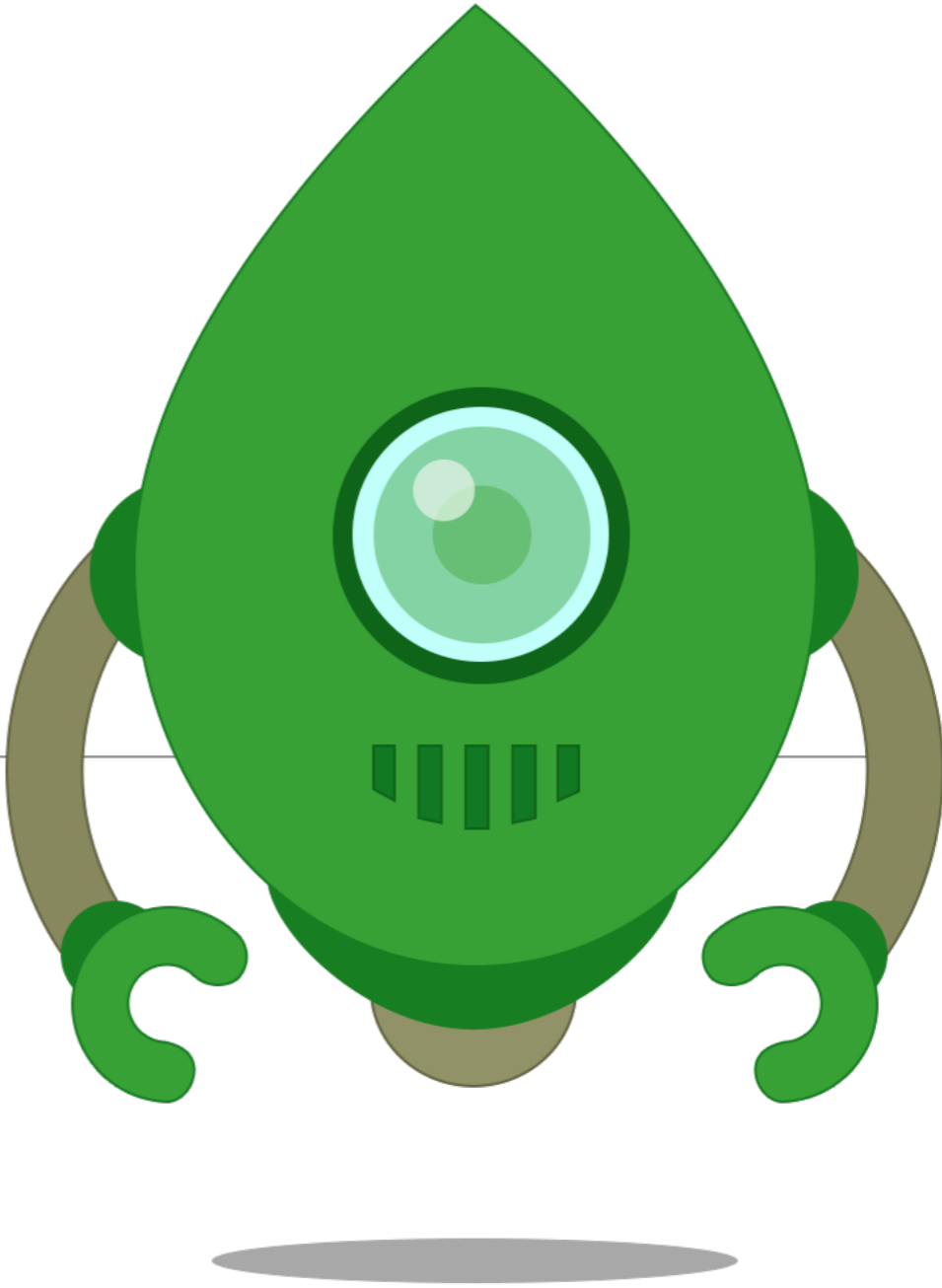


# Accessing Mongo

---



Connecting to  
Mongo (via  
Mongoose)

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

import mongoose

use the 'promise' library from node

declare the connection string

connect to the database

Log success/fail/disconnect

```
'use strict';

const mongoose = require('mongoose');
mongoose.Promise = global.Promise;

let dbURI = 'mongodb://localhost/donation';
if (process.env.NODE_ENV === 'production') {
  dbURI = process.env.MONGOLAB_URI;
}

mongoose.connect(dbURI);

mongoose.connection.on('connected', function () {
  console.log('Mongoose connected to ' + dbURI);
});

mongoose.connection.on('error', function (err) {
  console.log('Mongoose connection error: ' + err);
});

mongoose.connection.on('disconnected', function () {
  console.log('Mongoose disconnected');
});
```

