1 Introdução

O desenvolvimento moderno de aplicações Web é comumente realizado utilizando-se diferentes tecnologias. Muitas vezes, sua combinação dá origem a algo que tem levado o nome de desenvolvimento “Full Stack”. Uma solução desenvolvida segundo esse paradigma possui, em geral, duas aplicações independentes que se comunicam por meio de uma interface bem definida. Uma delas oferece interfaces gráfica para o usuário e geralmente é chamada de aplicação Front End. A outra é responsável por disponibilizar as funcionalidades do sistema e leva o nome de aplicação Back End. Em geral, ela faz uso de um sistema independente que possui implementações eficientes para operações de acesso à memória secundária.

Além disso, seja no Front End ou no Back End, é comum o uso de diferentes *frameworks* que supostamente entregam um nível maior de abstração e promovem a produtividade dos desenvolvedores.

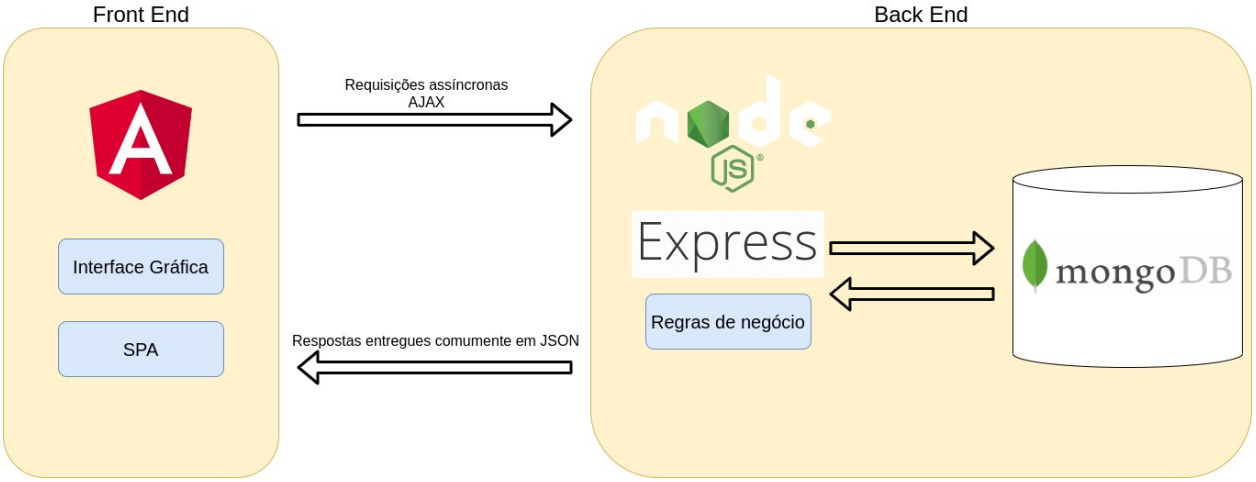
Nos dias atuais, uma das combinações mais utilizadas para esse fim tem a sigla “**MEAN**” associada, a qual deriva de “**MongoDB**”, “**Express**”, “**Angular**” e “**NodeJS**”.

Neste material desenvolveremos uma aplicação que faz uso da “pilha” MEAN.

2 Desenvolvimento

2.1 (Sobre o MongoDB) Lembre-se, novamente, da arquitetura de nossa aplicação, ilustrada na Figura 2.1.1.

Figura 2.1.1



- No momento, possuímos o Front End em Angular e uma parte do Back End com NodeJS + Express. Não temos, contudo, uma base de dados que permita o armazenamento de dados em meio persistente. Passaremos a utilizar o MongoDB para esse fim.

- Uma definição para o MongoDB é a seguinte:

Uma base de dados NoSQL que armazena documentos organizados em coleções.

Fazendo uma analogia, os documentos seriam as linhas de uma tabela no modelo relacional. Uma coleção é análoga a uma tabela. É claro, sem as restrições e detalhes do modelo relacional.

- A Tabela 2.1.1 mostra uma breve comparação entre bases SQL e NoSQL.

