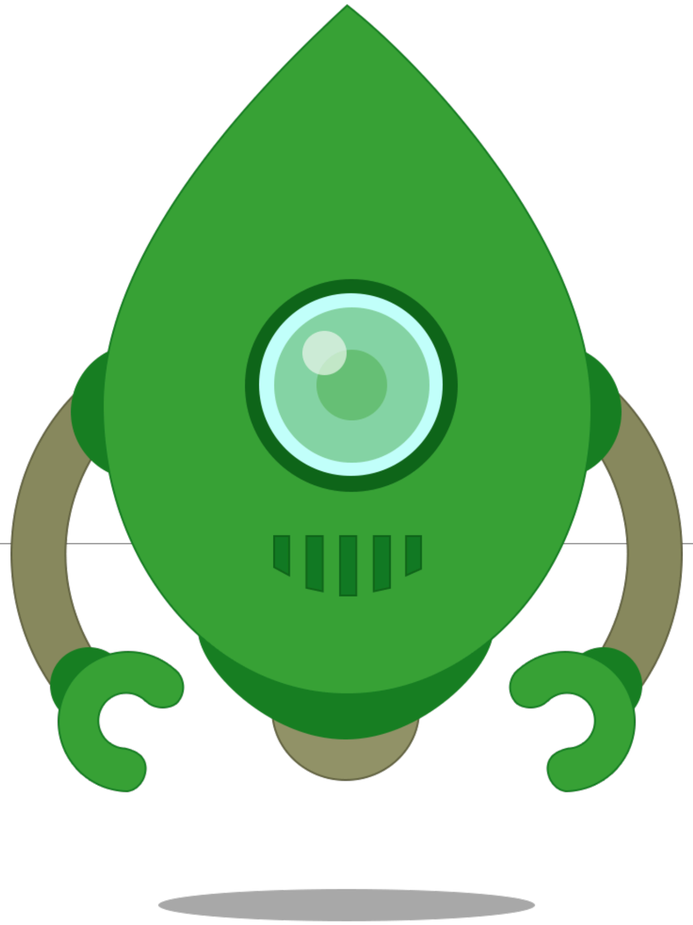
Accessing Mongo



index.js

```
...
require('./app/models/db');
...
```

.env

```
...
db=mongodb://localhost/donation
```

db.js

import .env

import mongoose

connect to the database service

Log success/fail/disconnect

```
'use strict';

require('dotenv').config();

const Mongoose = require('mongoose');

Mongoose.connect(process.env.db);
const db = Mongoose.connection;

db.on('error', function(err) {
  console.log(`database connection error: ${err}`);
});

db.on('disconnected', function() {
  console.log('database disconnected');
});

db.once('open', function() {
  console.log(`database connected to ${this.name} on ${this.host}`);
})
```

Mongo Core Concepts

- Database
- Documents
- Collections

mongoDB | Documentation

SERVER DRIVERS CLOUD TOOLS GUIDES Get MongoDB Search Documentation

Close ×

MONGODB MANUAL 4.0 (current)

Introduction
- Getting Started
- Create an Atlas Free Tier Cluster
- Databases and Collections
- Documents
- BSON Types
- Comparison/Sort Order
- MongoDB Extended JSON

Installation

The mongo Shell

MongoDB CRUD Operations

Aggregation

Data Models

Introduction to MongoDB

On this page

- Document Database
- Key Features

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.

Document Database

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

```
{
  name: "sue",          ← field: value
  age: 26,              ← field: value
  status: "A"           ← field: value
```

https://docs.mongodb.com/manual/introduction/

Databases

- A number of databases can be run on a single MongoDB server.
- Default database of MongoDB is 'db', which is stored within data folder.
- MongoDB can create databases on the fly. It is not required to create a database before you start working with it.

```
D:\mongodb\bin>mongo
MongoDB shell version: 1.8.1
connecting to: test
> show dbs
admin   (empty)
comedy  0.03125GB
local   (empty)
student 0.03125GB
test    0.03125GB
>
```

"show dbs" command provides you with a list of all the databases.

```
D:\mongodb\bin>mongo
MongoDB shell version: 1.8.1
connecting to: test
> db
test
>
```

Run 'db' command to refer to the current database object or connection.

```
> db
test
> use student
switched to db student
>
```

To connect to a particular database, run use command

Documents

- Document is the unit of storing data in a MongoDB database.
- Document use JSON (JavaScript Object Notation, is a lightweight, thoroughly explorable format used to interchange data between various applications) style for storing data.
- Often, the term "object" is used to refer a document.
- Documents are analogous to the records of a RDBMS. Insert, update and delete operations can be performed on a collection.

