Objetivos:

I. Introdução ao MongoDB;
II. Mongoose;
III. Esquemas e modelos no Mongoose;
IV. Validações no Mongoose;
V. Conclusão.

i. Introdução ao MongoDB

O MongoDB é um BD, mas não segue o modelo tradicional de Sistemas de Gerenciamento de Banco de Dados Relacionais (SGBD-R). Ele pertence à categoria de BD NoSQL (Not Only SQL), que oferecem uma abordagem flexível para o armazenamento e a manipulação de dados. A principal diferença entre BD NoSQL, como o MongoDB, e os relacionais está na forma como os dados são armazenados e organizados. A seguir, destacamos algumas características importantes do MongoDB:

- Modelo de dados NoSQL: o MongoDB utiliza um modelo baseado em documentos, onde os dados são armazenados no formato BSON (Binary JSON), que é uma representação binária do JSON;
- Esquema dinâmico: diferentemente dos BD relacionais, o MongoDB permite esquemas dinâmicos, o que significa que os documentos dentro de uma mesma coleção podem conter campos diferentes, sem a necessidade de um esquema fixo, como acontece em tabelas de um SGBD-R;
- Consultas baseadas em documentos: as consultas no MongoDB são feitas utilizando uma sintaxe específica baseada nos documentos BSON, o que torna a interação com os dados mais natural e alinhada ao formato JSON. Já nos bancos relacionais, as consultas são realizadas com a linguagem SQL.

> Para instalar o MongoDB sugere-se fazer o download da versão Community (gratuita) https://www.mongodb.com/try/download/community.
> O vídeo https://www.youtube.com/watch?v=l4HeaNRi8f8 pode ajudar na instalação.
> O MongoBD Compass é uma interface gráfica de usuário (GUI), instalada juntamente com o MongoDB, que facilita as tarefas administrativas, tais como, visualizar dados, criar bancos e collections (coleções) e gerenciar permissões de usuários.

No MongoDB, os dados são organizados hierarquicamente nas seguintes estruturas:

- BD:
  - É a unidade mais alta de armazenamento no MongoDB, sendo análogo a um BD em um SGBD-R;
  - Um BD no MongoDB pode conter várias coleções e é totalmente independente de outros bancos.
- Coleção (collection):
  - Uma coleção é análoga a uma tabela em BD relacionais;

- É composta por um grupo de documentos, mas, diferentemente das tabelas relacionais, as coleções não exigem que todos os documentos sigam o mesmo esquema;
- Essa flexibilidade permite armazenar documentos com diferentes campos na mesma coleção, adaptando-se facilmente a mudanças nos requisitos do sistema.

- Documento (document):
  - Um documento é a unidade básica de dados no MongoDB e é representado no formato BSON.
  - Ele é análogo a uma linha (registro) em uma tabela relacional, mas, diferentemente dos registros relacionais, os documentos em uma coleção podem ter estruturas distintas;
  - Isso significa que não há necessidade de um esquema fixo, permitindo maior flexibilidade para modelar dados.

Comparação com SGBD-R:

Aspecto	MongoDB	SGBD-R
Estrutura de dados	Documentos (BSON)	Tabelas
Esquema	Dinâmico	Fixo
Unidade Básica de Armazenamento	Documento	Linha (Registro)
Linguagem de Consulta	Document Query Syntax (JSON-like)	SQL

ii. Mongoose

No MongoDB, um esquema refere-se à estrutura ou à definição de como os documentos em uma coleção específica podem ser organizados. Embora o MongoDB seja conhecido por sua flexibilidade e por não impor um esquema fixo, o uso de esquemas pode ser introduzido em níveis superiores, como na camada de aplicação. Isso significa que, mesmo utilizando o MongoDB, é possível definir regras e validações para os dados dentro do código da aplicação que interage com o banco.

O Mongoose é uma biblioteca de modelagem de objetos para o MongoDB, desenvolvida especificamente para aplicações Node.js. Ele oferece uma camada de abstração que simplifica a interação com o MongoDB, fornecendo diversas funcionalidades adicionais (https://mongoosejs.com).

Principais funcionalidades do Mongoose:

1. Definição de esquemas
   - O Mongoose nos permite definir esquemas de dados diretamente no código da aplicação. Esses esquemas atuam como modelos para os documentos armazenados no MongoDB;
2. Validação de dados
   - Os esquemas definidos no Mongoose podem incluir regras de validação, garantindo que os dados armazenados atendam aos requisitos da aplicação;
3. Middlewares

- O Mongoose suporta middlewares, que permitem executar ações antes ou depois de eventos, como salvar ou excluir documentos. Isso é útil para adicionar lógica personalizada à manipulação de dados.

4. Métodos e consultas:
   - Ele fornece métodos personalizados para manipular e consultar os dados de maneira mais estruturada.

