratuito e oferece 0.5Gb livre, idealmente utilizado para prototipações, aprendizados e pequenos desenvolvimentos; e os serviços pagos **Shared** (limitado a 8Gb) e **Dedicated**.

2.1. Utilização do serviço

Para a utilização do serviço é necessária a criação de uma conta na plataforma <https://www.mongodb.com/cloud/atlas>, conforme os passos a seguir:

1. O primeiro passo é a criação da conta no MongoDB Atlas com o preenchimento do formulário conforme a figura abaixo:



Try MongoDB Atlas

Used by millions of developers around the world.

8 characters minimum

☐ I agree to the [terms of service](#) and [privacy policy](#).

[Get Started with 512 MB Free](#)

Com a criação da conta e após acessar o MongoDB Atlas, o próximo passo será a definição do tipo plano a ser utilizado, conforme Figura abaixo. Para este curso iremos seguir com o serviço do Sandbox gratuito que ainda não é compatível com a nova versão do MongoDB (4.0 e 4.2), mas é suficiente para o aprendizado. Caso seja necessário é possível realizar o upgrade.

Shared Clusters	Dedicated Clusters	Dedicated Multi-Region Clusters
For teams learning MongoDB or developing small applications.	For teams building applications that need advanced development and production-ready environments.	For teams developing world-class applications that require multi-region resiliency or ultra-low latency.
<ul style="list-style-type: none"> Highly available auto-healing cluster End-to-end encryption Role-based access control 	<ul style="list-style-type: none"> Includes all features from Shared Clusters Auto-scaling Network isolation Realtime performance metrics 	<ul style="list-style-type: none"> Includes all features from Shared and Dedicated Clusters Replicate data across multiple regions Globally distributed read and write operations Control data residency at the document level
Create a cluster	Create a cluster	Create a cluster
Starting at FREE	Starting at \$0.08/hr* <small>*estimated cost \$58.94/month</small>	Starting at \$0.13/hr* <small>*estimated cost \$98.55/month</small>

Após a definição do plano, selecionamos uma das opções de provedores disponíveis, neste caso selecionamos a AWS e a região de Virginia (us-east-1) e definimos o nome do nosso cluster dentro do provedor, por fim selecione Create Cluster conforme a figura abaixo.

Cloud Provider & Region
AWS, N. Virginia (us-east-1) >

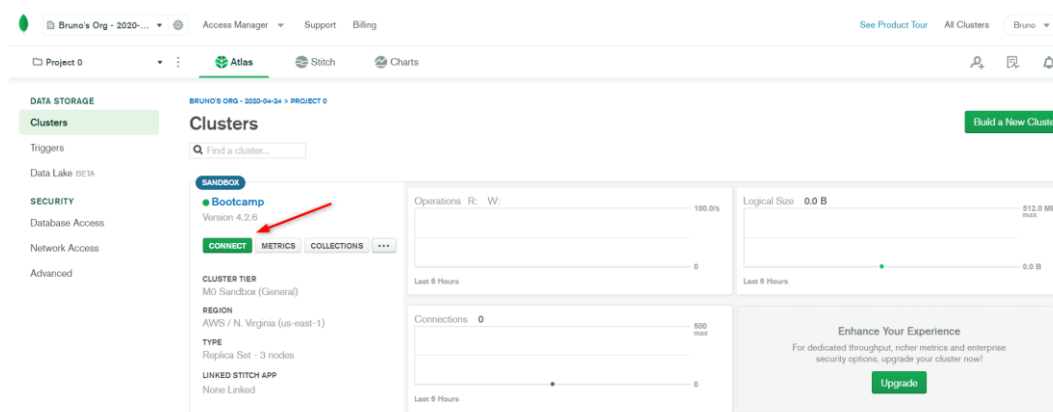
Cluster Tier
M0 Sandbox (Shared RAM, 512 MB Storage) Encrypted >

Additional Settings
MongoDB 4.2, No Backup >

Cluster Name
Bootcamp >

FREE
Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.
Back
Create Cluster

A criação do cluster demora uns minutos para o acesso e criação da base de dados. Após o painel disponibilizar a sandbox com o cluster criado, o próximo passo será a criação do banco de dados, conforme a figura abaixo.



Ao selecionar a conexão ao cluster, o primeiro acesso solicita a criação do banco de dados MongoDB, a figura abaixo. Neste passo você define as políticas de conexão ao servidor de banco, o usuário e senha para acesso ao banco.

Connect to Bootcamp

Setup connection security > Choose a connection method > Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more](#)

You can't connect yet. Set up your firewall access and user security permission below.

1 Whitelist a connection IP address

Add Your Current IP Address Add a Different IP Address

2 Create a MongoDB User

This first user will have [atlasAdmin](#) permissions for this project.

Keep your credentials handy, you'll need them for the next step.

Username Password Autogenerate Secure Password
igtiuser resultgi HIDE
Create MongoDB User

Com o banco criado, a conexão ao mesmo se dá por três opções, via mongo shell, pelo MongoDB Compass ou pelos drivers de conexão para a aplicação.

Connect to Bootcamp

✓ Setup connection security > Choose a connection method > Connect

Choose a connection method [View documentation](#)

Get your pre-formatted connection string by selecting your tool below.

Connect with the mongo shell
Interact with your cluster using MongoDB's interactive Javascript interface

Connect your application
Connect your application to your cluster using MongoDB's native drivers

Connect using MongoDB Compass
Explore, modify, and visualize your data with MongoDB's GUI

Go Back

Close

Para integrar a conexão de uma aplicação com a API do MongoDB Atlas, utilizaremos como exemplo a conexão de uma aplicação Node.js, embora a plataforma ofereça drivers de conexão com outras linguagens de programação.

2.1. Integração ao MongoDB

Na sessão anterior foram apresentadas diversas formas de integrar uma aplicação ao MongoDB. Uma das integrações mais populares é a biblioteca Mongoose com o Nodejs, esta biblioteca está presente em várias aplicações que utilizam o MongoDB.

O Mongoose é um framework ODM (*Object Document Mapper*) que realiza o mapeamento entre o modelo de documentos do MongoDB para o modelo orientado à objetos, semelhante aos frameworks ORM para os bancos relacionais. Desta forma, o Mongoose permite a definição de objetos com schema fortemente tipado, que mapeia aos documentos no MongoDB, permitindo, assim, o modelamento de sua aplicação.

Embora o MongoDB tenha uma flexibilidade para os schemas do banco, com essa biblioteca esta característica se limita de acordo com os modelos criados no desenvolvimento da aplicação. Essa biblioteca possui um sistema de conversão de tipos, validação, criação de consultas e hooks para lógica de negócios. Atualmente o Mongoose possui oito tipos para definição na criação dos schemas: String, Number, Date, Buffer, Boolean, Mixed, ObjectId e Array.

Em cada tipo de dados é possível especificar nos schemas:

- Um valor padrão.
- Uma função de validação customizada.
- Obrigatoriedade do campo.
- Uma função para manipular os dados antes de retornar um objeto.
- Uma função para manipular os dados antes de persistir no Mongo.
- Criação de índices.

Para alguns tipos de dados existem algumas opções adicionais, como por exemplo o tipo String possuir as seguintes opções:

- Converter para maiúsculo.
- Converter para minúsculo.
- Reduzir espaços extras (trim).
- Definir expressão regular para validar os dados.
- Enumeração para definir strings válidas.

Instalação:

O processo de instalação é pelo npm, assim, execute o comando abaixo no seu projeto:

```
npm install mongoose
```

Importação:

A biblioteca pode ser importada de duas formas, utilizando o require ou o imports do ES6.

```
const mongoose = require('mongoose');  
import mongoose from 'mongoose';
```

Conexão:

A conexão faz a requisição e retorna uma promise, onde é possível realizar o tratamento do erro de conexão ou notificar em caso de sucesso.

```
await mongoose.connect('mongodb://localhost/IGTI_DATABASE', {  
  useNewUrlParser: true,  
  useUnifiedTopology: true  
});
```

Schema:

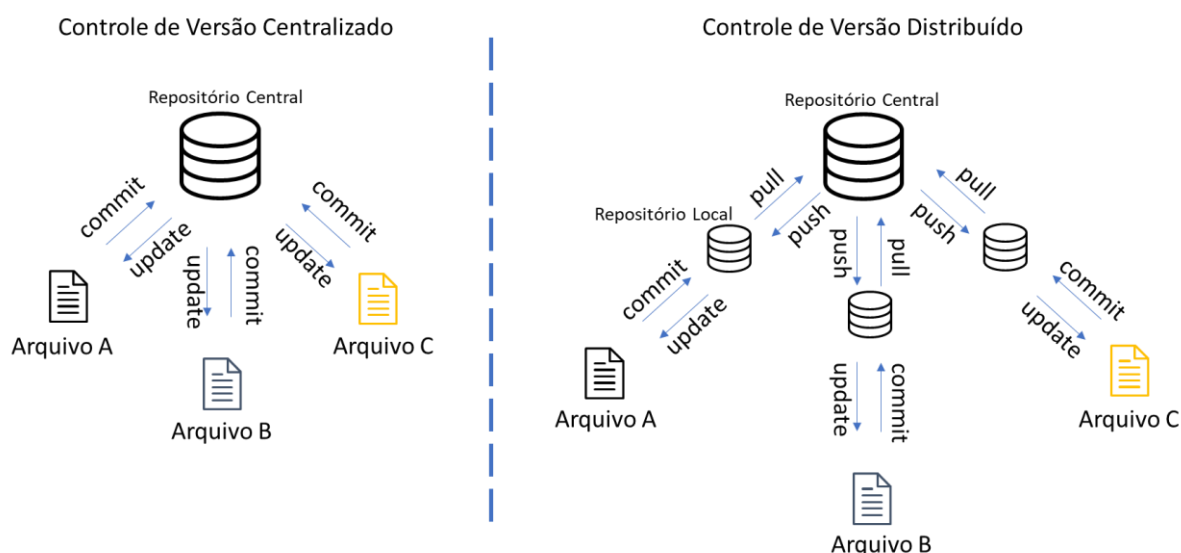
Um schema pode ser definido da seguinte forma:

```
const Medicamento = new Schema({  
  produto: String,  
  tarja: String,  
  preco: Number,  
  tipo: Date  
});
```