Características	NoSQL	SQL
Exemplos de bancos de dados	MongoDB, HBase, Cassandra	MySQL, PostreSQL, MS Server
Estrutura	Uma coleção pode armazenar documentos com estruturas diferentes.	Tabelas pré-definidas.
Sobre junções	Não é um ponto forte	Uma das principais características

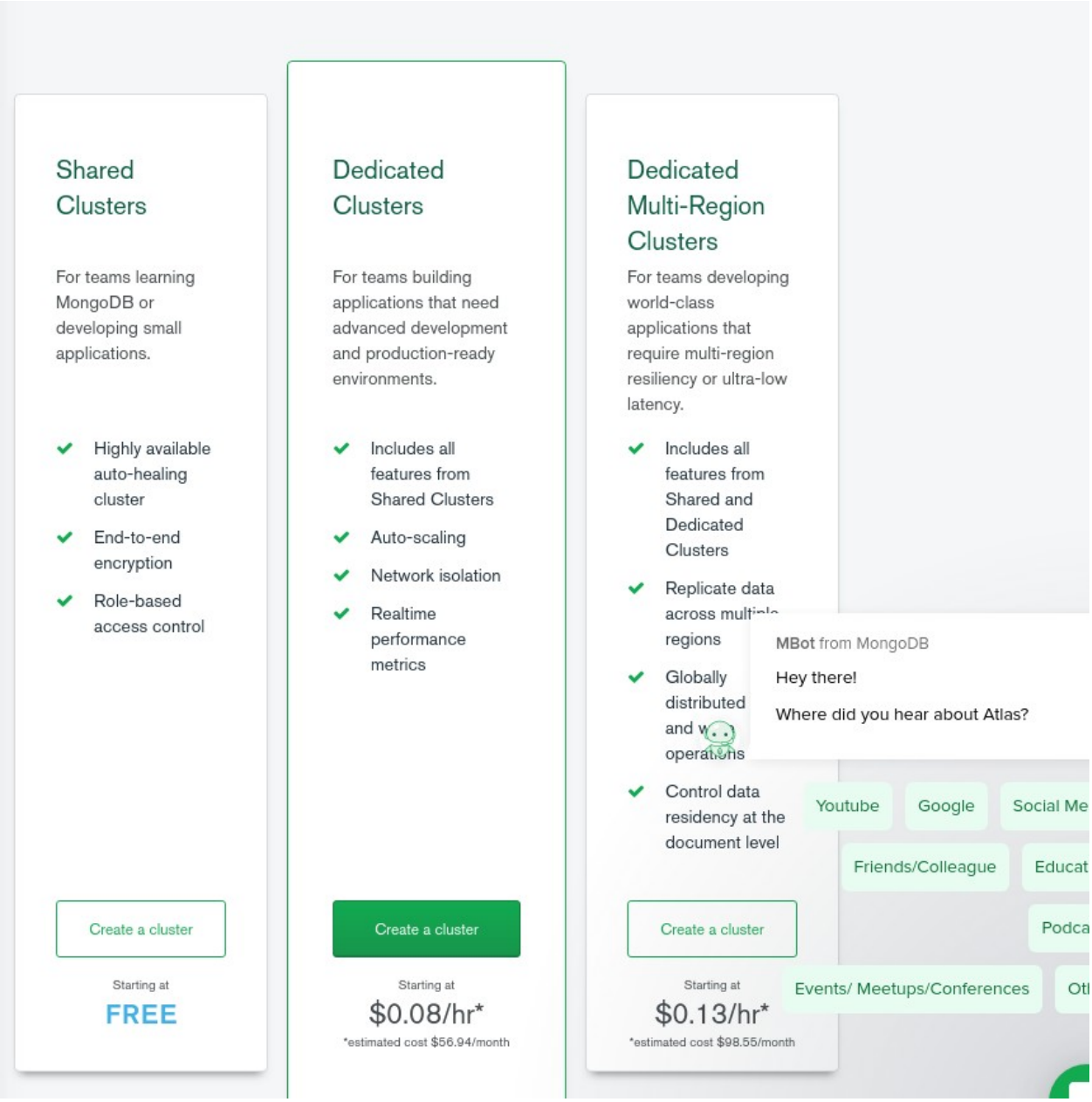
2.2 (Uso do MongoDB com Atlas) Há algumas formas para se utilizar o MongoDB. Podemos fazer o download e executar localmente, por exemplo. Uma outra solução se chama **Atlas**. Trata-se de uma solução em nuvem gratuita que dá acesso a uma instância MongoDB remotamente. Neste material, utilizaremos a segunda opção.

- Comece visitando o Link 2.2.1.

Link 2.2.1
https://www.mongodb.com/

- A seguir, clique em Cloud >> Atlas. Na tela seguinte, clique em **Start free** e crie uma conta para você. Uma vez feito o login, você deve ver uma tela parecida com aquela exibida pela Figura 2.2.1.

