Basic CRUD

MongoDB, ORM, Mongoose and CRUD Operations

SoftUni Team

Technical Trainers

Software University

SoftUni Foundation

Software University

http://softuni.bg

Table of Content

Have a Question?

SoftUni Foundation

sli.do

#tech-fund

MongoDB Configurations

Install and Start MongoDB, GUI

MongoDB

- Free **open-source** cross-platform document-oriented program
- Uses **JSON**-like documents with schemata.
- Good for **e-commerce** product catalog, blogs, evolving data requirements
- **Loosely coupled** objectives – the design **may change** by over time.

Developer Tools

- **Robo 3T**
  - Fully featured IDE with embedded shell
  - Visual Query Builder
  - IntelliShell with Auto-Completion
- Alternatives (**NoSQLBooster**)
  - Shell-centric cross platform GUI
  - Fluent Query Builder

Install MongoDB

- Download from: https://www.mongodb.com/download-center
- When **installed**, MongoDB needs a **driver**
  - One to use with Node.js, .NET, Java, etc..
  - Install MongoDB **driver** for Node.js:

```
npm install mongodb -g
```

Configure MongoDB

- Additional configurations are **needed**:
  - Go to installation folder and **run** a command prompt as an **administrator**
  - Type the following command:

Usually in **C:\Program Files\MongoDB\Server\3.4\bin**

```
"path to mongod.exe" mongod --dbpath "path to store data"
```

  - Additional information at: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/

Run MongoDB as a Windows Service

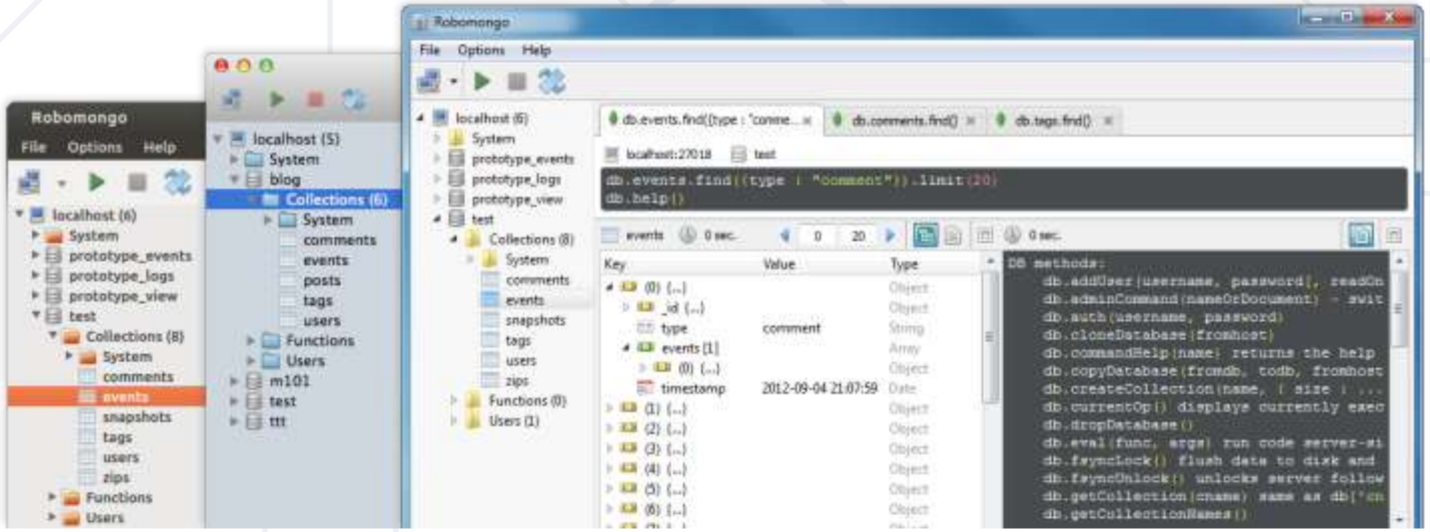
- Instead of **always** opening a CMD we can **run** MongoDB as a **service**

```
mongod --dbpath "C:\mymongodb" --logpath
"C:\mymongodb\logs.txt" --install --serviceName "MongoDB"
```

