JavaScript runtime built on Chrome's V8 JavaScript engine

MongoDB

Installazioni

- Docker
  - https://www.docker.com
- Postman (opzionale)
  - https://www.postman.com/

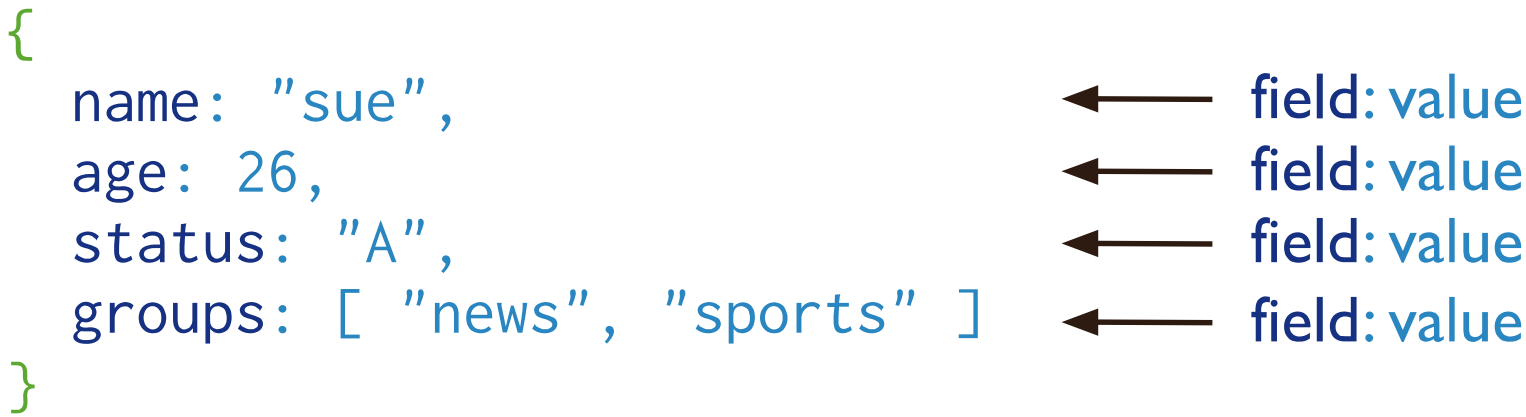
MongoDB

MongoDB è un database *document-oriented*, classificato come **NoSQL**.

- MongoDB gestisce uno o più **database**
- Un **database** contiene una o più **collezioni** (corrispettivo delle *tabelle*)
- Una **collezione** contiene **documenti** (corrispettivo dei *record*)
- Un **documento** contiene un **insieme di campi**

MongoDB

- Ogni **campo** è strutturato come una **coppia chiave-valore**
  - **chiave**: stringa di testo univoca all'interno del documento
  - **valore**: può essere semplice (stringa, numero, booleano) o complesso (oggetto, array, BLOB)



MongoDB

Un'installazione di MongoDB si compone, principalmente, di due programmi:

- `mongod`: eseguibile per avviare il demone del database
- `mongosh`: shell per l'interazione con il dabatase

È possibile eseguire MongoDB seguendo differenti approcci (e.g. scaricare i *sorgenti*, installare mediante *package manager*, *container*, *servizi cloud*, etc.)

MongoDB

Durante questa lezione utilizzeremo MongoDB tramite container Docker (https://hub.docker.com/_/mongo)

È possibile eseguire un'istanza di MongoDB con il seguente comando:

```
docker run --name my-db mongo
```

In un altro terminale, eseguire mongosh per interagire con il database:

```
docker exec -it my-db mongosh
```

MongoDB - Alcuni comandi

Stampa la lista dei database in MongoDB

```
show dbs
```

Seleziona il database su cui eseguire i comandi

```
use <db-name>
```

Crea una collezione chiamata `myColl`

```
db.createCollection("myColl")
```

Elenco delle collezioni disponibili nel database

```
db.getCollectionNames()
```

MongoDB - Alcuni comandi

Inserisce il documento nella collezione (crea la collezione qualora non ci sia)

```
db.myFantasticCollection.insertOne({ x: 1 })
```

Ricerca dei documenti (è possibile utilizzare le condizioni)

```
db.myFantasticCollection.find()
```

Aggiornamento di un documento

```
db.myFantasticCollection.deleteOne({ x: 1 })
```

Contare il numero di documenti in una collezione

```
db.myFantasticCollection.countDocuments()
```

Mongo Express

Mongo Express è un'interfaccia *web based* di gestione per MongoDB. Permette di agevolare l'esplorazione del database ed interagire con esso. Solitamente è utilizzato durante la fase di sviluppo.