Figura 2.2.1


Shared Clusters
For teams learning MongoDB or developing small applications.
Highly available auto-healing cluster
End-to-end encryption
Role-based access control
Create a cluster
Starting at
FREE
Dedicated Clusters
For teams building applications that need advanced development and production-ready environments.
Includes all features from Shared Clusters
Auto-scaling
Network isolation
Realtime performance metrics
Create a cluster
Starting at
$0.08/hr*
*estimated cost $56.94/month
Dedicated Multi-Region Clusters
For teams developing world-class applications that require multi-region resiliency or ultra-low latency.
Includes all features from Shared and Dedicated Clusters
Replicate data across multi
regions
Globally distributed and w
operations
Control data residency at the document level
Create a cluster
Starting at
$0.13/hr*
*estimated cost $98.55/month
MBot from MongoDB
Hey there!
Where did you hear about Atlas?
Youtube
Google
Social Me
Friends/Colleague
Educat
Podca
Events/ Meetups/Conferences
Ot

- Clique no botão **Create a cluster** que está próximo do texto **FREE**.

- A seguir, repare que você pode escolher qual serviço de computação em nuvem deseja utilizar. Mantenha a opção **AWS** e **us-east-1**.

- Logo abaixo, no menu **Cluster Tier**, escolha a primeira opção: **M0 Sandbox (Free forever**).

- Mantenha as demais opções com seu valor padrão e clique em **Create Cluster**. Note que a alocação do Cluster pode demorar alguns minutos.

- No menu à esquerda, clique em **Database Access** e, então, escolha a aba **Database Users**. Clique em **Add New Database User** para criar um novo usuário que será utilizado para acessar a base.

- Mantenha o método de autenticação com o valor **Password**. Escolha um nome de usuário e senha. Copie esses valores para um bloco de notas temporário para utilizarmos posteriormente.

- Mantenha o tipo **Read and Write to any Database**.

- Clique em **Create User**.

- Depois disso, precisamos especificar os endereços IP que terão acesso à base. Para isso, no menu à esquerda, clique em **Network access**. Depois de clicar em **Add IP Address**, você pode escolher configurar somente o seu endereço IP (que, se for dinâmico, deverá ser atualizado a cada vez que muda) ou permitir o acesso vindo de qualquer IP, o que é menos seguro e certamente não deve ser utilizado em produção. Faça a sua escolha e clique em **Confirm**.

2.3 (Pacote Mongoose) Há um pacote oficial para acesso a instâncias MongoDB apropriado para aplicações NodeJS. Ele se chama **mongodb**. Contudo, iremos utilizar o pacote chamado Mongoose. Sua implementação utiliza o mongodb internamente e torna o acesso um pouco mais simples. Além disso, ele permite a definição de um modelo de dados que, em geral, bases NoSQL não possuem, como mencionamos.

- A instalação do pacote Mongoose pode ser feita com

npm install --save mongoose

2.4 (Modelo de dados para o Back End) No Back End, do modelo de dados da aplicação. Para isso, clique com o direito na pasta **backend** e crie uma pasta chamada **models**.

- Na pasta **models**, crie um arquivo chamado **cliente.js**.

- A definição será feita utilizando recursos do pacote **mongoose**. Veja seu conteúdo na Listagem 2.4.1. Repare nos comentários.

Listagem 2.4.1

```
//importando o pacote
const mongoose = require ('mongoose');

//definindo o "schema"
//note a semelhança com recursos de bases relacionais
const clienteSchema = mongoose.Schema ({
  nome: {type: String, required: true},
  fone: {type: String, required: false, default: '00000000'},
  email: {type: String, required: true}

});

//criamos o modelo associado ao nome Cliente e exportamos
//tornando acessível para outros módulos da aplicação
module.exports = mongoose.model('Cliente', clienteSchema);
```

2.5 (Criando uma instância do tipo Cliente) Quando uma requisição post for recebida, desejamos criar uma instância de Cliente para, a seguir, fazer a sua inserção na base.

- No arquivo **backend/app.js**, comece importando o schema, como mostra a Listagem 2.5.1.

Listagem 2.5.1

```
const express = require ('express');
const app = express();
const bodyParser = require ('body-parser');

const Cliente = require ('./models/cliente');

app.use (bodyParser.json());

...
```

- Uma vez importado o módulo, podemos criar novas instâncias do tipo Cliente. Desejamos fazer isso no método **post**, já que é ele que vai inserir uma instância na base. Veja a Listagem 2.5.2.