- After that just type **'net start MongoDB'** and the database now runs as a service
- Additional information: https://www.mkyong.com/mongodb/how-to-run-mongodb-as-windows-service/

Working with MongoDB GUI

- Choose one of the many
- For example:
  - Robo 3T -> https://robomongo.org/download
  - MongoBooster -> https://mongobooster.com/downloads



```
db.events.find({type : "comment"}).limit(20)
db.help()
```

```
(0) {...}                              Object
  _id {...}                            Object
  type          comment                String
  events [1]                           Array
    (0) {...}                          Object
  timestamp     2012-09-04 21:07:59    Date
(1) {...}                              Object
(2) {...}                              Object
(3) {...}                              Object
(4) {...}                              Object
(5) {...}                              Object
(6) {...}                              Object
```

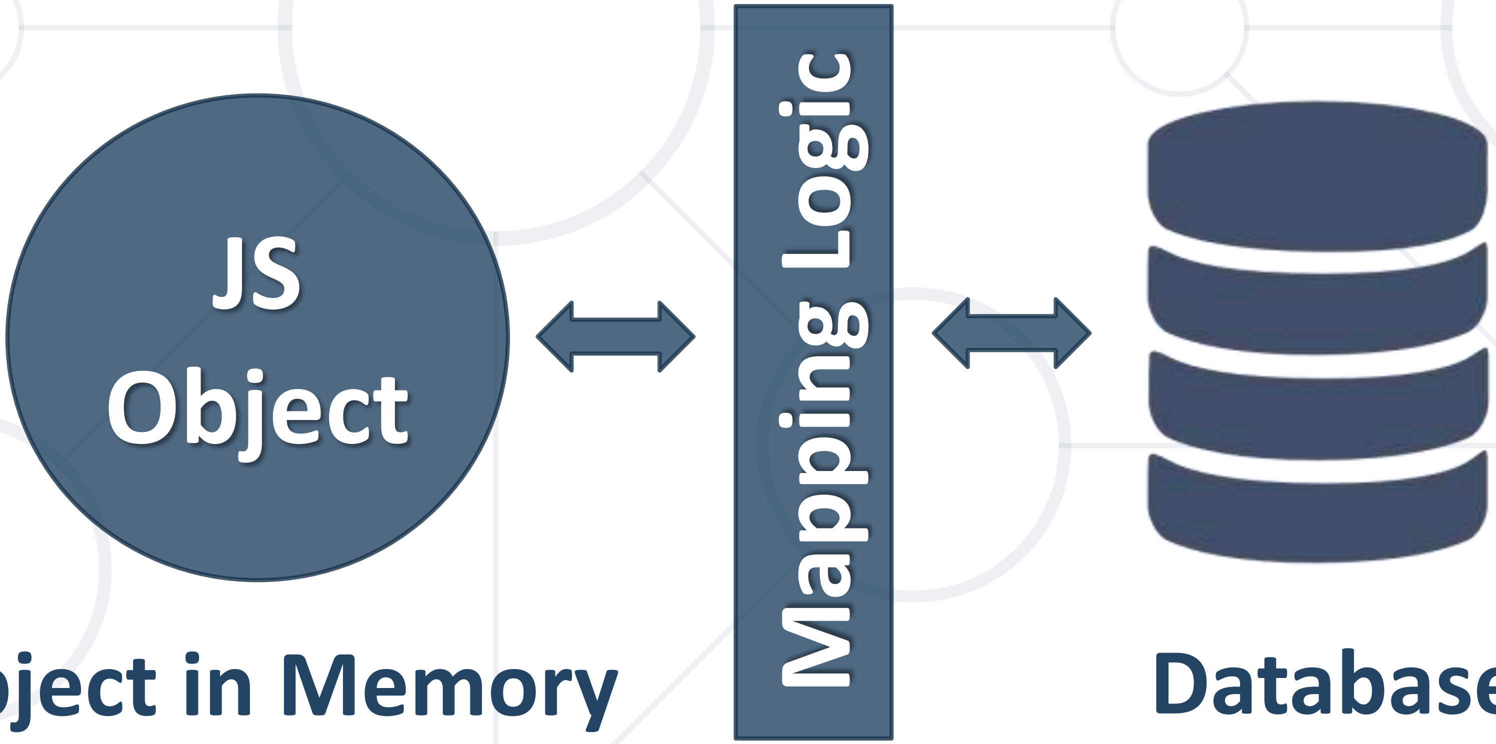
```
DB methods:
    db.addUser(username, password[, readOn
    db.adminCommand(nameOrDocument) - swit
    db.auth(username, password)
    db.cloneDatabase(fromhost)
    db.commandHelp(name) returns the help
    db.copyDatabase(fromdb, todb, fromhost
    db.createCollection(name, { size : ...
    db.currentOp() displays currently exec
    db.dropDatabase()
    db.eval(func, args) run code server-si
    db.fsyncLock() flush data to disk and
    db.fsyncUnlock() unlocks server follow
    db.getCollection(cname) same as db['cn
    db.getCollectionNames()
```

