Mongoose Validation p.1

Mise en place

Dans un terminal, tapez les trois commandes suivantes :

$ mkdir 📂 validation

$ cd validation

$ code .

Installation de modules

Nous devons initialiser un fichier package.json qui servira de support aux installations des différents modules.

Lancez dans votre éditeur un terminal à l'aide de la commande `crtl+ù`

🪛Lancez `npm init -y` pour créer le fichier  package.json.

Installez mongoose avec `npm i mongoose`

Choix d'une DB

Vous avez le choix de travailler en local ou avec le cloud Atlas.

En fonction de votre choix vous modifiez simplement la connexion dans vos différents fichiers.

🪛Créez un fichier  connect.js

 connect.js

```
1. // Import the mongoose module
2. const mongoose = require("mongoose");
```

```
3.
4. // Set `strictQuery: false` to globally opt into filtering by properties that
   aren't in the schema
5. // Included because it removes preparatory warnings for Mongoose 7.
6. // See:
   https://mongoosejs.com/docs/migrating_to_6.html#strictquery-is-remove
   d-and-replaced-by-strict
7. mongoose.set("strictQuery", false);
8.
9. // Define the database URL to connect to.
10. const mongoDB = "mongodb://127.0.0.1/validate";
11.
12. // Wait for database to connect, logging an error if there is a problem
13. main().catch((err) => console.log(err));
14.
15. async function main() {
16.   await mongoose.connect(mongoDB);
17.   console.log("connected to MongoDB");
18. }
```

Nous allons examiner différentes façons de valider un schéma.

Required

Créez un fichier  validate.js, qui reprend la structure du fichier 
connect.js

 validate.js

Mongoose Validation p.3

```
const mongoose = require("mongoose");

mongoose.set("strictQuery", false);

const mongoDB = "mongodb://127.0.0.1/validate";

main().catch((err) => console.log(err));

async function main() {
  await mongoose.connect(mongoDB);
  console.log("connected to MongoDB");
  const courses = await createCourse();
}

//defined a Schema
const courseSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
});

//compile the model
const Course = mongoose.model("Course", courseSchema);

async function createCourse() {

// create a course without a name
```

Mongoose Validation p.4

```
29.  const course = new Course();
30.
31.  try {
32.    const result = await course.save();
33.    console.log(result);
34.  } catch (exception) {
35.    console.log(exception.message);
36.  }
37.}
```

Lig. 29 : On teste la création d'un cours sans nom.

🪛Lancez la commande `node validate.js`

```
$ node validate.js
connected to MongoDB
Course validation failed: name: Path `name` is required.
```

Je vous laisse modifier la lig.29 avec le code suivant.

Lig. 29 `const course = new Course({ name: "HTML" });`

🪛Lancez la commande `node validate.js`

```
$ node validate.js
connected to MongoDB
{ name: 'HTML', _id: new ObjectId('654a0f16014afb75659c62d4'), __v: 0 }
```

Méthode validate

Vous pouvez appeler la méthode validate sur une instance de cours.

```
try {
  await course.validate();
  const result = await course.save();
  console.log(result);
} catch (exception) {
  console.log(exception.message);
}
```

Validation conditionnelle

🚀On aimerait vérifier la présence d'une donnée en fonction d'une autre !

Par exemple, si un cours est terminé, on demande un indice de satisfaction.

Créez un fichier conditionalValidation.js, qui reprend la structure du fichier conditionalValidation.js

```
1. const mongoose = require("mongoose");
2.
3. mongoose.set("strictQuery", false);
4.
```

```
5.  const mongoDB = "mongodb://127.0.0.1/validate";
6.
7.  main().catch((err) => console.log(err));
8.
9.  async function main() {
10.  await mongoose.connect(mongoDB);
11.  console.log("connected to MongoDB");
12.  const courses = await createCourse();
13.}
14.
15.const courseSchema = new mongoose.Schema({
16.  name: {
17.    type: String,
18.    required: true,
19.  },
20.  isFinished: Boolean,
21.  rate: {
22.    type: Number,
23.    required: function () {
24.      return this.isFinished;
25.    },
26.  },
27.});
28.
29.const Course = mongoose.model("Course", courseSchema);
30.
31.async function createCourse() {
32.  const course = new Course({ name: "HTML", isFinished: true });
```

```
33.
34.  try {
35.    const result = await course.save();
36.    console.log(result);
37.  } catch (exception) {
38.    console.log(exception.message);
39.  }
40.}
```

Notez la présence d'une fonction **non fléchée**[1] qui teste la valeur de isFinished.

```
21.  rate: {
22.    type: Number,
23.    required: function () {
24.      return this.isFinished;
25.    },
26.  },
```

Ainsi, si en ligne 32. vous définissez un cours ({ name: "HTML", isFinished: true }); qui est achevé, il faudra la présence de la propriété **rate**.

