

# Mongoose

# Conexión a Mongo

- Se utiliza una "connection string".
- Una connection string tiene un formato similar a:

```
mongodb://<direccion>/<base de datos>
```

- Para realizar la conexión a MongoDB con Mongoose, se usa:

```
mongoose.createConnection(<connection string>);
```

# Model & Schema

- Un model es como un constructor para una collection
- Un schema es lo que usa Mongoose para crear Mongo models.
- El model es finalmente lo que se usa para realizar operaciones sobre una collection

```
const Tank = mongoose.model("Tank", tankSchema);  
  
Tank.find({ size: "small" });
```

# Schema

- Tiene la siguiente información:
  - Campos de un document de la collection
  - Tipo de los campos (String, Number, etc)
  - Son requeridos o no
  - Métodos de validación
  - Y mas!
- Mongoose auto agrega el campo \_id
  - \_id es el identificador único predeterminado de un document
  - Tiene ese nombre para que esté de primero en orden alfabético
  - No es necesario “declararlo” en el schema

# Schema

```
const tankSchema = new Schema({  
  size: String,  
});
```

# Tipos de campos

- String
- Number
- Boolean
- ObjectId (como por ej. \_id)
- Schema (schema de Mongoose)
- Date
- Buffer (informacion binaria, como un archivo)
- Mixed (permite lo que sea, use a su propio riesgo)
- Array (se escribiría como [Number] para un array de numeros)
- Decimal128
- Map (una serie de keys y values, como un objeto de JavaScript)

# Opciones para todo tipo de campos

required	boolean o function
default	o function
validate	function
get	function
set	function
alias	string
immutable	boolean
transform	function
unique	boolean

# Opciones para todo tipo de campos

required	define si el campo es requerido
default	define un valor predeterminado para el campo
validate	define una funcion de validacion para el campo
get	define una funcion de 'get' para el campo
set	define una funcion de 'set' para el campo
alias	define otro nombre para el campo, para usar en get y set
immutable	define si el campo puede ser modificado despues de crear el document
transform	define como se muestra un document al volverlo string
unique	define si un campo puede tener valores repetidos en la collection



# Schema 2

```
const tankSchema = new Schema({  
  size: { type: String, required: true },  
});
```

# Ada Lovelace

| Considerada la primera programadora.

# Ada Lovelace

A

D

A

**A ctividad**

**D idactica de**

**A prendizaje**

# Para un Instagram,

Creemos un modelo/schema User

```
const UserSchema = new Schema({  
  // ??  
});
```

# Para un Instagram,

Creemos un modelo/schema Publicacion

```
const PostSchema = new Schema({  
  // ??  
});
```

# Opciones para Strings

lowercase	boolean
uppercase	boolean
trim	boolean
match	regex
enum	array
minLength	number
maxLength	number



# Opciones para Strings

lowercase	define si se pone la string en minusculas
uppercase	define si se pone la string en mayusculas
trim	define si la string se le hace trim (quita espacios iniciales y finales)
match	valida que la string concuerde con el regex
enum	valida que la string sea uno de los valores del array
minLength	valida que la string no sea mas corta que el valor dado
maxLength	valida que la string no sea mas larga que el valor dado

# Opciones para Numbers

min	number
max	number
enum	array

# Opciones para Numbers

min	valida que el numero no sea menor al valor dado
max	valida que el numero no sea mayor al valor dado
enum	valida que el numero sea uno de los valores del array

# Opciones para Date

min	Date
max	Date

# Opciones para Date

min	valida que la fecha no sea menor al valor dado
max	valida que la fecha no sea mayor al valor dado

# Queries

# Queries Multiples

- `Model.deleteMany` borra todos los documentos que apliquen al filter.
- `Model.find` retorna todos los documentos que apliquen al filter
- `Model.updateMany` modifica todos los documentos que apliquen al filter con los datos proveidos.

Query	Cantidad?	Modifica?
<code>Model.find()</code>	Multiple	No
<code>Model.deleteMany()</code>	Multiple	Si, elimina
<code>Model.updateMany()</code>	Multiple	Si, modifica

# Queries Singulares de Busqueda

- `Model.findOne()` retorna el primer document que aplique al filter.
- `Model.findById( id )` es igual a `Model.findOne( { _id: id } )`
- Esto tambien aplica para las queries siguientes.