Object-Relational Mapping

Overview, Advantages and Disadvantages

ORM Overview

- **ORM Frameworks** maps OOP **classes** to database **tables**



Object in Memory

Database

ORM Advantages

- Developers can only focus on **business logic** rather than writing interfaces between code and db
- Reduces **development time** and **costs** by avoiding redundant codes
- Capable of connecting to **different databases**, which comes handy during switching from one db to the other
- Helps to **effectively query** from multiple tables

ORM Disadvantages

- Loss in developer **productivity** whilst they learn to program with ORM
- Developers lose understanding of what the code is actually doing - the developer is more in **control** using SQL
- ORM has a tendency to be **slow**
- ORM fails to compete against SQL queries for **complex** queries

Mongoose Introduction

Overview, Mongoose Schemas, Validation, Models

Mongoose Overview

- Mongoose is a object-document **model module** in Node.js for MongoDB
  - It **provides** a straight-forward, **schema-based** solution to **model** your application data.
  - Includes build-in type **casting** and **validation**
  - **Extends** the native **queries** (much **easier** to use)
  - To **install** type in CMD:

```
npm install mongoose --save
```

Working with Mongoose in Node.js

- Load the following module:

```
const mongoose = require('mongoose')
```

- Connecting to the database:

```
mongoose.connect('mongodb://localhost:27017/myapp')
```

Connect to the **database** using mongoose **module**

Mongoose Schemas

- Everything in Mongoose starts with a Schema.
- Each schema **maps** to a MongoDB **collection** and defines the **shape** of the documents within that collection.

```
const Schema = mongoose.Schema;
const studentSchema = new Schema({
  name: String,
  age: Number,
  grades: Array
});
```

Define Schema types. Each **entity property** could be validated.

Mongoose Validations

- Mongoose has **built-in** schema validations to protect from **invalid** data entity insertion

```
const studentSchema = new Schema({
  name: { type: String, required: true },
  age: { type: Number, min: 6, max: 18 },
  grades: [ { type: Number, min: 2, max: 6 } ]
})
```