Capítulo 3. Git e GitHub

Sistemas de controle de versão (VCS) é uma categoria de ferramentas que tem por finalidade o gerenciamento de diferentes versões de um ou mais documentos num projeto. Esses sistemas possuem diversos benefícios e se dividem em duas categorias: **centralizados** e **distribuídos**. Os sistemas centralizados (CVCS) utilizam um único repositório central, que armazena todos os arquivos no qual a equipe possa trabalhar de forma colaborativa. Enquanto que nos sistemas distribuídos, além do repositório central, cada membro da equipe possui um repositório local onde suas alterações são armazenadas e posteriormente compartilhadas com o restante da equipe, conforme a Figura 8.

Figura 8 – Comparação Controle de Versão Centralizado e Distribuído.



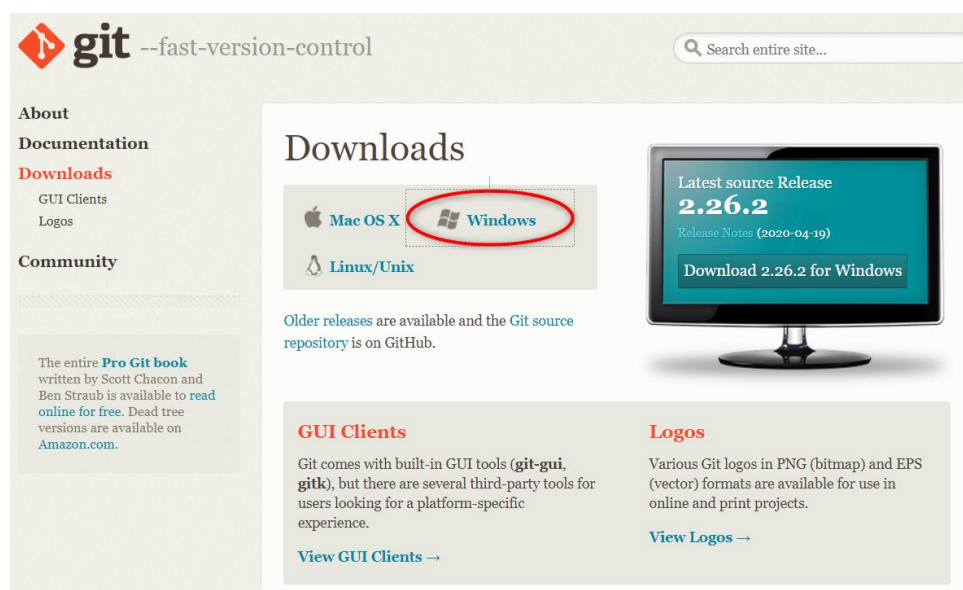
3.1. Introdução ao Git

O Git é um sistema de controle de versão de código aberto, com manutenção ativa que foi desenvolvido em 2005 por Linus Torvalds, a mesma pessoa que criou o sistema operacional Linux. O Git é um sistema distribuído no qual permite o desenvolvimento de um projeto, onde várias pessoas podem contribuir simultaneamente no seus arquivos, seja em edição, criação ou até mesmo exclusão.

3.2. Instalação e configuração do Git

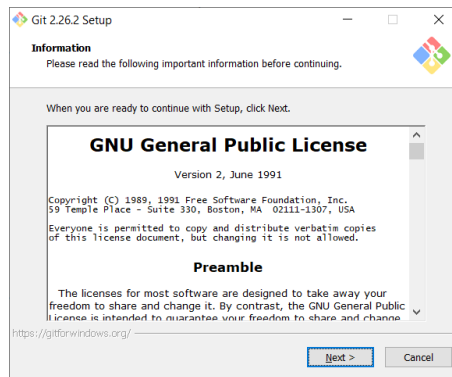
A instalação e configuração do Git é simples, mas é recomendado sempre consultar a página oficial com as últimas atualizações. Neste curso será apresentado o processo de instalação do git para Windows, porém a ferramenta está disponível para instalação em outras plataformas.

O primeiro passo para instalação é realizar o download do instalador mais recente para Windows (<https://git-scm.com/downloads>), conforme a figura abaixo.

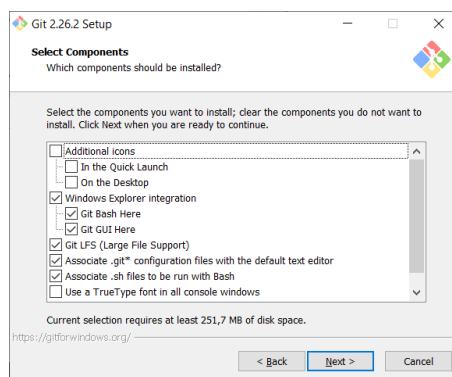


O processo de instalação deve ser iniciado por meio do instalador do download realizado e, a partir disto, a tela do assistente de instalação do Git será exibida. Seguiremos as opções padrão neste momento, pois é a mais recomendada para a maioria dos usuários, portanto, siga as instruções Avançar [Next] e Concluir [Finish] para concluir a instalação, conforme as figuras abaixo:

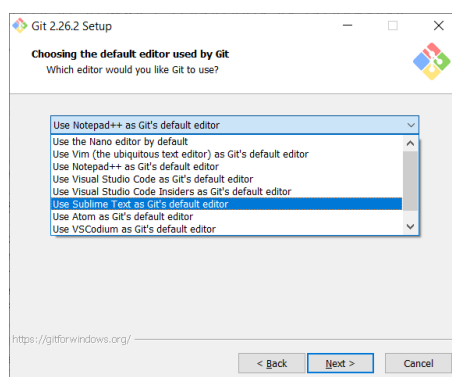
- **Passo 1:** Termo de licença do Git.



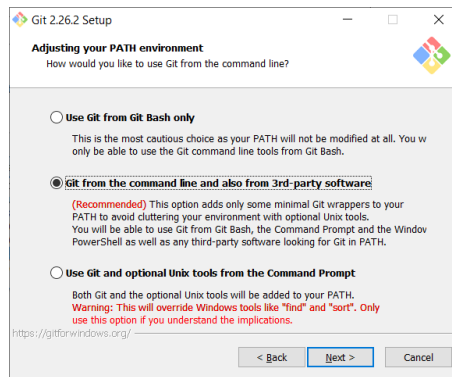
- **Passo 2:** utilizaremos as configurações padrões, instalando inclusive o Git Bash para execução de comandos.



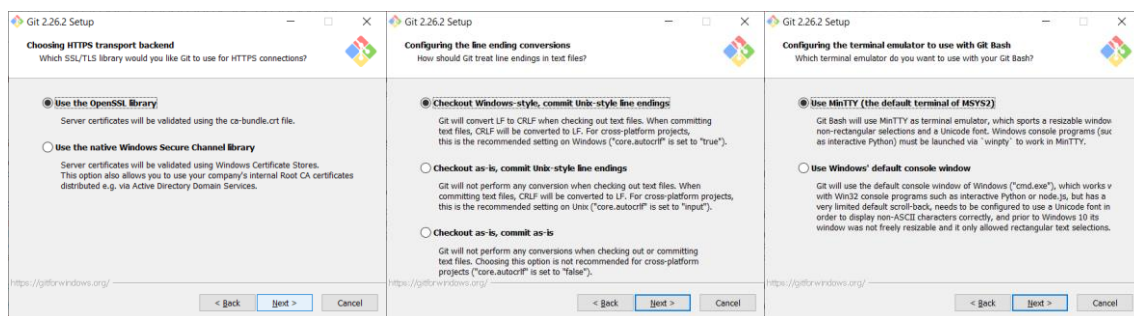
- **Passo 3:** esta define o editor de texto padrão a ser vinculado ao Git, fica a critério de cada um a escolha do editor preferido. Utilizaremos como exemplo o Notepad++.



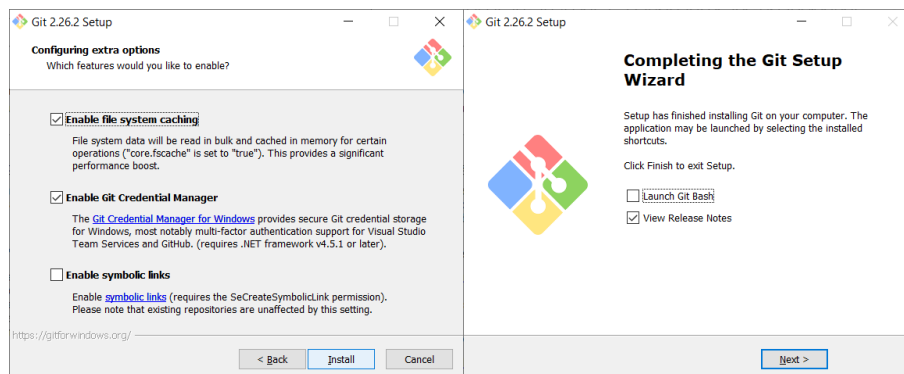
- **Passo 4:** defina se a execução de comandos poderá ser realizada além do Git Bash, no prompt comando e outras softwares. Automaticamente o Git será incluído no PATH das variáveis de ambiente do Windows.



