Accessing Mongo

Full Stack Web Development

import .env

import & configure mongoose

connect to the database service

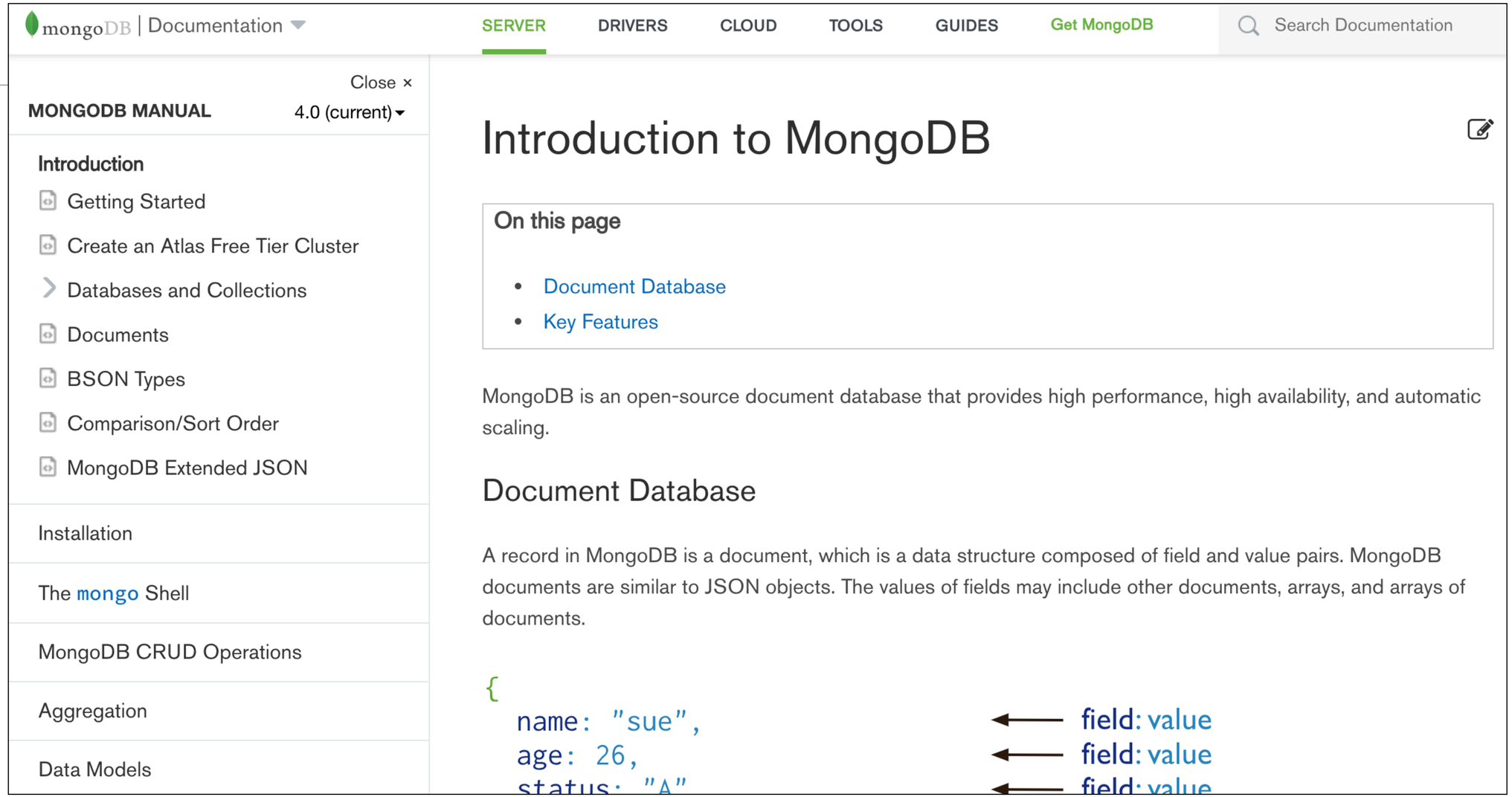
Log success/fail/ disconnect

connect.js

```
import * as dotenv from "dotenv";
import Mongoose from "mongoose";

export function connectMongo() {
  dotenv.config();

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

  db.on("error", (err) => {
    console.log(`database connection error: ${err}`);
  });

  db.on("disconnected", () => {
    console.log("database disconnected");
  });

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

Mongo Core Concepts

- Database
- Documents
- Collections



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.

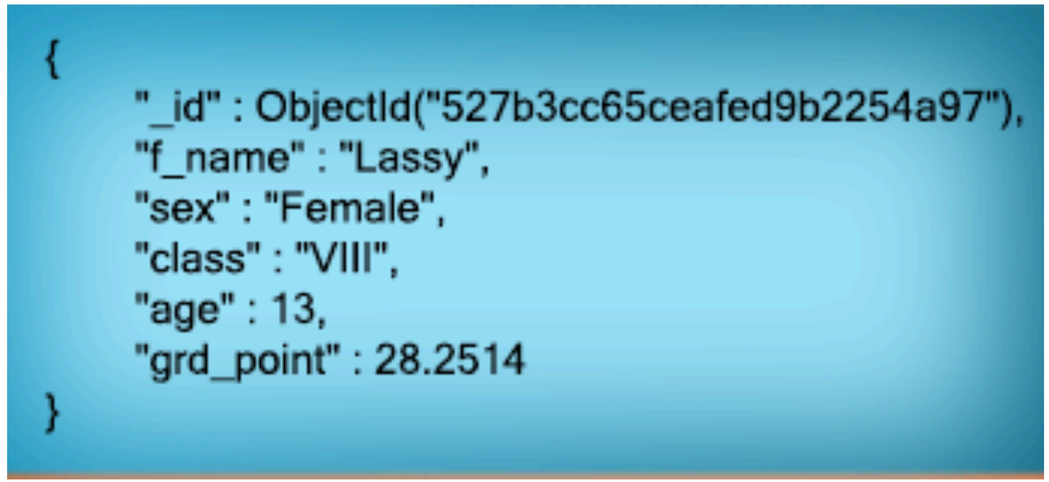
```
> db
test
> use student
switched to db student
>
```

- 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



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.

user.js

```
import Mongoose from "mongoose";
const { Schema } = Mongoose;

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.

user.js

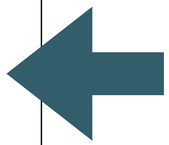
```
import Mongoose from "mongoose";
const { Schema } = Mongoose;

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

export const User = Mongoose.model("User", userSchema);
```

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

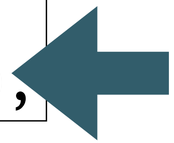
Creating and saving Documents / Objects

```
import Mongoose from "mongoose";
const { Schema } = Mongoose;

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

export const User = Mongoose.model("User", userSchema);
```