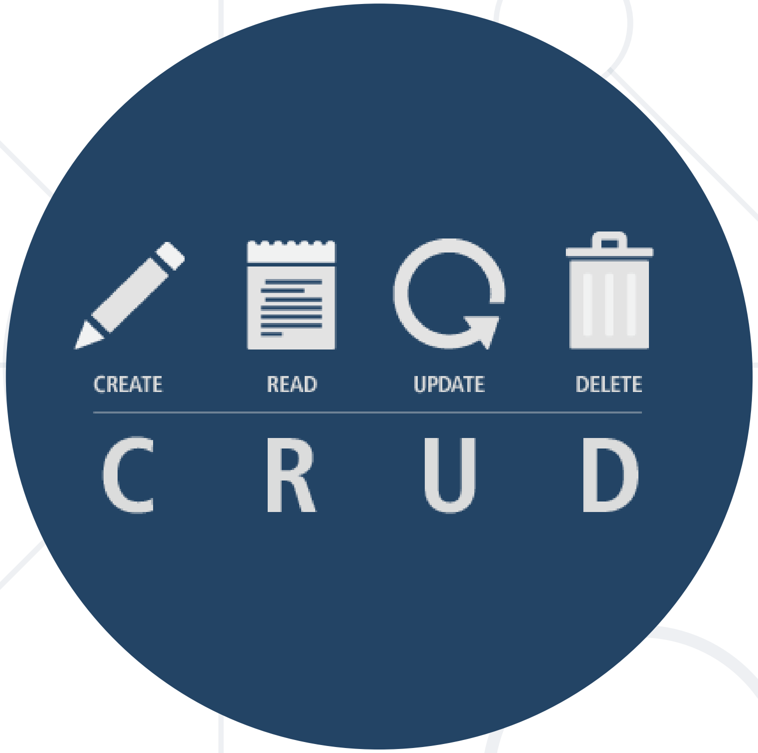
Mongoose Models

- Models are fancy **constructors** compiled from Schema definitions
- An **instance** of a model is called a **document**
- Models are responsible for **creating** and **reading** documents from the underlying MongoDB database

```
const Student = mongoose.model('Student', studentSchema);
```

Name of the **collection** your model is for

The Model Schema



Basic CRUD with MongoDB & Mongoose

Create, Read, Update, Delete

Create an Entity

- This is how we can **create** new items in the database
- This will commonly be from an **HTTP POST** request, although you can do this anywhere you want:

```
Student.create({
  name: 'George',
  age: 12,
  grades: [4, 5, 2]
})
.then((data) => console.log(data))
.catch((err) => console.error(err))
```

CRUD operations inside a database are **asynchronous**

List Entities

- To retrieve **all entities** from a collection use the following:

```
Student.find({})
.then((students) => console.log(students))
.catch((err) => console.error(err))
```

Will return an array with **all students** from the collection

- To fetch only **one student** by **id** use the following:

```
Student.findById(id)
.then((student) => console.log(student))
.catch((err) => console.error(err))
```

Will return a **single object** from the collection

List Entities (2)

- To **filter** by **given criteria** you can insert an **object** inside find:

```
Student.find({ name: 'George', age: 7 })
.then((students) => console.log(students))
.catch((err) => console.error(err))
```

Will return an **array** with **all students** that answer to the **given criteria**.

- To **filter** a **single object** by **given criteria** use the following:

```
Student.findOne({ name: 'George', age: 7 })
.then((student) => console.log(student))
.catch((err) => console.error(err))
```

Will return a single **object**. The **first entity** that matches the **given criteria**.

Update an Entity

- To update an entity we need the **entity id** and the **properties** you want to modify as an **object**. After that use the **findByIdAndUpdate** method:

```
Student.findByIdAndUpdate(id, { age: 13 })
  .then((student) => console.log(student))
  .catch((err) => console.error(err));
```

Will return the **old entity**. List them **again** to see the updated one.

Delete an Entity

- Deleting can be done by **id** and using the **findByIdAndRemove** method:

```
Student.findByIdAndRemove(id)
 .then((student) => console.log(student))
 .catch((err) => console.error(err));
```

Will return a single **object**. The entity that has been **deleted**.