- **Passo 5:** seguiremos com o padrão para o protocolo de segurança, estilo de versionamento e emulador do Git Bash.

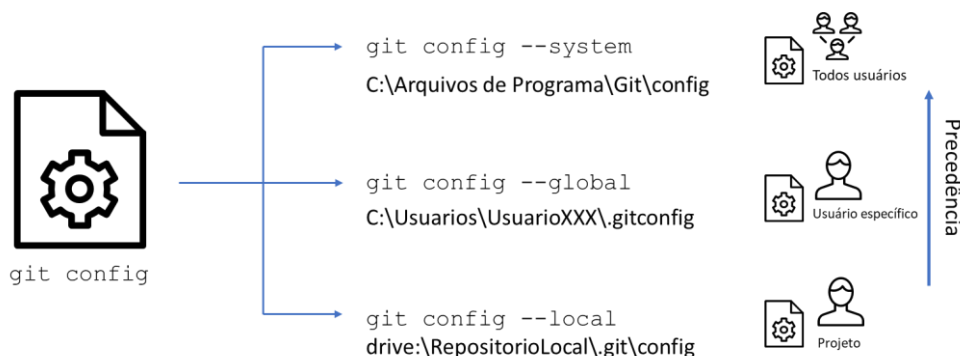


- **Passo 6:** as features adicionais serão adicionadas no processo de instalação, sobretudo o Git Credential Manager.



Após a instalação é necessário realizar algumas configurações no Git relacionadas à identidade, editor de texto padrão etc. Por meio do comando `git config` é possível visualizar e definir as variáveis de configuração em um dos três tipos de arquivo de configuração existente: sistema, usuário e projeto. Esses arquivos

de configuração são armazenados em lugares diferentes, sendo necessário definir qual o escopo da configuração a ser realizada, conforme a figura abaixo.



Cada nível sobreescreve os valores do nível anterior, ou seja, valores em `.git/config` prevalecem sobre `usuarioXXX \.gitconfig`. As localização dos arquivos de configuração podem ser visualizadas com o seguinte comando: `git config --list --s:how-origin`

Configurar identidade:

A configuração da identidade é o primeiro passo após a instalação do Git, onde são definidos o nome de usuário e endereço de e-mail. Essa configuração é importante, pois todos os commits no Git utilizam essas informações. Os comandos para definir essas informações, são:

```
git config --global user.name "Seu nome para exibição"
```

```
git config --global user.email "seu-email@email.com"
```

Visualizar as configurações:

Podemos visualizar as configurações existentes com o comando:

```
git config --list
```

Configurar editor de texto:

O Git permite associar um editor de texto de preferência do usuário às suas configurações, por padrão o Git utiliza o **vim**, entretanto, no nosso processo de

instalação definimos o Notepad++. Caso seja necessária a modificação do editor de texto padrão, podemos utilizar o comando abaixo, onde em EDITOR é definido qual será o editor padrão (SublimeText, Notepad++, Vim, ...).

```
git config --global core.editor EDITOR
```

Colorir as linhas:

Podemos melhorar a visualização do git no terminal com linhas de diferentes cores, com o comando abaixo:

```
git config --global color.ui auto
```

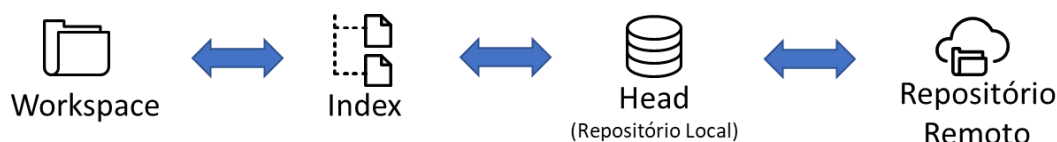
```
git config --global color.diff auto
```

```
git config --global color.status auto
```

```
git config --global color.branch auto
```

3.3. Principais comandos do Git

Nessa seção vamos passar pelos principais comandos do Git, que podem ser utilizados tanto no Git Bash quanto no prompt comando. Antes de iniciarmos com os principais comandos, é importante destacar a organização do Git. Podemos dizer que o Git se organiza em quatro estruturas de árvores, a primeira é o diretório de trabalho (**Workspace**), que contém os arquivos; a segunda é a **Index**, considerada como uma área temporária; a terceira é a **HEAD**, que aponta para o último commit realizado; e a última é o **Repositório Remoto**, conforme a figura abaixo:



- `git init`: este comando cria um novo repositório local (**Head**) e inicia a rastreabilidade dos arquivos no diretório. Este comando adiciona uma pasta

oculta que armazena a estrutura dos dados necessários para o controle de versão do Git. No exemplo da figura abaixo, foi iniciado o repositório na pasta C:\IGTI\Projetos\.

```
MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos
$ git init
Initialized empty Git repository in C:/IGTI/Projetos/.git/
```

Podemos observar na figura abaixo que imediatamente foi criada a pasta oculta com a estrutura de dados deste novo repositório.

C:\IGTI\Projetos			
Name	Date modified	Type	Size
.git	01/05/2020 15:09	File folder	

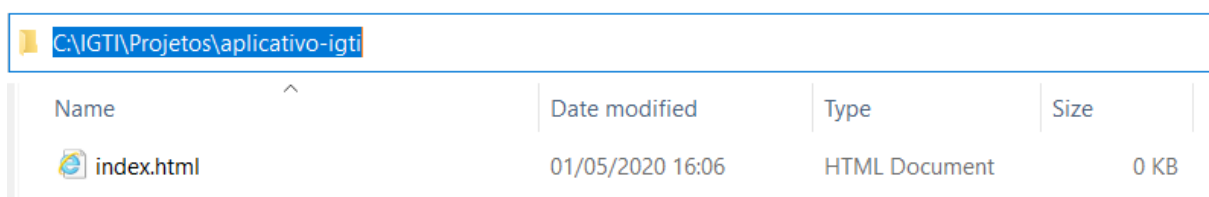
- `git clone`: cria uma cópia local de um projeto que já existe no repositório. Este comando carrega todos os arquivos do projeto, históricos e branches. No exemplo das figuras abaixo foram realizados o clone do projeto do próprio Git. Conforme mencionamos, o Git é um sistema de código aberto, portanto, podemos clonar o projeto do Git e fazer edições caso necessário.

```
MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git clone https://github.com/git/git.git
Cloning into 'git'...
remote: Enumerating objects: 286375, done.
remote: Total 286375 (delta 0), reused 0 (delta 0), pack-reused 286375
Receiving objects: 100% (286375/286375), 135.86 MiB | 2.17 MiB/s, done.
Resolving deltas: 100% (213084/213084), done.
Updating files: 100% (3757/3757), done.
```

C:\IGTI\Projetos			
Name	Date modified	Type	Size
.git	01/05/2020 15:09	File folder	
git	01/05/2020 15:18	File folder	

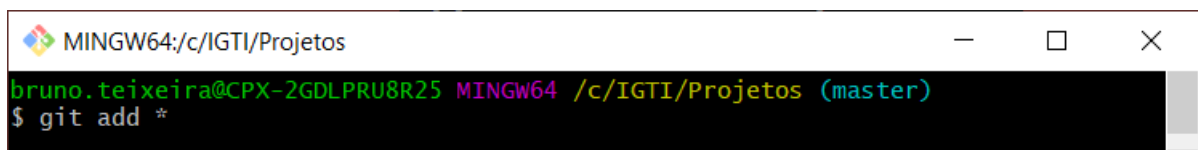
No exemplo acima foi utilizado o clone de um repositório remoto, no qual não necessita de identificação para realizar o clone, porém existem repositórios nos quais é necessário informar o usuário de conexão e para esses casos a sintaxe do comando é `git clone usuário@servidor:/caminho/para/o/repositório`.

- `git add`: este comando transfere ou adiciona os novos arquivos na área temporária (Index ou Stage area). A figura abaixo apresenta a adição de um novo arquivo dentro do repositório criado, com isso vamos transferir esse arquivo para a área temporária com o comando `git add`.



Name	Date modified	Type	Size
index.html	01/05/2020 16:06	HTML Document	0 KB

Executamos o comando `git add` especificando um arquivo ou utilizamos `*` para transferir todos os novos arquivos dentro do repositório.



```

MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git add *
  
```

Uma outra variação é executar o comando `git add` com a extensão dos arquivos que serão transferidos: `git add *.txt`, ou transferir somente os arquivos modificados com o comando `git add -A`.