Listagem 2.5.2

```
app.post ('/api/clientes', (req, res, next) => {
  const cliente = new Cliente({
    nome: req.body.nome,
    fone: req.body.fone,
    email: req.body.email

  })
  console.log (cliente);
  res.status(201).json({mensagem: 'Cliente inserido'})
});
```

- Abra um segundo terminal e coloque o servidor responsável pelo Back End em execução com

npm run start:server

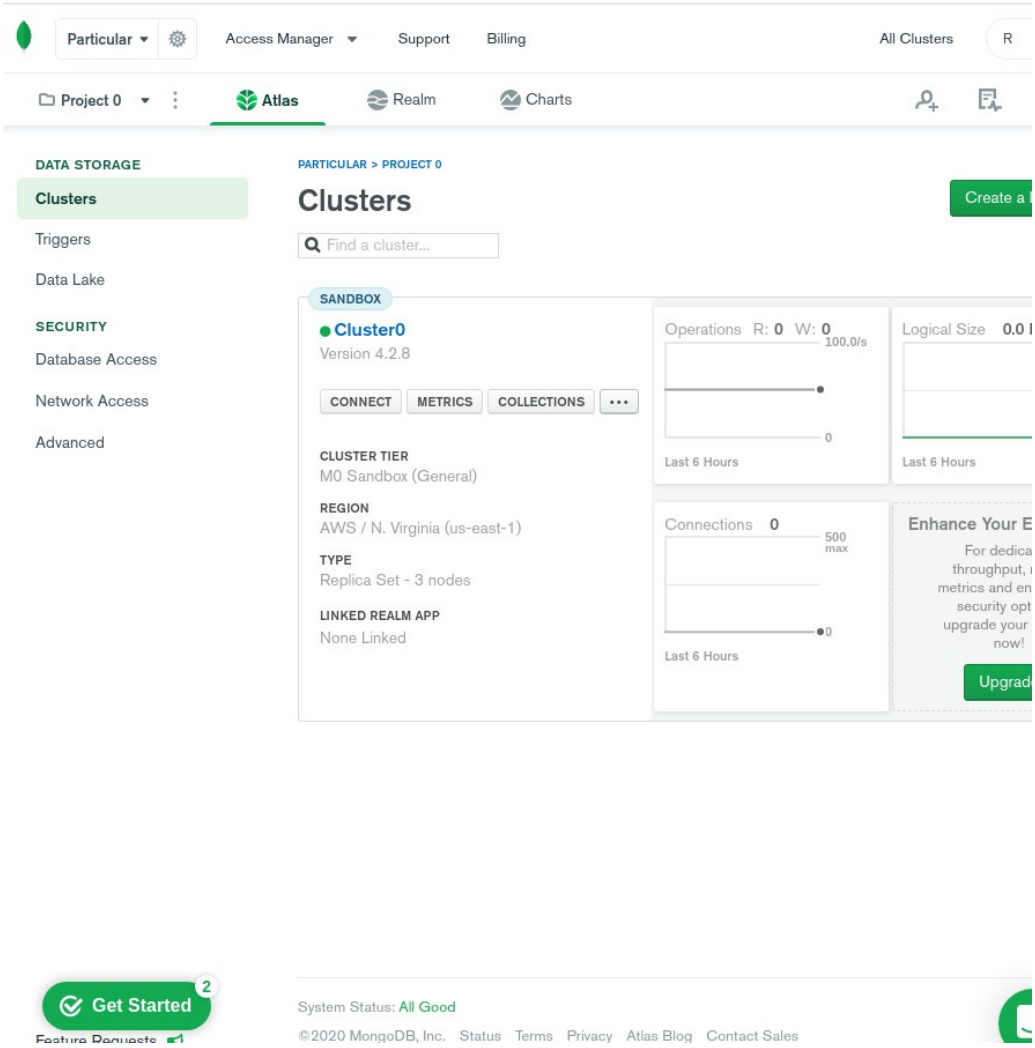
- Abra um terminal e coloque o servidor responsável pela aplicação Angular em execução com

ng serve --open

- Verique a saída do terminal responsável pela execução do Back End**.** Repare que o objeto Cliente criado possui um campo **_id**.

2.6 (Acessando o MongoDB Atlas a partir do Node) Neste momento, o cluster no MongoDB Atlas já deve ter sido alocado. Visite novamente a página dele, que deve ser parecida com o que exibe a Figura 2.6.1.

Figura 2.6.1



- Clique no botão **CONNECT**. Escolha, a seguir, **Connect your application**. Mantenha as opções **Node.js** e **3.6 or later** selecionadas.