🪛Lancez la commande `node conditional Validation.js`

```
$ node conditionalValidation.js
connected to MongoDB
Course validation failed: rate: Path `rate` is required.
```

[1] Le this d'une fonction fléchée est lexical ! Cela ne conviendrait pas.

🪛Modifiez le code de la lig. 32 avec ({ name: "HTML", isFinished: true, rate:3})

Lancez la commande `node conditional Validation.js`

```
$ node conditionalValidation.js
connected to MongoDB
{
  name: 'HTML',
  isFinished: true,
  rate: 3,
  _id: new ObjectId('654a39ff348f8e7438eb68d0'),
  __v: 0
}
```

Validation intégrée

Il existe un grand nombre de validations intégrées :

Pour le type String :

- minLength,
- maxLength,
- match:/pattern/
- enum

Pour le type Number :

- min,
- max

Mongoose Validation p.9

Nous allons détailler la validation avec enum

enumModifiez dans le code de conditionalValidation.js le schéma avec

```
const courseSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  }
  level: {
    type: String,
    enum: ["L1", "L2", "L3"],
  },
});
```

🪛Testez le code avec la création d'un cours de niveau M1 en écrivant par exemple :

const course = new Course({ name: "HTML", **level: "M1"** });

Lancez la commande `node validate.js`

$ node validate.js

connected to MongoDB

Course validation failed: level: `M1` is not a valid enum value for path `level`.

Mongoose Validation p.10

Custom validator

🚀Nous voudrions imposer qu'un cours dispose forcément d'une étiquette.

💡L'idée première est de définir le schéma suivant

```
const courseSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  tags: {
    type: [String],
    required: true,
  },
});
```

😈 Un test avec const course = new Course({ name: "HTML" }); montre qu'un tableau vide est parfaitement valide.

```
$ node validate.js
connected to MongoDB
{
  name: 'HTML',
  tags: [],
```

Mongoose Validation p.11

```
  _id: new ObjectId('654a61474436fd25c4ae6eb0'),
  __v: 0
}
```

🪛Nous allons donc écrire notre propre validation à l'aide de la propriété validate qui est un objet avec une propriété **validator**.

```
tags: {
    type: Array,
    validate: {
      validator: function (v) {
        return v && v.length > 0;
      },
    },
  },
```

Créez un fichier  constumValidation.js, qui reprend la structure du fichier  constumValidation.js,

```
1. const mongoose = require("mongoose");
2.
3. mongoose.set("strictQuery", false);
4.
5. const mongoDB = "mongodb://127.0.0.1/validate";
6.
7. main().catch((err) => console.log(err));
```

```
8.
9.  async function main() {
10.  await mongoose.connect(mongoDB);
11.  console.log("connected to MongoDB");
12.  const courses = await createCourse();
13.}
14.
15.const courseSchema = new mongoose.Schema({
16.  name: {
17.    type: String,
18.    required: true,
19.  },
20.  tags: {
21.    type: Array,
22.    validate: {
23.      validator: function (v) {
24.        return v && v.length > 0;
25.      },
26.    },
27.  },
28.});
29.
30.const Course = mongoose.model("Course", courseSchema);
31.
32.async function createCourse() {
33.  const course = new Course({ name: "HTML" });
34.
35.  try {
```

```
    const result = await course.save();
    console.log(result);
  } catch (exception) {
    console.log(exception.message);
  }
}
```

Lancez la commande `node constumValidation.js`

```
$ node constumValidation.js
connected to MongoDB
Course validation failed: tags: Validator failed for path `tags` with value ``
```

🪛Ajoutez un message lors de la validation avec la propriété **message.**

```
  tags: {
    type: Array,
    validate: {
      validator: function (v) {
        return v && v.length > 0;
      },
      message: `Vous devez indiquer au moins une étiquette par exemple "Langage" `,
    },
  },
```

Mongoose Validation p.14

Lancez la commande `node constumValidation.js`

$ node constumValidation.js

connected to MongoDB

Course validation failed: tags: Vous devez indiquer au moins une étiquette par exemple "Langage"

Message d'erreur

🚀Supposons que nous ayons plusieurs messages d'erreur !