- `git commit`: este comando salva o histórico do projeto no momento, ou seja, todas as mudanças que estão na área temporária são confirmadas no repositório local. A execução deste comando abrirá o editor de texto padrão para que seja inputado o comentário desta modificação, que está sendo confirmada, conforme a figura abaixo.

```
MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git commit
[master (root-commit) fd62851] Primeiro commit realizado.
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 aplicativo-igti/index.html
```

Se preferir adicionar uma mensagem simples do commit, podemos utilizar o comando `git commit -m "Mensagem"`, conforme a figura abaixo:

```
MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git commit -m "Segundo commit realizado"
[master dfcelfd] Segundo commit realizado
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 aplicativo-igti/default.asp
```

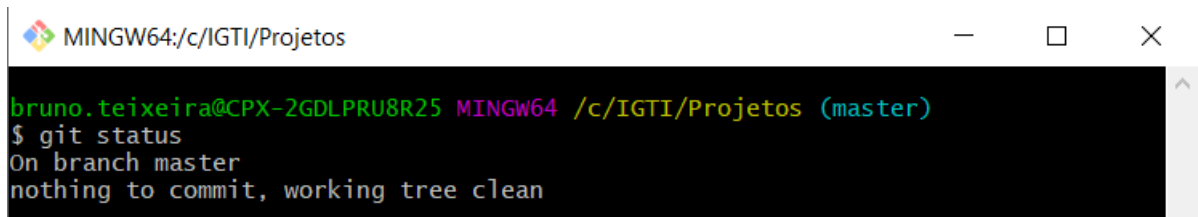
Suponhamos que você tenha errado a mensagem escrita no commit ou simplesmente deseja melhorar a descrição das mudanças realizadas. Caso você tenha comitado e ainda não fez o `push` das suas modificações para o servidor (veremos este comando mais à frente), pode utilizar a flag `--amend`, conforme a figura abaixo:

```
MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git commit --amend
[master 18d9cde] Melhorando a mensagem do último commit realizado.
Date: Fri May 1 16:21:18 2020 -0300
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 aplicativo-igti/default.asp
```

O comando `git commit --amend` modifica a mensagem do commit mais recente, ou seja, o último commit feito por você no projeto. Além de possibilitar a mudança na mensagem do commit, é possível adicionar arquivos que esquecemos ou retirar arquivos comitados por engano. O git cria um commit totalmente novo e corrigido.

- `git status`: este comando apresenta o status das alterações que não foram adicionadas, as modificações realizadas e o que está na área temporária. Se

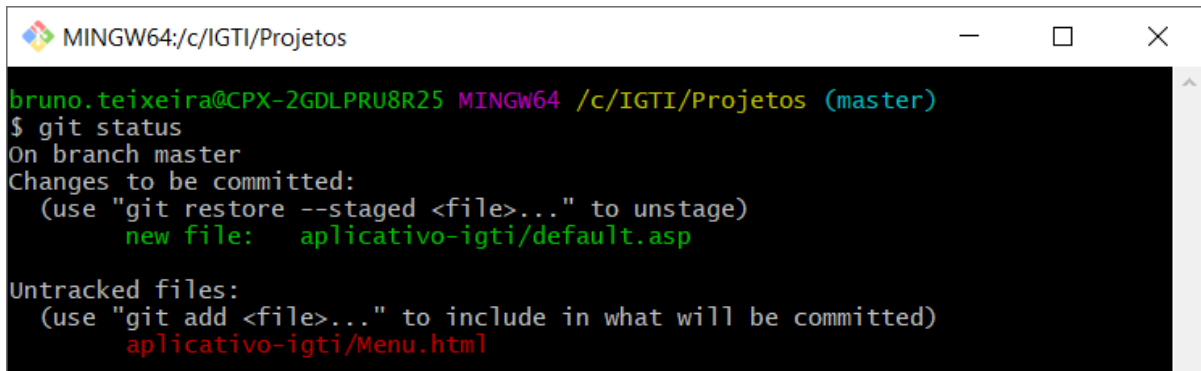
executarmos este comando, não haverá nada a ser apresentado conforme a figura abaixo:



```

MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git status
On branch master
nothing to commit, working tree clean
  
```

No exemplo da figura abaixo, podemos observar que há arquivos na área temporária (default.asp) e arquivos que ainda não foram adicionados para versionamento (menu.html).



```

MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   aplicativo-igti/default.asp

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    aplicativo-igti/Menu.html
  
```

- `git branch`: apresenta os branches que estão disponíveis no repositório remoto e qual está sendo utilizado atualmente no workspace. Podemos observar que atualmente só existe o branch master que está em uso no workspace (*).



```

MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git branch
* master
  
```

- `git checkout`: cria um novo branch no repositório local, a partir do branch atual, e modifica o workspace para este novo branch. Com a criação de um novo branch, podemos realizar modificações nos arquivos e comitar neste branch, o branch principal (master) não é modificado.

```

MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git checkout -b funcionalidade_XYZ
Switched to a new branch 'funcionalidade_XYZ'

```

Nesse novo branch, adicionamos um novo arquivo (Menu.html) e modificamos o conteúdo de um arquivo existente (Index.html), realizamos a transferência desses arquivos para a área temporária e comitamos no repositório local, conforme a figura abaixo.

```

MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (funcionalidade_XYZ)
$ git add *

bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (funcionalidade_XYZ)
$ git commit -m "Modificacoes realizadas no branch"
[funcionalidade_XYZ a496d4c] Modificacoes realizadas no branch
2 files changed, 2 insertions(+)
create mode 100644 aplicativo-igti/Menu.html

```



Podemos retornar para o branch principal (master) com o comando `git checkout` e verificar que esses arquivos não estão no branch.

```

MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (funcionalidade_XYZ)
$ git checkout master
Switched to branch 'master'

```

C:\IGTI\Projetos\aplicativo-igti

Name	Date modified	Type	Size
 default.asp	01/05/2020 16:18	ASP File	0 KB
 index.html	01/05/2020 16:55	HTML Document	0 KB

- `git merge`: este comando é utilizado para combinar mudanças realizadas em dois branches distintos. Vamos pegar como exemplo o branch que criamos anteriormente: `funcionalidade_XYZ`, ele não está no branch `master`. Se executarmos o merge todas as mudanças serão combinadas no branch atual.

Podemos observar pela figura abaixo, que estamos atualmente no branch master:

```

MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git branch
  funcionalidade_XYZ
* master
  
```

Se mergearmos o branch funcionalidade_XYZ com o master, o master passará a ter todas as mudanças que foram realizadas no branch funcionalidade_XYZ, conforme a figura abaixo:

```

MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git merge funcionalidade_XYZ
Updating 18d9cde..a496d4c
Fast-forward
 aplicativo-igti/Menu.html | 0
 aplicativo-igti/index.html | 2 ++
 2 files changed, 2 insertions(+)
 create mode 100644 aplicativo-igti/Menu.html
  
```

Podemos excluir o branch que já foi mergeado com o comando `git branch -d funcionalidade_XYZ`, conforme a figura abaixo.

```

MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git branch -d funcionalidade_XYZ
Deleted branch funcionalidade_XYZ (was a496d4c).
  
```

- `git log`: apresenta os logs dos commits realizados no branch atual. No exemplo da figura abaixo podemos visualizar parte dos logs das operações realizadas no repositório local.

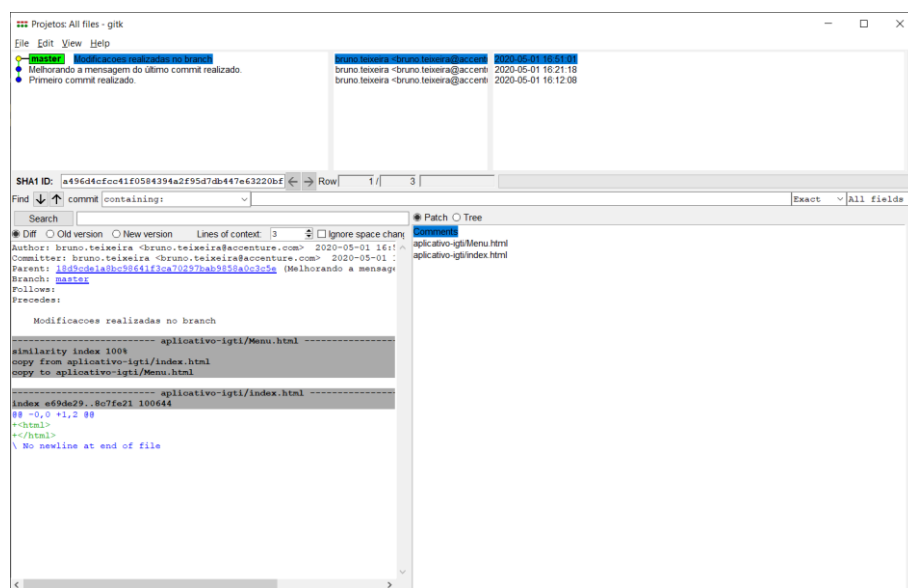
```

MINGW64:/c:/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c:/IGTI/Projetos (master)
$ git log
commit a496d4cfcc41f0584394a2f95d7db447e63220bf (HEAD -> master)
Author: bruno.teixeira <bruno.teixeira@accenture.com>
Date: Fri May 1 16:51:01 2020 -0300

    Modificacoes realizadas no branch
:...skipping...
commit a496d4cfcc41f0584394a2f95d7db447e63220bf (HEAD -> master)
Author: bruno.teixeira <bruno.teixeira@accenture.com>

```

- gitk: abre a interface gráfica padrão do git para visualização da estrutura das árvores, bem como os logs dos commits realizados, conforme a figura abaixo.