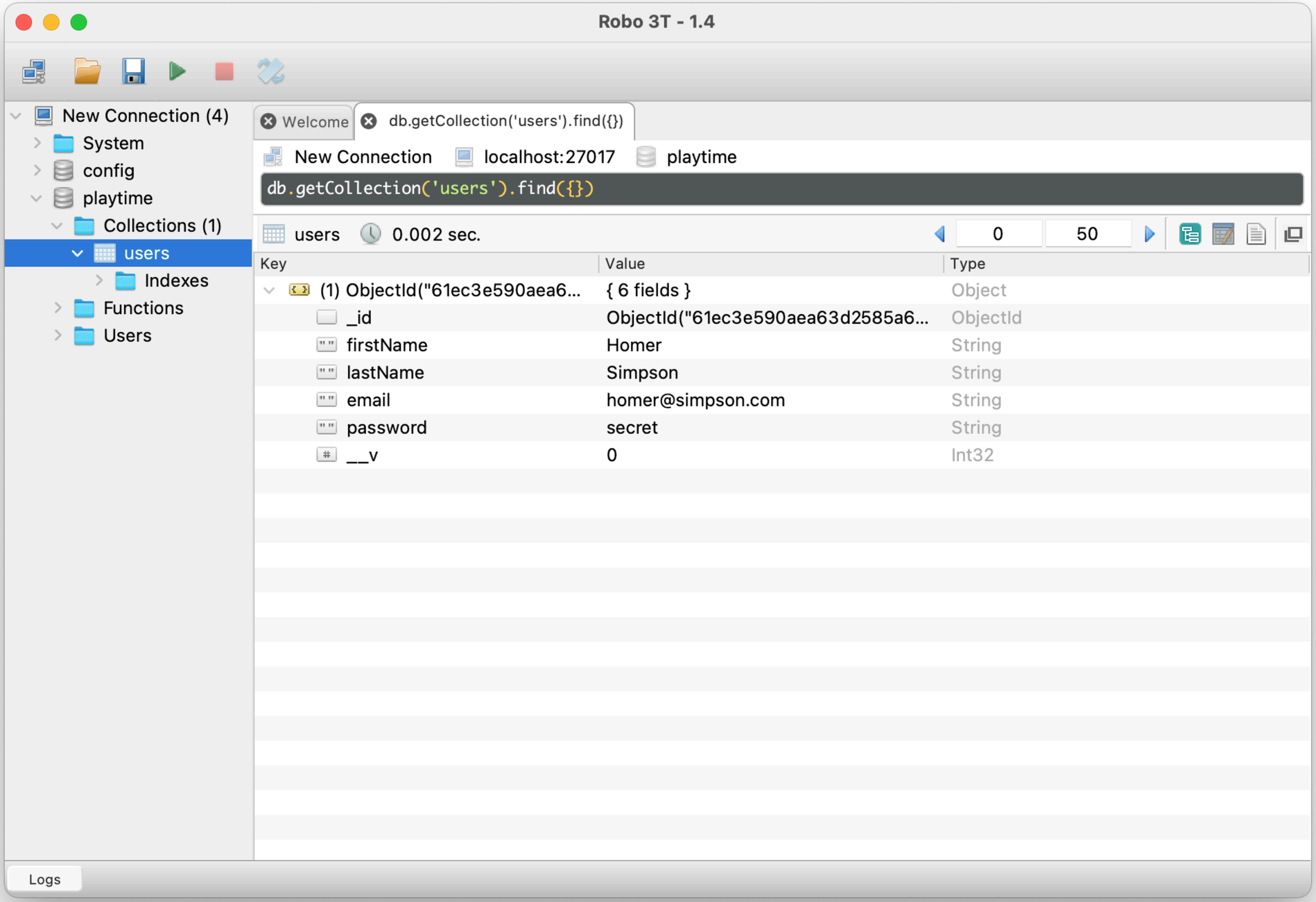
```
import { User } from “./user.js";


const newUser = new User({
  firstName: “Homer”,
  lastName: “Simpson”,
  email: “homer@simpson.com",
  password: “secret”
});



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



Create a Document



Save the Document


Robo 3T - 1.4
New Connection (4)
System
config
playtime
Collections (1)
users
Indexes
Functions
Users
Welcome
db.getCollection('users').find({})
New Connection
localhost:27017
playtime
db.getCollection('users').find({})
users
0.002 sec.
0
50
Key
Value
Type
(1) ObjectId("61ec3e590aea6...
{ 6 fields }
Object
_id
ObjectId("61ec3e590aea63d2585a6...
ObjectId
firstName
Homer
String
lastName
Simpson
String
email
homer@simpson.com
String
password
secret
String
__v
0
Int32
Logs

Find a Document (Object)

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

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

Mongo Query

Null => No match found

Update a Document (Object)

Mongo Query

Query succeeded, replace the fields

Save the new version

```
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();
}
```

Accessing Mongo

Full Stack Web Development