Nella cartella `examples/example-01` è fornito un file docker compose minimale per l'esecuzione di MongoDB e un'istanza di Mongo Express.

```
cd examples/example-01
docker compose up
```

Visitare l'indirizzo http://localhost:8081

Node + Mongo

Per interagire con le API di MongoDB tramite Node.js è possibile utilizzare il driver ufficiale.

Questo approccio, di più **basso livello**, permette massima flessibilità.

```
1  const { MongoClient } = require('mongodb');
2
3  new MongoClient('mongodb://localhost:27017')
4      .connect()
5      .then(() => {
6          const database = client.db('myDB');
7          const collection = database.collection('myColl');
8          const document = { myKey: 'My Value' };
9          return collection.insertOne(document);
10     })
11     .then(result => {
12         console.log(`Document inserted ${result.insertedId}`);
13     })
14     .catch(error => {
15         console.error('Error connecting to MongoDB', error);
16     })
17     .finally(() => {
18         client.close();
19     });
```

examples/example-01/index.js

Mongoose

Mongoose è una libreria di ODM (Object Document Mapper) per MongoDB, che permette di gestire i dati con schemi definiti, semplificando le operazioni sul database.

Fornisce un'interfaccia per **creare**, **leggere**, **aggiornare** e **cancellare** documenti, rendendo l'utilizzo di MongoDB più strutturato e scalabile.

Gli schemi definiti per le collezioni permettono di **validare** i dati.

Mongoose

In Mongoose è sempre necessario uno **schema** che descriva la struttura dei documenti della collezione.

Nello schema va specificato il **tipo dei dati**, valori di default, esprimere l'obbligatorietà del dato, etc.

I tipi di dato disponibili sono: `String`, `Number`, `Date`, `Buffer`, `Boolean`, `Mixed`, `ObjectId`, `Array`, `Decimal128`, `Map`, `Schema`, `UUID` e `BigInt`.

Mongoose

Nell'utilizzo di Mongoose si possono identificare 3 passaggi principali, ricorrenti ad ogni utilizzo della libreria.

1. **Definizione** dello Schema
2. **Compilazione** da Schema a modello
3. **Utilizzo** del modello

```
1  const mongoose = require('mongoose');
2
3  const uri = 'mongodb://localhost:27017';
4
5  const mySchema = new mongoose.Schema({
6      myKey: String
7  });
8
9  mongoose.connect(uri, { dbName: 'myDB' })
10     .then(() => {
11         const MyModel = mongoose.model('myCollWithMongoose', mySchema);
12         const document = new MyModel({ myKey: 'My Value' });
13         return document.save();
14     })
15     .then(result => {
16         console.log(`Document inserted: ${result._id}`);
17     })
18     .catch(error => {
19         console.error('Error connecting to MongoDB', error);
20     })
21     .finally(() => {
22         mongoose.connection.close();
23     });
```

examples/example-02/index.js

Mongoose - Esempio di schema

```
const schema = new Schema({
    name: String,
    living: Boolean,
    updated: { type: Date, default: Date.now },
    age: { type: Number, min: 18, max: 65 },
    array: [],
    ofNumber: [Number],
    ofDates: [Date],
    ofArrayOfNumbers: [[Number]],
    nested: {
      stuff: { type: String, lowercase: true, trim: true }
    }
});
```

Mongoose - Query

È possibile realizzare le query in due varianti:

```
Person
  .find({
    occupation: 'host',
    'name.last': 'Ghost',
    age: { $gt: 17, $lt: 66 },
    likes: { $in: ['vaporizing', 'talking'] }
  })
  .limit(10)
  .sort({ occupation: -1 })
  .select({ name: 1, occupation: 1 })
  .exec();
```

JSON object

```
Person
  .find({ occupation: 'host' })
  .where('name.last').equals('Ghost')
  .where('age').gt(17).lt(66)
  .where('likes').in(['vaporizing', 'talking'])
  .limit(10)
  .sort('-occupation')
  .select('name occupation')
  .exec();
```

Query builder

Mongoose - Query

È possibile realizzare le query in due varianti:

```
Person
  .find({
    occupation: 'host',
    'name.last': 'Ghost',
    age: { $gt: 17, $lt: 66 },
    likes: { $in: ['vaporizing', 'talking'] }
  })
  .limit(10)
  .sort({ occupation: -1 })
  .select({ name: 1, occupation: 1 })
  .exec();
```

JSON object

```
Person
  .find({ occupation: 'host' })
  .where('name.last').equals('Ghost')
  .where('age').gt(17).lt(66)
  .where('likes').in(['vaporizing', 'talking'])
  .limit(10)
  .sort('-occupation')
  .select('name occupation')
  .exec();
```

Query builder

Mongoose - API

Di seguito sono riportate alcune API di Mongoose per l'interazione e la manipolazione dei dati nel database.