Os comandos abaixo terão seus exemplos apresentados na seção seguinte.

- git pull: transfere as mudanças realizadas no repositório local para o repositório remoto.
- git push: atualiza o repositório local com a versão mais recente existente no repositório remoto.
- git fetch: atualiza todo o repositório local com todos os commits e versões existentes no repositório remoto.
- git reset: basicamente desfaz as alterações realizadas no workspace para a versão existente no repositório remoto.

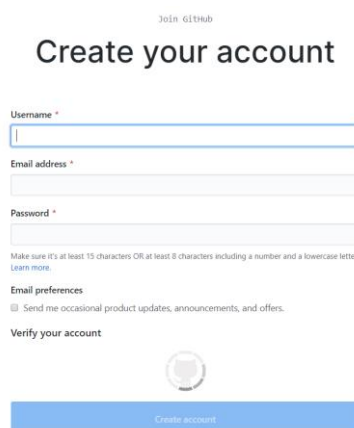
3.4. GitHub

O GitHub, como seu próprio nome diz, é um “hub” ou uma plataforma de hospedagem para projetos que utilizam o software de controle de versão Git. Embora sua maior utilização esteja direcionada para versionamento de código fonte, é possível utilizá-la para versionamento de outros tipos de projetos ou documentos. Além disso, ela oferece funcionalidades extras aplicadas ao Git.

Para acessá-la é necessário ter uma conta na plataforma, que explicaremos nos passos a seguir:

Criar a conta:

O processo de criação da conta é simples, basta preencher o formulário e confirmar a autenticidade da conta com o e-mail enviado pela plataforma.



Com a conta criada podemos criar nossos repositórios, contribuir em repositórios públicos, participar dos fóruns, entre outras funcionalidades que a plataforma oferece. Seguiremos com o passo a passo para criação e publicação de um repositório no GitHub.

Criar um repositório:

Precisamos criar um repositório no GitHub para publicar o projeto já existente no repositório local do Git. Para isso, definimos um nome para o repositório, a privacidade de acesso e se terá os arquivos de informação.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner

Repository name *

brunoaugustoteixeira

IGTI_Bootcamp

Great repository names are short and memorable. Need inspiration? How about [expert-spork?](#)

Description (optional)

Public

Anyone can see this repository. You choose who can commit.

Private

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ Initialize this repository with a README

This will let you immediately clone the repository to your computer.

Add .gitignore: None

Add a license: None

Create repository

Com o repositório criado, podemos fazer o upload dos arquivos diretamente pelo GitHub ou transportar o repositório local com o comando sugerido pela página, conforme a figura abaixo.

Code

Issues 0

Pull requests 0

Actions

Projects 0

Wiki

Security 0

Insights

Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop

or

HTTPS

SSH

https://github.com/brunoaugustoteixeira/IGTI_Bootcamp.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# IGTI_Bootcamp" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/brunoaugustoteixeira/IGTI_Bootcamp.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/brunoaugustoteixeira/IGTI_Bootcamp.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

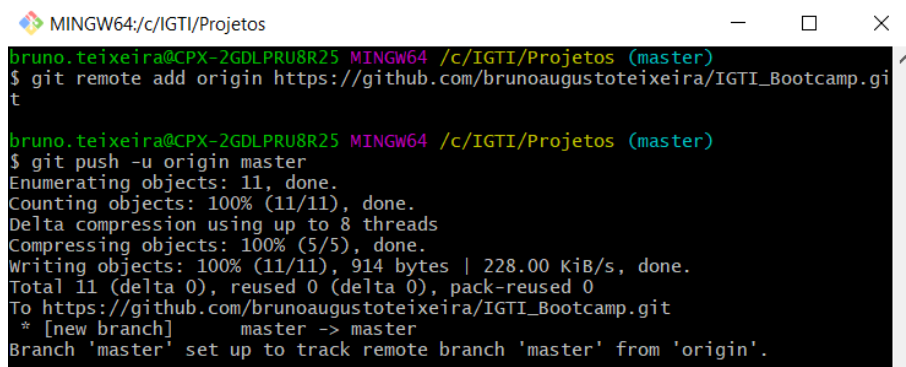
Ao executar o comando `git remote add`, estaremos vinculando ou definindo o repositório remoto do nosso repositório local. Esse comando necessita de informar um nome (`origin`) para o repositório remoto e o endereço.

```
git remote add origin https://github.com/brunoaugustoteixeira/IGTI_Bootcamp.git
```

Com a definição do repositório remoto, podemos transportar o branch do repositório local para o repositório remoto com o comando:

```
git push -u origin master
```

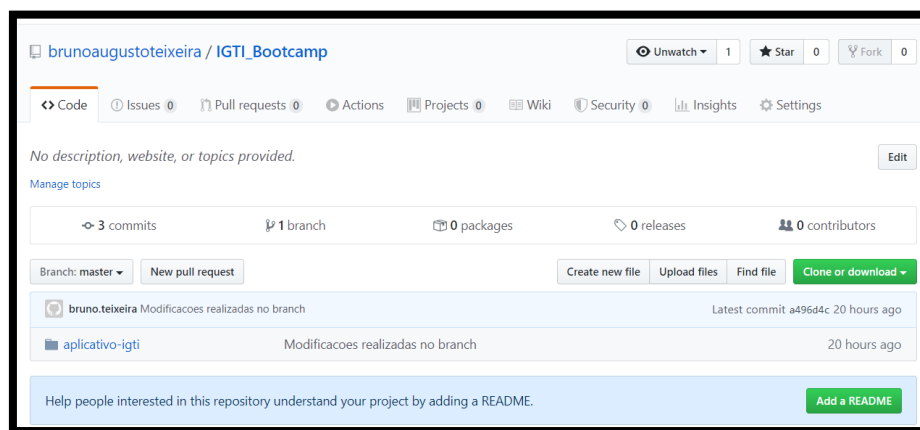
A execução deste comando irá abrir um pop-up para inserir as credenciais (usuário e senha) do GitHub, a fim de garantir sua identidade no repositório remoto informado, conforme a figura abaixo.



```

MINGW64/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git remote add origin https://github.com/brunoaugustoteixeira/IGTI_Bootcamp.git
$ git push -u origin master
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (11/11), 914 bytes | 228.00 KiB/s, done.
Total 11 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/brunoaugustoteixeira/IGTI_Bootcamp.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
  
```

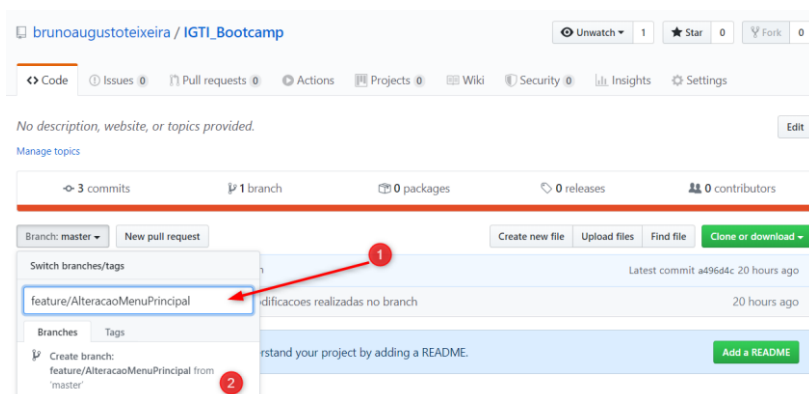
Podemos observar que nosso repositório local já está no repositório remoto pela figura abaixo.



Criar um Branch:

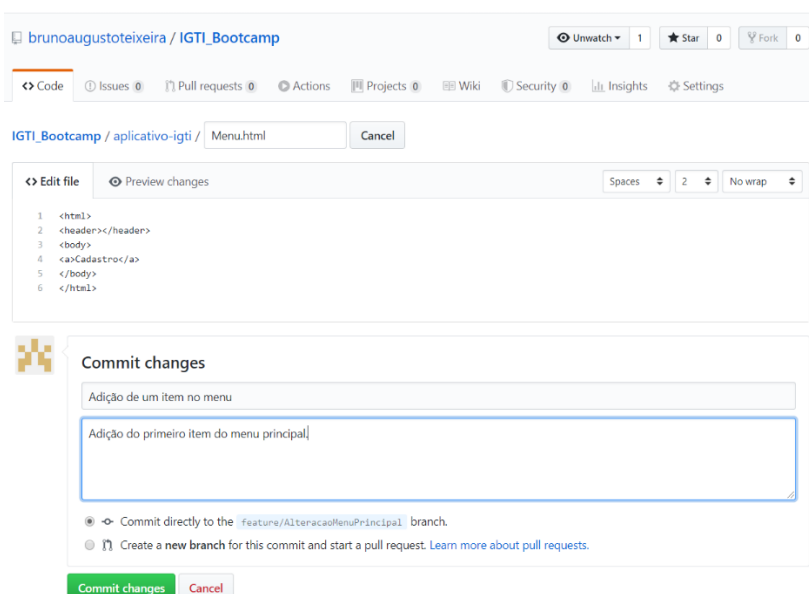
Como mencionamos anteriormente, o GitHub não é só um local para armazenamento de projetos versionados pelo Git, além disso é possível também gerenciar seu projeto diretamente na plataforma. No exemplo da figura abaixo, iremos

criar um novo branch para realizar uma modificação num dos arquivos do projeto. A criação do branch é simples, basta digitar o nome do novo branch e criá-lo.