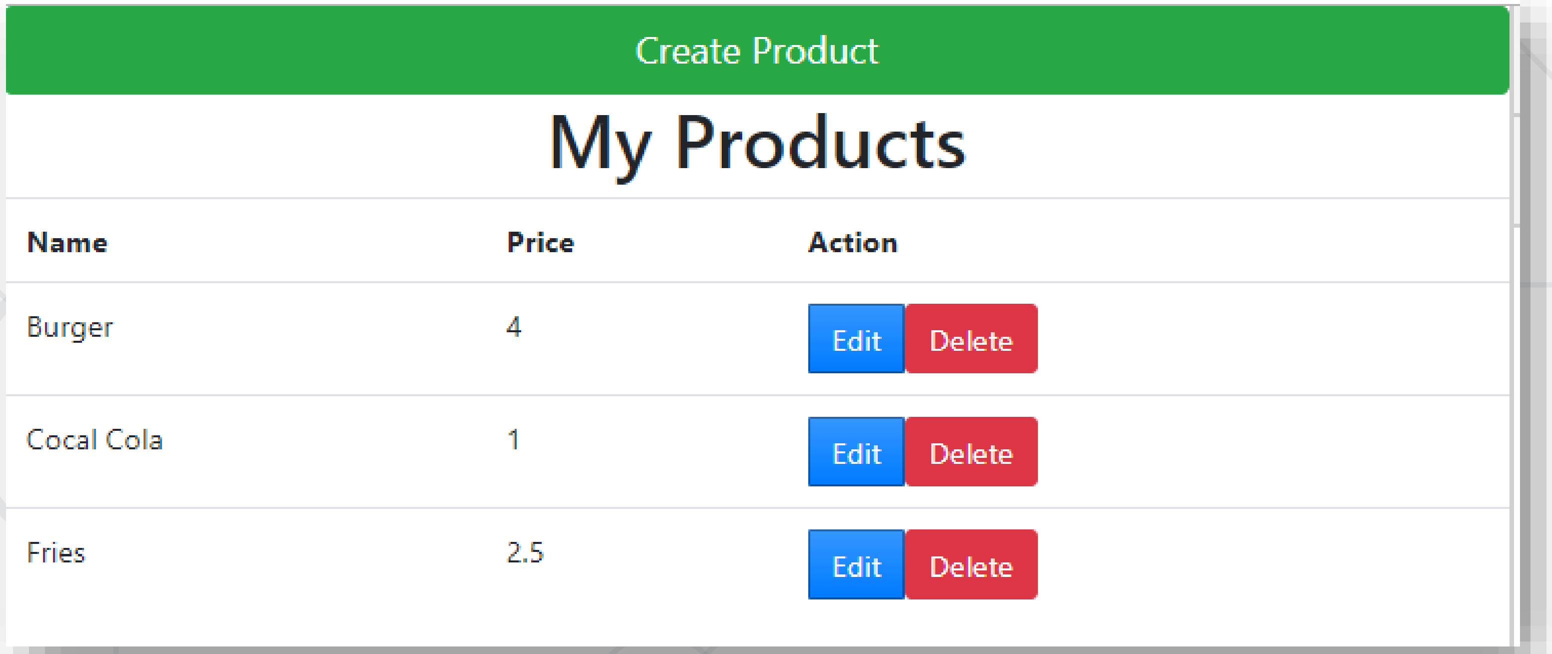
- To delete **many entities** by **criteria** use the following:

```
Student.deleteMany({ name: 'Rick' })
 .then((data) => console.log(data))
 .catch((err) => console.error(err));
```

Will return a single **object** with information **how many** entities have been deleted.

Problem: Simple Products Store

- Write an application that Creates, Lists, Edits and Deletes products
- Use Node.js, Express.js, MongoDB and Mongoose ORM



Create Product

My Products

Name	Price	Action
Burger	4	Edit Delete
Cocal Cola	1	Edit Delete
Fries	2.5	Edit Delete

Live Exercises

Summary

- ORM is used to **map objects** to **database tables**
- Mongoose is a object-document model **module** in Node.js for **MongoDB**
  - It uses **Schemas** & **Models** to connect with the database
- Basic CRUD operations are done using the **Mongoose models**

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech
we know sports

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

Postbank
Решения за твоето утре

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

INFRAGISTICS

LIEBHERR

æternity

codexio

SoftUni Organizational Partners

SoftUni Foundation

ИНФОРМАЦИОННО ОБСЛУЖВАНЕ

OneBit SOFTWARE

WORLD OF MYTHS

Lukanet.com

codexio

Trainings @ Software University (SoftUni)

SoftUni Foundation

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

SoftUni Foundation

License