`model.find()`: legge tutti i dati dalla collezione

`new Model(data).save()`: crea un nuovo elemento e lo salva nel database

`model.findById(id)`: trova un elemento dato l'id

`model.findByIdAndUpdate(id, newContent)`: modifica l'elemento dato l'id

`model.findByIdAndDelete(id)`: elimina l'elemento dato l'id

Esercizio 1

Modellare, utilizzando gli **schemi di Mongoose**, il seguente dominio.

Nel sistema sono presenti 3 entità: `Utenti`, `Articoli` e `Commenti` agli articoli. Ogni utente ha un *nome* e una *mail*: Ogni articolo ha un *titolo*, *contenuto*, *voto* e *autore* (un utente del sistema). Ogni commento ha un *contenuto*, *articolo* a cui si riferisce e *utente* che lo ha realizzato.

Inoltre, realizzare degli endpoint che permettono l'inserimento e il recupero di queste entità.

Esercizio 1

- Nella cartella `exercises/exercise-01` è presente la struttura di partenza per l'esercizio
- Per testare le API realizzare aiutarsi con **Postman** (o altro client)
- Per facilitare l'osservazione del database utilizzare **Mongo Express**.
- Il file `index.js` contiene il web server node con express: implementare direttamente qua la logica delle rotte che utilizzano i modelli definiti.
- Nel file `models.js`, definire gli schemi e i modelli tramite Mongoose.

Esercizio 1

Nella cartella `exercises/exercise-01` eseguire i servizi per il database.

```
cd exercises/exercise-01
docker compose up
```

In un altro terminale installare le dipendenze del progetto

```
npm install
```

Eseguire il progetto

```
node index.js
```

Esercizio 2

- Creazione un **webserver** con Express che esponga delle **API** RESTful per gestire un set di film, salvati su MongoDB.
- N.B. Non verranno gestite tutte le casistiche di errore.
- Operazioni consentite dalle API:
  - Ottenere informazioni su tutti i film disponibili
  - Ottenere informazioni su un film particolare
  - Aggiungere un film alla collezione
  - Modificare un film
  - Cancellare un film

Esercizio 2

- In exercises/exercise-02/src è fornita la **struttura Express** vista nelle scorse esercitazioni.
  - **Routes** contiene un file js che gestisce tutte le rotte dell'applicazione
  - **Controllers** contiene un file js che gestisce la logica di ogni chiamata
  - **Models** contiene lo schema del database

Esercizio 2

- Gestire solo i casi in cui per lettura, modifica e cancellazione viene specificato un id esistente.
- Si assume che i dati inseriti siano sempre corretti.
- Testare le API con **Postman** (o altri strumenti)

Esercizio 2 - Database

- Spostarsi nella cartella `exercises/exercise-02/db`
- Eseguire i servizi docker: `docker compose up`

Nota: questo compose file permette di caricare anche un valore iniziale del database!

Esercizio 2 - Il server

Con un altro terminale, spostarsi dentro la cartella `exercises/exercise-02`

```
npm init
node index.js
```

Lo schema

- Nella cartella Models
  - Lavorare sul file **moviesModel.js**
  - Implementare un **modulo** che:
    - Utilizzi tale schema per generare il **modello** Mongoose
    - **Esporti** il modello

Controllers

Nel file `moviesControllers.js` implementare la logica dei metodi associati alle rotte

Connessione al DB

- Nel file **index**.js, è importato mongoose per eseguire la connessione al DB

```
const mongoose = require('mongoose’)
mongoose.connect('mongodb://localhost:27017/dbMovies);
```

Esercizio 3

- Aggiungere una rotta che riceva una query in GET, contenente un attore e due anni come parametri.
  - Restituire tutti i film in cui compare l'attore e che siano stati pubblicati nell'intervallo di tempo definito dalle due date.