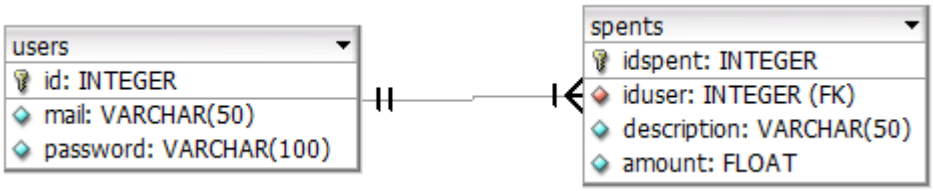
Diferença entre MongoDB e Mongoose:

Aspecto	MongoDB	Mongoose
Função	BD NoSQL que armazena dados	Biblioteca de abstração para interação com MongoDB
Estrutura	Dados armazenados sem esquema fixo	Permite a definição de esquemas na camada de aplicação
Validação	Não possui validação nativa de esquema	Suporte para validações avançadas nos esquemas
Middlewares	Não suportado	Suportado

Embora o Mongoose forneça funcionalidades similares às de um ORM (Object-Relational Mapping), ele é mais adequado ser classificado como um ODM (Object-Document Mapping), já que trabalha com documentos e não com tabelas relacionais. O Mongoose mapeia objetos JS para documentos MongoDB, permitindo que os desenvolvedores manipulem dados de forma mais intuitiva.

iii. Esquemas e modelos no Mongoose

Como exemplo, considere o seguinte modelo de dados relacional: um usuário pode ter N gastos. Esse relacionamento é representado no MongoDB usando esquemas e modelos do Mongoose.



Esquemas no Mongoose:

No Mongoose, cada esquema mapeia para uma coleção do MongoDB e define a estrutura dos documentos nessa coleção. A seguir estão os esquemas que representam as coleções `users` e `spents`.

Resumindo:

No Mongoose, um esquema serve para dizer como os dados vão ser guardados dentro do MongoDB.

Ou seja: ele define os campos (atributos) que cada documento vai ter em uma coleção.

Como exemplo, vamos criar dois esquemas — **Usuário e Gastos.**

```
import mongoose from "mongoose";
const { Schema } = mongoose;

// Esquema do Usuário
const UserSchema = new Schema({
  mail: { type: String, required: true },        // e-mail do usuário
  password: { type: String, required: true }     // senha do usuário
});

// Esquema dos Gastos
const SpentSchema = new Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },
                                                  // quem fez o gasto
  description: { type: String, required: true },  // descrição do gasto
  value: { type: Number, required: true }         // valor do gasto
});
```

O que cada parte faz:

- UserSchema → define que um usuário tem mail e password.
- SpentSchema → define que um gasto tem:
  - o usuário que fez o gasto,
  - uma descrição,
  - e o valor.

Criando documentos com o modelo:

Um modelo é uma classe que usamos para construir os documentos. Cada documento é um objeto com as propriedades e comportamentos definidos pelo esquema.

Exemplo de criação de um documento com o modelo `User`:

```
const document = new User({ mail:'a@teste.com', password:'abcdef' });
```

O conteúdo da variável `document` será o seguinte:

```
{
  mail: 'a@teste.com',
  password: 'abcdef',
  _id: new ObjectId('659acc6bd202c436bb835d2d')
}
```

O MongoDB, o campo `_id` é adicionado automaticamente como identificador único do documento. Esse identificador é do tipo `ObjetctId`, composto por 12 bytes e geralmente representado como uma string hexadecimal de 24 caracteres:

Salvando documentos no BD:

Para adicionar o documento à coleção correspondente no MongoDB, basta chamar o método `save`:

```
const resp = await doc.save();
```

O documento será adicionado à coleção `users`. Exemplo do conteúdo armazenado no MongoDB:

```
_id: ObjectId('659acc6bd202c436bb835d2d')
mail: "a@teste.com"
password: "abcdef"
__v: 0
```

O campo `__v` é gerado pelo Mongoose para controlar a versão dos documentos, facilitando a resolução de conflitos em atualizações concorrentes. Ele é incrementado automaticamente sempre que o documento é modificado.

Diferenças entre esquema, modelo e documento

Componente	Definição	Finalidade
Esquema	Estrutura que define os campos, tipos de dados e opções de validação para documentos em uma coleção.	Determina a estrutura e as regras de validação dos dados.
Modelo	Representação compilada de um esquema, usada para interagir com uma coleção específica no MongoDB.	Realiza operações CRUD (Create, Read, Update, Delete) na coleção.
Documento	Instância específica de um modelo, representando um registro na coleção.	São os dados reais armazenados no MongoDB, seguindo a estrutura definida pelo esquema.

iv. Validações no Mongoose

No Mongoose, as validações são definidas no esquema e aplicadas quando criamos ou atualizamos um documento usando um modelo. As validações podem ser built-in (prontas) ou personalizadas, conforme a necessidade.

Definindo validações no esquema:

As validações são configuradas diretamente na definição dos campos do esquema. Por exemplo, para um campo obrigatório, podemos usar `required: [true, "O e-mail é obrigatório"]`. Já para validações personalizadas, utilizamos a propriedade `validate`, que permite implementar lógicas específicas, como validar o formato de um e-mail.

v. Conclusão

Com base no conteúdo apresentado, pode-se concluir que o MongoDB representa uma alternativa moderna e flexível aos bancos de dados relacionais, oferecendo um modelo baseado em documentos BSON, com esquemas

dinâmicos e consultas alinhadas ao formato JSON, o que o torna especialmente adequado para aplicações que exigem escalabilidade e adaptação rápida a mudanças. O uso do Mongoose complementa essa abordagem ao fornecer uma camada de abstração na aplicação, permitindo a definição de esquemas, validações e regras de negócio que tornam a manipulação dos dados mais consistente e estruturada.

Além disso, a distinção entre esquema, modelo e documento evidencia a organização do processo de armazenamento e acesso às informações, enquanto os recursos de validação no Mongoose reforçam a confiabilidade dos dados. Assim, a integração entre MongoDB e Mongoose possibilita ao desenvolvedor unir a flexibilidade do NoSQL com a segurança e a padronização de dados exigidas em sistemas robustos, configurando-se como uma solução poderosa no desenvolvimento de aplicações modernas baseadas em Node.js.