Commit:

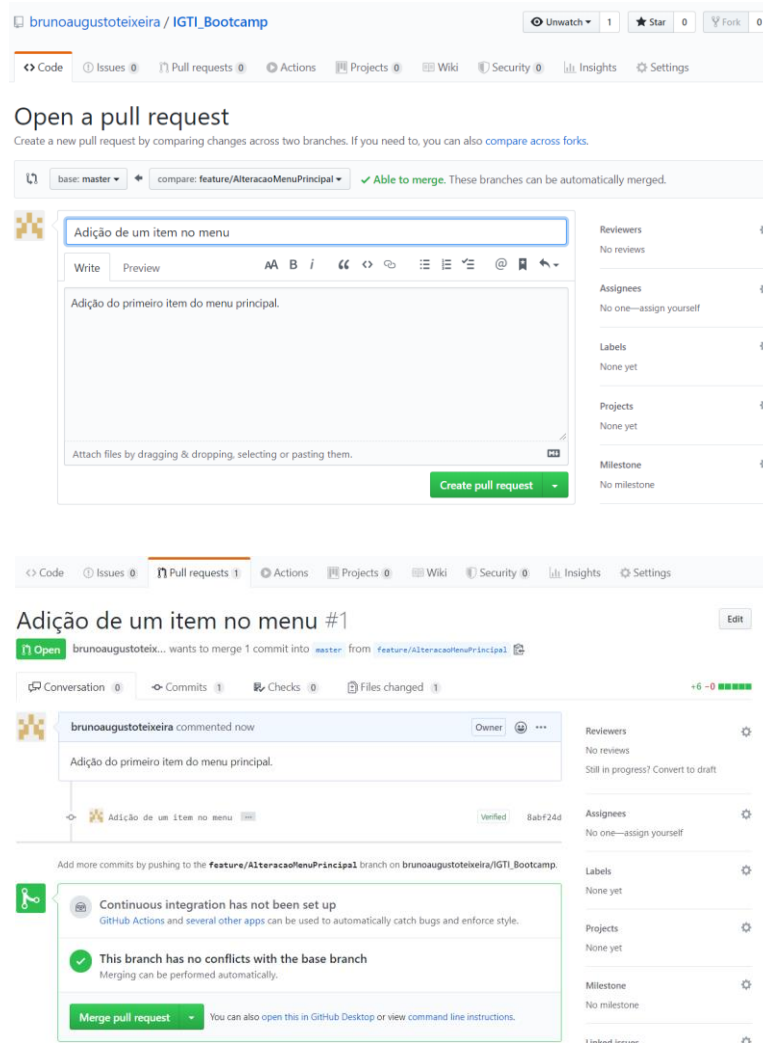
Podemos realizar modificações nos arquivos diretamente pelo GitHub, no exemplo da figura abaixo realizamos uma modificação num dos arquivos do projeto e realizamos o commit.



Criar um pull request:

O desenvolvimento de novas melhorias ou até mesmo correções de bugs, geralmente são realizados em branches separados para posteriormente serem mergeados ao branch principal. No GitHub, por ser uma plataforma colaborativa, esse

merge ocorre com uma solicitação de pull request no qual é possível adicionar revisores às mudanças que estão sendo incorporadas ao branch principal. Portanto, para realizar o merge de um branch no GitHub é necessário criar um pull request, adicionar os comentários, revisores etc., vinculados à este pull request, e posteriormente realizar o merge, conforme as figuras abaixo.



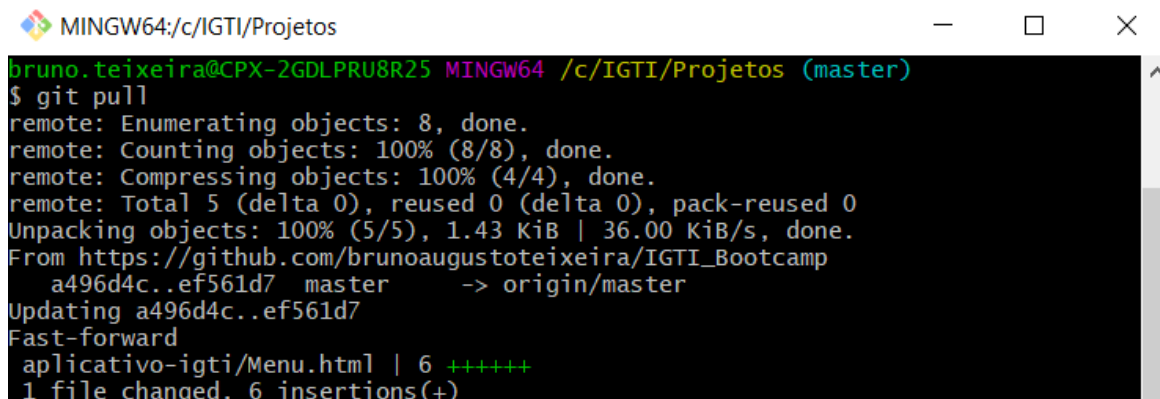
The first screenshot shows the 'Open a pull request' page for the repository 'brunoaugustoteixeira / IGTI_Bootcamp'. It displays the 'base: master' and 'compare: feature/AlteracaoMenuPrincipal' branches, indicating they are 'Able to merge'. The pull request title is 'Adição de um item no menu' and the description is 'Adição do primeiro item do menu principal.' The 'Create pull request' button is visible.

The second screenshot shows the pull request page after it has been created. It displays the title 'Adição de um item no menu #1' and the description 'Adição do primeiro item do menu principal.' The pull request is owned by 'brunoaugustoteixeira' and is currently in progress. The 'Merge pull request' button is visible at the bottom.

Com o repositório local vinculado (publicado) num repositório remoto, podemos avaliar os comandos do fluxo entre o repositório local e remoto, são eles:

- `git pull`: atualiza o repositório local com a versão mais recente existente no repositório remoto. Esse comando é importante quando o projeto tem várias equipes e as mudanças são realizadas e publicadas por cada um no repositório remoto e precisamos atualizar o nosso repositório local com as mudanças já

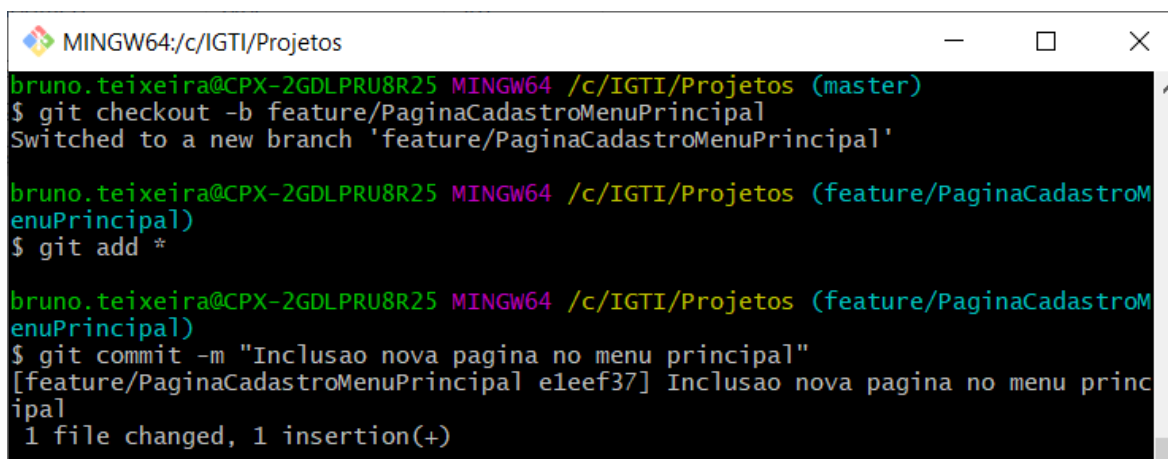
realizadas. No último exemplo, fizemos uma mudança no projeto diretamente pelo GitHub, que já foi mergeada no master. Para atualizarmos nosso repositório local basta executar o comando `git pull` e todas as mudanças serão transferidas para o repositório local para o branch ativo no workspace, conforme a figura abaixo.



```

MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git pull
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), 1.43 KiB | 36.00 KiB/s, done.
From https://github.com/brunoaugustoteixeira/IGTI_Bootcamp
   a496d4c..ef561d7  master       -> origin/master
Updating a496d4c..ef561d7
Fast-forward
 aplicativo-igti/Menu.html | 6 ++++++
 1 file changed, 6 insertions(+)
  