# Mongo Core Concepts

---

- Database
- Documents
- Collections

Home  
Introduction to NoSQL  
Introduction to MongoDB  
Install MongoDB on Windows  
Install MongoDB on Linux  
Introduction to mongo Shell  
Data Types  
**Databases, Documents, Collections**  
MongoDB CONNECTIONS  
Query and Projection Operators  
Update Operators  
Aggregation Pipeline Operators  
MongoDB count() cursor method  
DATABASE COMMANDS  
MongoDB INSERT  
MongoDB UPDATE  
MongoDB DELETE  
MongoDB INDEX  
Shell Methods  
QUERIES  
BACKUP RESTORE  
**▼ MongoDB DRIVER**  
Java DRIVER  
Python DRIVER  
PHP DRIVER  
--More--

The logo for w3resource, featuring the text "w3resource" in a white, sans-serif font on a dark blue rectangular background.

# 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 those who 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 userSchema = mongoose.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

```
'use strict';  
  
const mongoose = require('mongoose');  
  
const userSchema = mongoose.Schema({  
  firstName: String,  
  lastName: String,  
  email: String,  
  password: String,  
});  
  
const User = mongoose.model('User', userSchema);  
module.exports = User;
```



- User object 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 user = new User({  
  firstName: 'Homer',  
  lastName: 'Simpson',  
  email: 'homer@simpson.com',  
  password: 'secret',  
});
```

Save the Document (Promises)

```
user.save().then(newUser => {  
  // user saved successfully  
  // newUser is the saved object  
}).catch(err => {  
  // an Error has occurred  
});
```

success

error

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

```
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

# Find a Document (Object)

---

One attribute  
we are searching  
on

DB Query

Query  
succeeded  
check if match  
found

error accessing  
DB

```
const userEmail = 'homer@simpson.com';

User.findOne({ email: userEmail }).then(foundUser => {
  // Query success, check foundUser to see if match

  if (foundUser) {
    // we found a match - complete document in foundUser
  } else {
    // no match found
  }

}).catch(err => {
  // some Error
});
```

# Update a Document (Object)

---

One attribute

Revised contents

DB Query

Query succeeded, replace the fields

Save the new version

New version saved

error accessing DB

```
const userEmail = 'homer@simpson.com';

const editedUser = //...new fields for user

User.findOne({ email: userEmail }).then(user => {
  // found the user, replace the fields in the document

  user.firstName = editedUser.firstName;
  user.lastName = editedUser.lastName;
  user.email = editedUser.email;
  user.password = editedUser.password;

  return user.save();
}).then(user => {
  // new version of the user saved
}).catch(err => {
  // Some error occurred
});
```

# HAPI Handlers

---

- Create
- Read
- Update

# Creating a Document in Handler

---

- Register HAPI Event Handler

```
exports.register = {
  auth: false,
  handler: function (request, reply) {
    const user = new User(request.payload);
    user.save().then(newUser => {
      reply.redirect('/login');
    }).catch(err => {
      reply.redirect('/');
    });
  },
};
```



# Search for a Document in Handler

---

- authenticate  
HAPI event  
handler

```
exports.authenticate = {
  auth: false,
  handler: function (request, reply) {
    const user = request.payload;
    User.findOne({ email: user.email }).then(foundUser => {
      if (foundUser && foundUser.password === user.password) {
        request.cookieAuth.set({
          loggedIn: true,
          loggedInUser: user.email,
        });
        reply.redirect('/home');
      } else {
        reply.redirect('/signup');
      }
    }).catch(err => {
      reply.redirect('/');
    });
  },
};
```

# Update a Document in Handler

- updateSettings  
HAPI event  
handler

```
exports.updateSettings = {  
  
  handler: function (request, reply) {  
    const editedUser = request.payload;  
    const loggedInUserEmail = request.auth.credentials.loggedInUser;  
  
    User.findOne({ email: loggedInUserEmail }).then(user => {  
      user.firstName = editedUser.firstName;  
      user.lastName = editedUser.lastName;  
      user.email = editedUser.email;  
      user.password = editedUser.password;  
      return user.save();  
    }).then(user => {  
      reply.view('settings', { title: 'Edit Account Settings', user: user });  
    }).catch(err => {  
      reply.redirect('/');  
    });  
  },  
  
};
```