Query	Cantidad?	Modifica?
<code>Model.findOne()</code>	Singular	No
<code>Model.findById()</code>	Singular	No



# Queries Singulares de Eliminacion

- Delete y Remove son usados como sinonimos
- Model.deleteOne() borra el primer documento que aplique al filter.
- Model.findOneAnd...() borra el primer documento que aplique al filter.

Query	Cantidad?	Modifica?
Model.deleteOne()	Singular	Si, elimina
Model.findOneAndDelete()	Singular	Si, elimina
Model.findOneAndRemove()	Singular	Si, elimina
Model.findByIdAndDelete()	Singular	Si, elimina
Model.findByIdAndRemove()	Singular	Si, elimina

# Queries Singulares de Modificación

- Delete y Remove son usados como sinonimos
- Model.deleteOne() borra el primer documento que aplique al filter.
- Model.findOneAnd...() borra el primer documento que aplique al filter.

Query	Cantidad?	Modifica?
Model.updateOne()	Singular	Si, modifica
Model.findByIdAndUpdate()	Singular	Si, modifica
Model.findOneAndUpdate()	Singular	Si, modifica

# Filter (Toda Query)

- Para toda query a Mongo, el primer parametro es el filter.

```
// busca todos los documentos
await MyModel.find();
await MyModel.find({});

// busca todos los documentos con nombre john y edad 18
await MyModel.find({ name: "john", age: 18 });
```

# Filter (Toda Query)

- Para toda query a Mongo, el primer parametro es el filter.

```
// borra el primer documento
await MyModel.findOneAndDelete();
await MyModel.findOneAndDelete({});

// borra el primer documento con nombre john y edad 18
await MyModel.findOneAndDelete({ name: "john", age: 18 });
```

# Projections (Queries Find)

- Para las queries tipo find, se puede agregar un segundo parametro, la "projection"

```
// retorna todos los modelos con nombre john, pero solo retorna los campos name y friends
await MyModel.find({ name: john }, "name friends");
await MyModel.find({ name: john }).select("name friends");
await MyModel.find({ name: john }).select(["name", "friends"]);
await MyModel.find({ name: john }).select({ name: 1, friends: 1 });
```

# Projections (Queries Find)

- Para las queries tipo find, se puede agregar un segundo parametro, la "projection"

```
// retorna todos los modelos con nombre john, pero no retorna password  
await MyModel.find({ name: john }, "-password");  
await MyModel.find({ name: john }).select("-password");  
await MyModel.find({ name: john }).select(["-password"]);  
await MyModel.find({ name: john }).select({ password: 0 });
```

# Update object (Queries Update)

- Para las queries de tipo update, el Segundo parametro es el "cambio"

```
// modifica el primer document, poniendole el nombre John
await MyModel.findOneAndUpdate({}, {name: 'John'})

// modifica el primer documento con nombre john y edad 18,
// poniendole el nombre John
await MyModel.findOneAndUpdate({ name: 'john', age: 18 }, {name: "John"})
```

# Metodos adicionales (Queries Find)

- El metodo skip se vuela los primeros n documentos de un find.

```
MyModel.find({ name: "john" }).skip(n);
```



# Metodos adicionales (Queries Find)

- El metodo sort organiza los resultados de un find segun un campo, y en el orden deseado.

```
// "asc", "ascending", 1  
// "desc", "descending", -1  
MyModel.find({ name: "john" }).sort({ age: -1 });
```

# Metodos adicionales (Queries Find)

- El metodo limit solo retorna los n primeros documentos

```
MyModel.find({ name: "john" }).limit(n);
```

# Paginacion?

# Paginacion?

```
await MyModel.find({ name: "john" }).sort({ age: 1 }).skip(100).limit(20);  
// una pagina tiene 20 documentos  
// estamos en la pagina 6
```

# Programacion Asincrona (await)

- Los queries a una base de datos son funciones asincronas, por esto:

```
const doc = await MyModel.find({ name: "john" }, "name friends");  
res.status(200).json(doc);
```

# Programacion Asincrona (.then)

- Los queries a una base de datos son funciones asincronas, por esto:

```
MyModel.find({ name: "john" }, "name friends") //  
  .then((doc) => {  
    res.status(200).json(doc);  
  });
```