Nous pouvons dans un premier temps explorer les messages d'erreurs avec Object.entries.

🔪Créez un fichier messageValidation.js, qui reprend la structure du fichier messageValidation.js,

```
1. const mongoose = require("mongoose");
2.
3. mongoose.set("strictQuery", false);
4.
5. const mongoDB = "mongodb://127.0.0.1/validate";
6.
7. main().catch((err) => console.log(err));
8.
9. async function main() {
10.  await mongoose.connect(mongoDB);
```

```
11.  console.log("connected to MongoDB");
12.  const courses = await createCourse();
13.}
14.
15.const courseSchema = new mongoose.Schema({
16.  name: {
17.    type: String,
18.    required: true,
19.  },
20.  period: {
21.    type: String,
22.    enum: ["S1", "S2", "S3", "S4"],
23.    required: true,
24.  },
25.});
26.
27.const Course = mongoose.model("Course", courseSchema);
28.
29.async function createCourse() {
30.  const course = new Course();
31.  try {
32.    const result = await course.save();
33.    console.log(result);
34.  } catch (exception) {
35.    // for (field in exception.errors) console.log(exception.errors[field]);
36.    for ([k, v] of Object.entries(exception.errors)) {
37.      console.log(k, v);
38.    }
```

```
}
}
```

Lancez la commande `node messageValidation.js`

```
$ node messageValidation.js
…
```

Pour cibler les messages, nous pouvons écrire

```
async function createCourse() {
  const course = new Course();
  try {
    const result = await course.save();
    console.log(result);
  } catch (exception) {
    for (let v of Object.values(exception.errors)) {
      console.log(v.message);
    }
  }
}
```

Lancez la commande `node messageValidation.js`

```
$ node messageValidation.js
connected to MongoDB
Path `period` is required.
Path `name` is required.
```

Mongoose Validation p.17

Options

🚀Nous pouvons ajouter des options

```
1. const courseSchema = new mongoose.Schema({
2.   name: {
3.     type: String,
4.     required: true,
5.     lowercase: true,
6.   },
7. });
```

Ainsi dans le code const course = new Course({ name: "Html" }) ,nous aurons la valeur du nom en minuscule.

$ node optionsValidation.js

connected to MongoDB

{ name: 'html', _id: new ObjectId('654bb4e6245cee3c89be49a7'), __v: 0 }

Je vous laisse tester l'option

🔧uppercase:true,

🔧trim: true,

pour obtenir :

connected to MongoDB

Mongoose Validation p.18

{ name: 'HTML', _id: new ObjectId('654bb58f31fa12344b826d56'), __v: 0 }

Get/set

🪛Créez un fichier optionsValidation.js, qui reprend la structure du fichier optionsValidation.js,

```
1.  const mongoose = require("mongoose");
2.
3.  mongoose.set("strictQuery", false);
4.
5.  const mongoDB = "mongodb://127.0.0.1/validate";
6.
7.  main().catch((err) => console.log(err));
8.
9.  async function main() {
10.  await mongoose.connect(mongoDB);
11.  console.log("connected to MongoDB");
12.  await createCourse();
13.   //await getCourses();
14.}
15.
16.const courseSchema = new mongoose.Schema({
17.  ucts: {
18.    type: Number,
19.    get: (v) => Math.round(v),
20.    set: (v) => Math.round(v),
```