```

- `git push`: tranfere as mudanças realizadas no repositório local para o repositório remoto. Para o exemplo da figura abaixo foi criado um novo branch, realizada uma alteração no código, transportada as mudanças para a área temporária e posteriormente esse branch foi comitado no repositório local.



```

MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (master)
$ git checkout -b feature/PaginaCadastroMenuPrincipal
Switched to a new branch 'feature/PaginaCadastroMenuPrincipal'

bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (feature/PaginaCadastroMenuPrincipal)
$ git add *

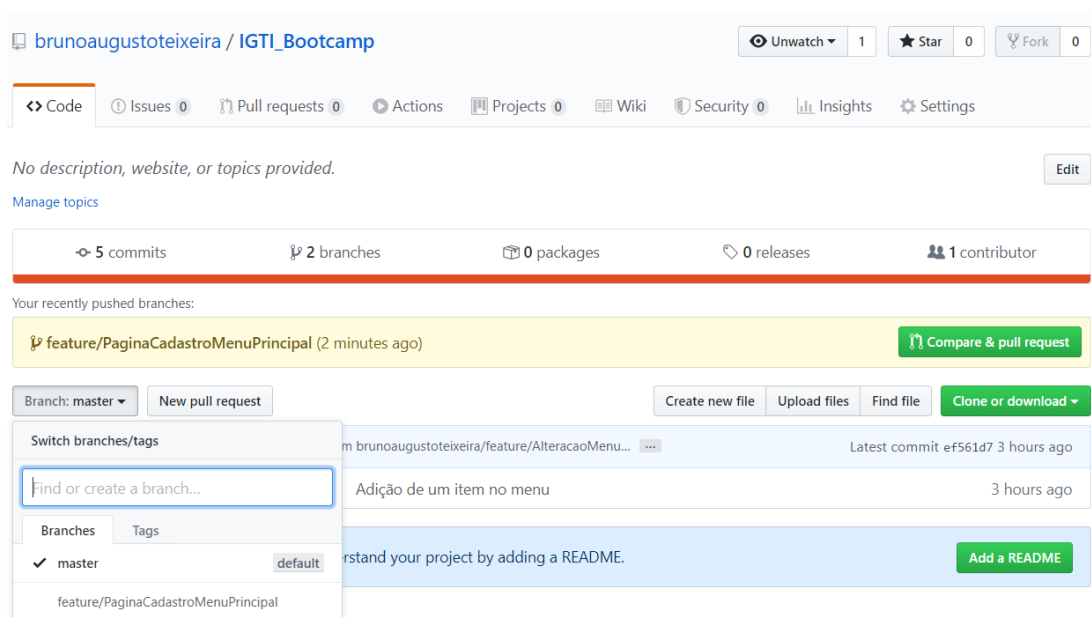
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (feature/PaginaCadastroMenuPrincipal)
$ git commit -m "Inclusao nova pagina no menu principal"
[feature/PaginaCadastroMenuPrincipal e1eef37] Inclusao nova pagina no menu principal
1 file changed, 1 insertion(+)
  
```

Para disponibilizarmos esse branch no repositório remoto é necessário transferi-lo, ou seja, ninguém conseguirá acessá-lo até que seja transportado.

```

MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (feature/PaginaCadastroMenuPrincipal)
$ git push origin feature/PaginaCadastroMenuPrincipal
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 439 bytes | 439.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature/PaginaCadastroMenuPrincipal' on GitHub by visiting:
remote:   https://github.com/brunoaugustoteixeira/IGTI_Bootcamp/pull/new/feature/PaginaCadastroMenuPrincipal
remote:
To https://github.com/brunoaugustoteixeira/IGTI_Bootcamp.git
 * [new branch]      feature/PaginaCadastroMenuPrincipal -> feature/PaginaCadastroMenuPrincipal
  
```

Se observamos no GitHub, o branch já está disponível para clone por outras pessoas, conforme figura abaixo.



- `git fetch`: atualiza todo o repositório local com todos os commits e versões existentes no repositório remoto. Esse comando é semelhante ao `pull`, porém todos os branches do repositório local serão atualizados. Note na figura abaixo que embora o workspace esteja no branch `feature/PaginaCadastroMenuPrincipal`, a execução do comando irá atualizar também os outros branches do repositório local.

```

MINGW64:/c/IGTI/Projetos
bruno.teixeira@CPX-2GDLPRU8R25 MINGW64 /c/IGTI/Projetos (feature/PaginaCadastroM
enuPrincipal)
$ git fetch origin

```

- `git reset`: este comando permite desfazer as alterações realizadas no workspace para a versão existente no repositório remoto. Este comando é conveniente realizar um fetch para posteriormente o reset: `git reset --hard origin/master`.

Capítulo 4. Heroku

4.1. Introdução ao Heroku

O Heroku é uma plataforma de serviço na nuvem (PaaS), que permite os desenvolvedores criar, executar e operar aplicações. Atualmente, desenvolvedores, pequenas equipes, empresas de todos os tamanhos utilizam o Heroku para implantar, gerenciar e escalar aplicações, pois a plataforma oferece a hospedagem e possui uma integração com um dos sistemas de controle de versão mais popular, o Git. Essa integração com o Git possibilita a realização da prática de desenvolvimento de Software de DevOps, ou seja, é possível aplicar a integração contínua no projeto. A plataforma é preferencialmente escolhida pela comunidade devido à quantidade de serviços suportados. Como as aplicações feitas em: Node.js, Ruby, Java, Php, Python, Go, Scala e Clojure.

Para a utilização do Heroku é necessário possuir uma conta na plataforma, cujo processo é muito simples, portanto, é importante realizar este passo para prosseguir com os exemplos do curso. A hospedagem da aplicação necessita de um repositório no GitHub, embora existam outras formas de realizar este processo.

Vantagens:

- Produtividade.
- Integração com GitHub.
- Disponibilidade de add-ons.

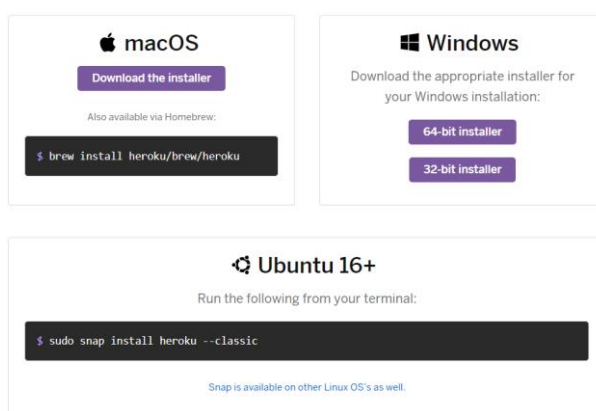
Desvantagens:

- Limitação de 512Mb RAM.
- Aplicação adormece.

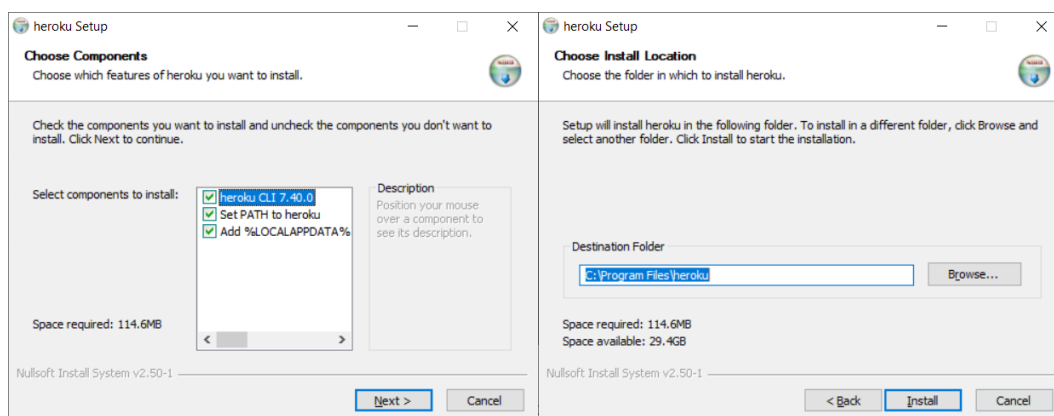
4.2. Instalação do Heroku CLI

A instalação do Heroku CLI (*Command Line Interface*) é utilizada para o gerenciamento e dimensionamento das aplicações, assim como o provisionamento de complementos, verificação de logs da aplicação e, por fim, a execução da aplicação localmente. O Heroku CLI pode ser instalado em várias plataformas e neste curso iremos apresentar a instalação no Windows, conforme passos a seguir:

- Faça o download do instalador correspondente ao sistema operacional a ser instalado.



- Instale com as configurações padrões para que os comandos do Heroku operem no prompt comando do SO.



Após a instalação, acesse o prompt de comando e digite o comando `heroku login`. Nesta primeira execução será aberta uma página para inserção das credenciais de acesso da sua conta na plataforma, conforme a figura abaixo:

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\bruno.teixeira>heroku login
heroku: Press any key to open up the browser to login or q to exit:
Opening browser to https://cli-auth.heroku.com/auth/cli/browser/4e9ac9ce-0995-4691-9158-ed16585a8df1
Logging in... done
Logged in as brunoaugustoteixeira@gmail.com
```

4.3. Implantação no Heroku

Uma vez que a aplicação está disponível no repositório do GitHub, podemos realizar a implantação no Heroku, conforme passos a seguir:

Criação:

Acesse o diretório da aplicação e execute o comando `heroku create`. Após essa execução, um git remote adicional será vinculado a este repositório, conforme figura abaixo:

```
C:\WINDOWS\system32\cmd.exe
C:\Users\bruno.teixeira\Desktop\IGTI\Bootcamp_Modulo5\Heroku\backend-app>heroku create
Creating app... done, ⬢ stormy-dawn-44846
https://stormy-dawn-44846.herokuapp.com/ | https://git.heroku.com/stormy-dawn-44846.git

C:\Users\bruno.teixeira\Desktop\IGTI\Bootcamp_Modulo5\Heroku\backend-app>git remote -v
destination https://github.com/brunoaugustoteixeira/IGTI_backend-app.git (fetch)
destination https://github.com/brunoaugustoteixeira/IGTI_backend-app.git (push)
heroku https://git.heroku.com/stormy-dawn-44846.git (fetch)
heroku https://git.heroku.com/stormy-dawn-44846.git (push)
```

Deploy:

Com isso podemos realizar o deploy da aplicação com o comando:

`git push heroku master`, conforme a figura abaixo.