- Note que essa página exibe a string de conexão. É por meio dela que conectaremos nossa aplicação Node com o MongoDB. Clique no botão **Copy** para copiá-la para o seu clipboard.

- Para fazer a conexão, abra o arquivo **backend/app.js** e, após importar o mongoose, execute seu método **connect**. Ele recebe a string de conexão copiada no Atlas. Note que é preciso chamar o método **connect depois** de o objeto **express** ter sido criado. Veja a Listagem 2.6.2.

Listagem 2.6.2

```
const express = require ('express');
const app = express();
const bodyParser = require ('body-parser');
const mongoose = require ('mongoose');
const Cliente = require ('./models/cliente');


mongoose.connect('sua string de conexão aqui');
app.use (bodyParser.json());
```

- Lembre-se de especificar corretamente seu usuário e senha.

- Note que o método **connect** devolve uma promise que nos permite verificar se a conexão foi realizada com sucesso ou não. Veja a Listagem 2.6.3.

Listagem 2.6.3

```
mongoose.connect('sua string de conexão aqui')
.then(() => {
  console.log ("Conexão OK")
}).catch(() => {
  console.log("Conexão NOK")
});
```

2.7 (Inserindo um cliente na base) Cada modelo criado pelo mongoose oferece alguns métodos que simplificam as operações de persistência.

- Para fazer a inserção de um cliente na base, utilizamos o método **save** do objeto cliente. Veja a Listagem 2.7.1.

Listagem 2.7.1

```
app.post ('/api/clientes', (req, res, next) => {
  const cliente = new Cliente({
    nome: req.body.nome,
    fone: req.body.fone,
    email: req.body.email

  })
  cliente.save();
  console.log (cliente);
  res.status(201).json({mensagem: 'Cliente inserido'})
});
```

- Note, também, que na sua string de conexão há uma região parecida com **<dbname>**. Ela pode ser substituída por um nome que faça sentido para nossa aplicação. A base de dados será criada automaticamente caso não exista no momento em que alguma operação CRUD for realizada pelo cliente. Altere <dbnode> para algo como **app-mean**.

- Utilize a sua aplicação Angular para fazer a inserção de um cliente.

2.8 (Visualizando os dados) Além de visualizar os dados diretamente no Atlas, podemos fazê-lo usando um cliente de linha de comando. Volte à página do Atlas e clique em **CONNECT** novamente. Faça o download do cliente apropriado para o seu sistema operacional e siga as instruções.

- Na linha de comando você deverá utilizar o comando **mongo** seguido da string de conexão, usando o exemplo fornecido na página do Atlas. Por exemplo:

mongo "mongodb+srv://cluster0.mongodb.net/**seubd**" --username **seuusuario**

Esse é só um exemplo. Copie o comando fornecido na página do Atlas.

- Precisamos especificar explicitamente qual banco de dados queremos utilizar. Isso pode ser feito com

use app-mean

Nota: O comando **help** exibe uma lista de comandos disponíveis.

- A seguir, execute os seguintes comandos

show dbs
show collections

- Note que há uma coleção chamada **clientes**. O objeto **db** faz referência ao banco atualmente selecionado (**app-mean**). Execute o seguinte comando para ver o cliente inserido.

db.clientes.find()

2.9 (Buscando dados) Até o momento, as requisições get enviadas ao servidor Node têm como resposta um vetor JSON de clientes que está fixo no código. Vamos ajustar a sua implementação para que os dados sejam trazidos diretamente da base gerenciada pelo MongoDB.

- Para a busca, usaremos o método estático **find** do modelo **Cliente**. Ele devolve uma promise por meio da qual podemos acessar a coleção de documentos. Veja a Listagem 2.9.1.

Listagem 2.9.1

```
app.get('/api/clientes', (req, res, next) => {
  Cliente.find().then(documents => {
    res.status(200).json({
      mensagem: "Tudo OK",
      clientes: documents
    });
  })
});
```

Referências

Angular. 2020. Disponível em <https://angular.io>. Acesso em agosto de 2020.

Angular Material UI component library. 2020. Disponível em <https://material.angular.io>. Acesso em agosto de 2020

Express - Node.js web application framework. 2020. Disponível em <https://expressjs.com>. Acesso em agosto de 2020.

Node.js. 2020. Disponível em <https://nodejs.org/en/>. Acesso em agosto de 2020.

The most popular database for modern apps | MongoDB. 2020. Disponível em <https://www.mongodb.com>. Acesso em agosto de 2020.