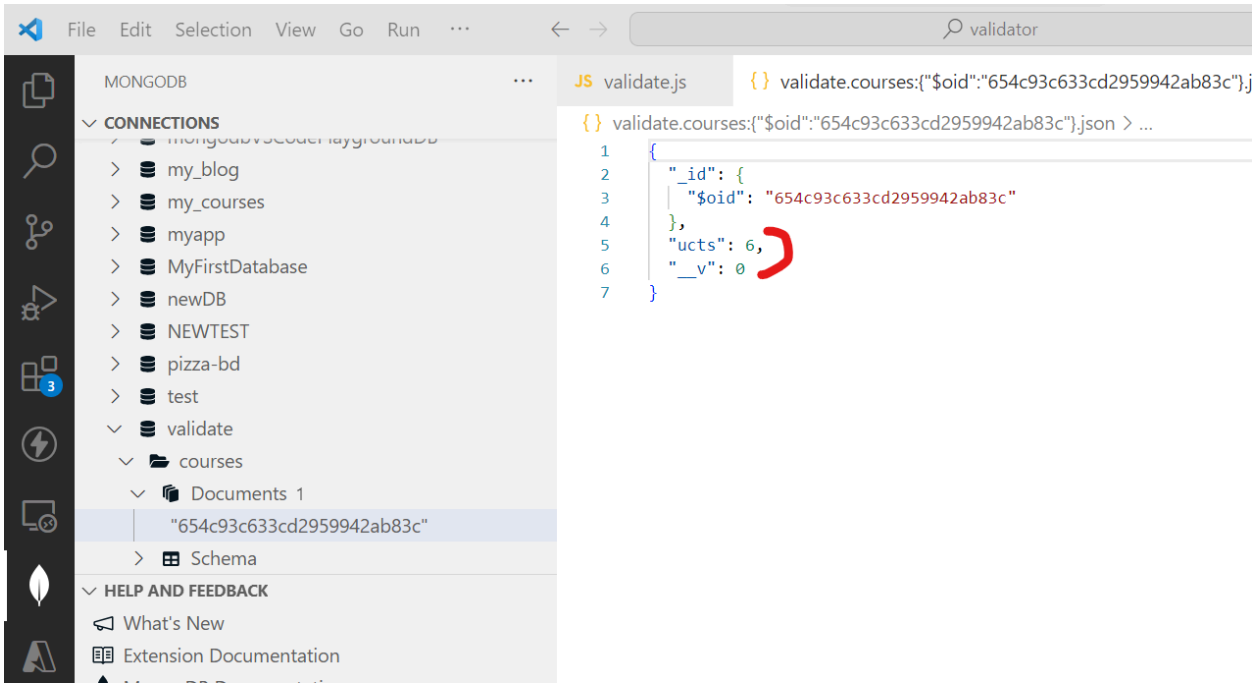
```
  },
});

const Course = mongoose.model("Course", courseSchema);

async function createCourse() {
  const course = new Course({ ucts: 6.2 });
  try {
    const result = await course.save();
    console.log(result);
  } catch (exception) {
    for (let v of Object.values(exception.errors)) {
      console.log(v.message);
    }
  }
}
```

Lancez la commande `node optionsValidation.js`

```
$ node optionsValidation.js
connected to MongoDB
{ ucts: 6, _id: new ObjectId('654c93c633cd2959942ab83c'), __v: 0 }
```