```
Command Prompt
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\bruno.teixeira\Desktop\IGTI\Bootcamp_Modulo5\Heroku\backend-app>git push heroku master
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 4.43 KiB | 1.48 MiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Node.js app detected
remote: -----> Build
remote:
remote: -----> Caching build
remote:      - node_modules
remote:
remote: -----> Pruning devDependencies
remote:      audited 70 packages in 0.772s
remote:      found 5 vulnerabilities (4 moderate, 1 high)
remote:      run `npm audit fix` to fix them, or `npm audit` for details
remote:
remote: -----> Build succeeded!
remote: ! This app may not specify any way to start a node process
remote:   https://devcenter.heroku.com/articles/nodejs-support#default-web-process-type
remote: -----> Compressing...
remote:      Done: 23.5M
remote: -----> Launching...
remote:      Released v3
remote:      https://arcane-wave-81944.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/arcane-wave-81944.git
 * [new branch]      master -> master
```

Iniciar:

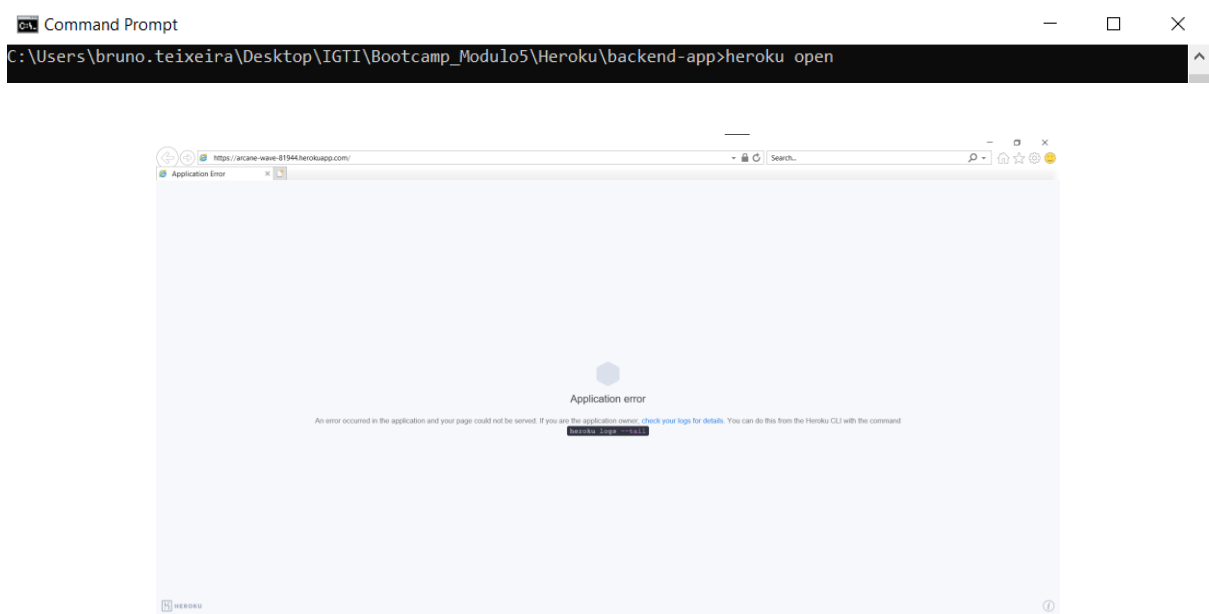
Uma vez que a aplicação foi publicada, o heroku cria um “dyno”, que seria uma versão compactada do executável da aplicação que está armazenada. Esse dyno precisa ser iniciado, ou seja, será iniciada a aplicação publicada num dos servidores da Heroku.

O comando para iniciar a aplicação é: `heroku ps:scale web=1`, onde estamos dimensionando a aplicação para executar somente num servidor.

```
Command Prompt
C:\Users\bruno.teixeira\Desktop\IGTI\Bootcamp_Modulo5\Heroku\backend-app>heroku ps:scale web=1
Scaling dynos... done, now running web at 1:Free
```

Acessar:

A aplicação pode ser visualizada ou acessada com o comando `heroku open`. Com a execução deste comando podemos observar que a aplicação abrirá uma página com alguns erros:



Visualizar logs:

Podemos visualizar a causa do erro com o comando de visualização de logs `heroku log -tails`. Pelo log é possível identificar que a aplicação não abriu, pois não existe o arquivo no qual o Heroku utiliza para definir o comando de início da aplicação, conforme a figura abaixo.

```
Command Prompt - heroku logs --tail
C:\Users\bruno.teixeira\Desktop\IGTI\Bootcamp_Modulo5\Heroku\backend-app>heroku logs --tail
2020-05-03T19:55:34.639602+00:00 app[api]: Release v1 created by user brunoaugustoteixeira@gmail.com
2020-05-03T19:55:34.639602+00:00 app[api]: Initial release by user brunoaugustoteixeira@gmail.com
2020-05-03T19:55:34.856271+00:00 app[api]: Enable Logplex by user brunoaugustoteixeira@gmail.com
2020-05-03T19:55:34.856271+00:00 app[api]: Release v2 created by user brunoaugustoteixeira@gmail.com
2020-05-03T19:56:33.000000+00:00 app[api]: Build started by user brunoaugustoteixeira@gmail.com
2020-05-03T19:56:45.967762+00:00 app[api]: Scaled to web@1:Free by user brunoaugustoteixeira@gmail.com
2020-05-03T19:56:45.946046+00:00 app[api]: Deploy 40fe8670 by user brunoaugustoteixeira@gmail.com
2020-05-03T19:56:45.946046+00:00 app[api]: Release v3 created by user brunoaugustoteixeira@gmail.com
2020-05-03T19:56:46.000000+00:00 app[api]: Build succeeded
2020-05-03T19:56:50.515996+00:00 heroku[web.1]: State changed from starting to crashed
2020-05-03T19:56:50.519694+00:00 heroku[web.1]: State changed from crashed to starting
2020-05-03T19:56:50.437374+00:00 app[web.1]: npm ERR! missing script: start
2020-05-03T19:56:50.445872+00:00 app[web.1]:
2020-05-03T19:56:50.446221+00:00 app[web.1]: npm ERR! A complete log of this run can be found in:
2020-05-03T19:56:50.446376+00:00 app[web.1]: npm ERR! /app/.npm/_logs/2020-05-03T19_56_50_438Z-debug.log
2020-05-03T19:56:55.017258+00:00 heroku[web.1]: State changed from starting to crashed
2020-05-03T19:56:54.920759+00:00 app[web.1]: npm ERR! missing script: start
2020-05-03T19:56:54.928181+00:00 app[web.1]:
2020-05-03T19:56:54.928575+00:00 app[web.1]: npm ERR! A complete log of this run can be found in:
2020-05-03T19:56:54.928869+00:00 app[web.1]: npm ERR! /app/.npm/_logs/2020-05-03T19_56_54_921Z-debug.log
2020-05-03T20:02:25.010705+00:00 heroku[router]: at=error code=H10 desc="App crashed" method=GET path="/" host=arcane-wave-81944.herokuapp.com request_id=91750ee2-c47f-402d-b7b8-66a80d35f24b fwd="186.206.255.55" dyno= connect= service= status=503 bytes= protocol=https
2020-05-03T20:02:27.391824+00:00 heroku[router]: at=error code=H10 desc="App crashed" method=GET path="/favicon.ico" host=arcane-wave-81944.herokuapp.com request_id=5e410cc0-d3aa-4caf-a95c-8dbc5d042381 fwd="186.206.255.55" dyno= connect= service= status=503 bytes= protocol=https
```

Procfile:

O Procfile é um arquivo texto para definir explicitamente qual comando deve ser executado para iniciar seu aplicativo, semelhante à quando iniciamos a aplicação

com o comando `node nome_do_arquivo.js`. A figura abaixo apresenta o arquivo Procfile com o comando de início, assim, é necessário realizar o commit no repositório remoto e em seguida podemos executar o comando `git heroku push master` para que o deploy da aplicação ocorra automaticamente.

```
backend-app > H procfile
1 web: node index.js|
```

Executar localmente:

O Heroku permite a execução local de uma aplicação, para isso basta executar o comando `heroku local web` e abrir o navegador no endereço informado no console, após o deploy da aplicação.

Referências

MONGODB. *Home*. 2020. Disponível em: <<https://www.mongodb.com/>>. Acesso em: 14 jan. 2021.

DB-ENGINES. *DB-Engines Ranking*. 2020. Disponível em: <<https://db-engines.com/en/ranking>>. Acesso em: 14 jan. 2021.

BSON. *Home*. Disponível em: <<http://bsonspec.org/>>. Acesso em: 14 jan. 2021.

MLAB. *Home*. Disponível em: <<http://mlab.com/>>. Acesso em: 14 jan. 2021.

GIT. *Home*. Disponível em: <<https://git-scm.com/>>. Acesso em: 14 jan. 2021.

HEROKU. *Home*. Disponível em: <<https://www.heroku.com/>>. Acesso em: 14 jan. 2021.