Example Document

```
{
    "_id" : ObjectId("527b3cc65ceafed9b2254a97"),
    "f_name" : "Lassy",
    "sex" : "Female",
    "class" : "VIII",
    "age" : 13,
    "grd_point" : 28.2514
}
```

Documents vs Tables

Relational DB	MongoDB
Table	Collection
Column	Key
Value	Value
Records / Rows	Document / Object

Data Types	Description
string	May be an empty string or a combination of characters.
integer	Digits.
boolean	Logical values True or False.
double	A type of floating point number.
null	Not zero, not empty.
array	A list of values.
object	An entity which can be used in programming. May be a value, variable, function, or data structure.
timestamp	A 64 bit value referring to a time and unique on a single "mongod" instance.
Object IDs	Every MongoDB object or document must have an Object ID which is unique. This is a BSON(Binary JavaScript Object Notation, which is the binary interpretation of JSON) object id, a 12-byte binary value which has a very rare chance of getting duplicated.

Collections

- A collection may store number of documents.
- A collection is analogous to a table of a RDBMS.
- A collection may store documents that are not same in structure.
- This is possible because MongoDB is a Schema-free database.
- In a relational database like MySQL, a schema defines the organization / structure of data in database.
- MongoDB does not require such a set of formula defining structure of data.

Mongoose Schema

- 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 Mongoose = require('mongoose');
const Schema = Mongoose.Schema;

const userSchema = new Schema({
  firstName: String,
  lastName: String,
  email: String,
  password: String
});
```

```
mongoose.Schema.Types.
```

```
String
Number
Date
Buffer
Boolean
Mixed
ObjectId
Array
```

Mongoose Models

- Models are constructors compiled from Schema definitions.
- Instances of these models represent documents which can be saved and retrieved from our database.
- All document creation and retrieval from the database is handled by these models.

users.js

```
const Mongoose = require('mongoose');
const Schema = Mongoose.Schema;

const userSchema = new Schema({
  firstName: String,
  lastName: String,
  email: String,
  password: String
});


module.exports = Mongoose.model('User', userSchema);
```

- User model can be used in other modules to interact with the “User” collection

Creating and saving Documents / Objects

import the Model

```
const User = require('../models/user');
```

Create a Document

```
...
const newUser = new User({
  firstName: payload.firstName,
  lastName: payload.lastName,
  email: payload.email,
  password: payload.password
});
```

Save the Document

```
user = await newUser.save();
```

Robomongo 0.9.0-RC7

- New Connection (2)
  - System
  - donation
    - Collections (1)
      - users
    - Functions
    - Users

db.getCollection('users').find({})

New Connection | localhost:27017 | donation

```
db.getCollection('users').find({})
```

users | 0.004 sec. | 0 | 50

Key	Value	Type
(1) ObjectId("5720b60b02b12...	{ 6 fields }	Object
_id	ObjectId("5720b60b02b126ae0a193f...	ObjectId
firstName	homer	String
lastName	simpson	String
email	homer@simpson.com	String
password	secret	String
__v	0	Int32

Logs

Find a Document (Object)

Mongo Query

No match found

Match Found, set cookie and let user in

```
let user = await User.findOne(email : email);

if(!user) {
  return h.redirect('/');
}


request.cookieAuth.set({ id: user.id })
return h.redirect('/home');
```

Update a Document (Object)

new/edited properties

key property

Mongo Query

Query succeeded, replace the fields

Save the new version

```
const userEdit = request.payload;

const id = request.auth.credentials.id;

const user = await User.findById(id);

if (user) {
  user.firstName = userEdit.firstName;
  user.lastName = userEdit.lastName;
  user.email = userEdit.email;
  user.password = userEdit.password;
  await user.save();
}
```

HAPI Handlers

- Create
- Read
- Update

Creating a Document in Handler

- Signup Handler

```
signup: {
  auth: false,
  handler: async function(request, h) {
    const payload = request.payload;
    const newUser = new User({
      firstName: payload.firstName,
      lastName: payload.lastName,
      email: payload.email,
      password: payload.password
    });
    const user = await newUser.save();
    request.cookieAuth.set({ id: user.id });
    return h.redirect('/home');
  }
},
```

Search for a Document in Handler

- login handler

```
login: {
  auth: false,
  handler: async function(request, h) {
    const { email, password } = request.payload
    let user = await User.findOne(email : email);
    if(!user) {
      return h.redirect('/');
    }
    request.cookieAuth.set({ id: user.id })
    return h.redirect('/home');
  }
},
```

Update a Document in Handler

- updateSettings handler

```
updateSettings: {
  handler: async function(request, h) {
    const userEdit = request.payload;
    const id = request.auth.credentials.id;
    const user = await User.findById(id);
    user.firstName = userEdit.firstName;
    user.lastName = userEdit.lastName;
    user.email = userEdit.email;
    user.password = userEdit.password;
    await user.save();
    return h.redirect('/settings');
  }
},
```