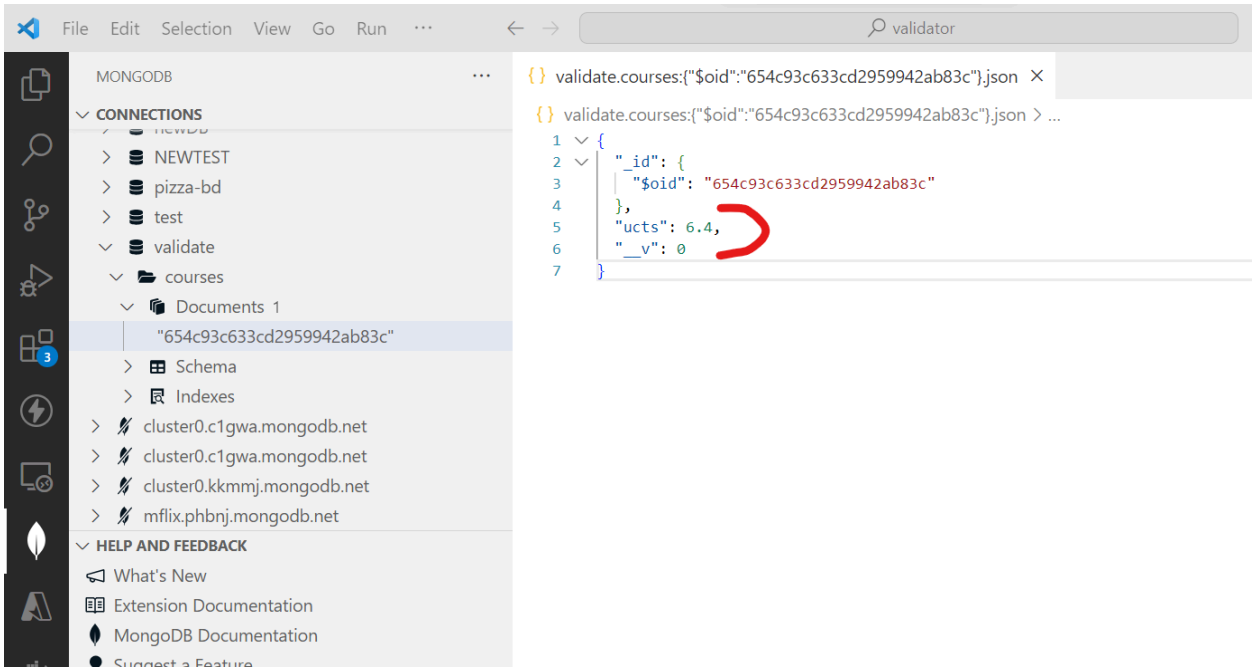
On peut vérifier directement dans la base que la valeur 6.2 est tronquée à 6.

Mongoose Validation p.20



🪛Utilisation du get

Modifiez avec à l'aide de Visual Studio la valeur de UCTS de l'objet.



🪛Modifiez le fichier optionsValidation.js avec le code incluant la fonction de recherche getCourses.

Mongoose Validation p.21

optionsValidation.js

```
1.  const mongoose = require("mongoose");
2.
3.  mongoose.set("strictQuery", false);
4.
5.  const mongoDB = "mongodb://127.0.0.1/validate";
6.
7.  main().catch((err) => console.log(err));
8.
9.  async function main() {
10.  await mongoose.connect(mongoDB);
11.  console.log("connected to MongoDB");
12.  // await createCourse();
13.  await getCourses();
14.}
15.
16.const courseSchema = new mongoose.Schema({
17.  ucts: {
18.    type: Number,
19.    get: (v) => Math.round(v),
20.    set: (v) => Math.round(v),
21.  },
22.});
23.
24.const Course = mongoose.model("Course", courseSchema);
25.
26.async function createCourse() {
27.  const course = new Course({ ucts: 6.2 });
```

```
28.  try {
29.    const result = await course.save();
30.    console.log(result);
31.  } catch (exception) {
32.    for (let v of Object.values(exception.errors)) {
33.      console.log(v.message);
34.    }
35.  }
36.}
37.
38.async function getCourses() {
39.  const c = await Course.find({});
40.  // utilisation du get
41.  console.log(c[0].ucts);
42.}
```

Lancez la commande node optionsValidation.js

$ node optionsValidation.js

connected to MongoDB

6

Asynchrone validator

La vérification peut être asynchrone lorsque l'on fait appel au réseau ou à une base de données. Voici une simulation de code .

Créez un fichier asynValidation.js, qui reprend la structure du fichier asynValidation.js,

```
1. const mongoose = require("mongoose");
2.
3. mongoose.set("strictQuery", false);
4.
5. const mongoDB = "mongodb://127.0.0.1/validate";
6.
7. main().catch((err) => console.log(err));
8.
```

```
async function main() {
  await mongoose.connect(mongoDB);
  console.log("connected to MongoDB");
  const courses = await createCourse();
}

async function checkTags(v) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(v && v.length > 0);
    }, 2000);
  });
}

const courseSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  tags: {
    required: true,
    type: Array,
    validate: {
      validator: async function (v) {
        return await checkTags(v);
      },
      message: `Vous devez indiquer au moins une étiquette par exemple
    "Langage" `,
```

```
    },
  },
});

const Course = mongoose.model("Course", courseSchema);

async function createCourse() {
  const course = new Course({ name: "HTML" });

  try {
    const result = await course.save();
    console.log(result);
  } catch (exception) {
    console.log(exception.message);
  }
}
```

Lig.15 : On simule une fonction asynchrone.