What is MongoDB?

One of the most popular NoSQL DBMS
Why use MongoDB?

- Very efficient when we need to write a lot to the database.
- **Schemas** are very prone to changes (we’ll get to this!).
- Intuitive and natural mapping to object oriented languages.
- Relatively easy to use.
Structure

- MongoDB Instance
Structure

- MongoDB Instance
  - Database
Structure

- MongoDB Instance
  - Database
    - Collections
Structure

- MongoDB Instance
  - Database
    - Collections
  - Documents

Lots of documents in a bin (i.e. collection)
Structure

- MongoDB Instance
  - Database
  - Collections
  - Documents
  - Fields

- title
- paragraph
- conclusion
- signatures
Structure: Collection Example

- MongoDB Instance
  - Database
    - Collections
    - Documents
      - Fields

```json
[{
  "name": "Robert",
  "age": 12
},
{
  "name": "Rupayan",
  "age": 13
},
{
  "name": "Aashish",
  "age": 184
}]
```
Yes that’s cool, but how Do I Use Mongo?
How Do I Mongo with NodeJS?
mLab

- Allows you to connect to your database from any server.
- Is an FREE online service, so you need to sign up for it.
Mini-Workshop:

sign up for mLab

https://mlab.com/signup/
Create your account

Sign up for a free account to create fully managed cloud MongoDB databases. No credit card required.
### MongoDB Deployments

**Development and Utility**

Single-node deployments intended for environments that do not require high availability.

<table>
<thead>
<tr>
<th>NAME</th>
<th>PLAN</th>
<th>RAM</th>
<th>SIZE</th>
<th>SIZE ON DISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ds034807/rawr_db</td>
<td>Sandbox</td>
<td>shared</td>
<td>88.11 KB</td>
<td>80.00 MB</td>
</tr>
<tr>
<td>ds143900/rv_resources</td>
<td>Sandbox</td>
<td>shared</td>
<td>24.56 KB</td>
<td>16.00 MB</td>
</tr>
</tbody>
</table>

### Private Environments

None exist at this time. Click "Create new" to create an mLab Private Environment (VPC) on AWS.

---

[https://mlab.com/signup/](https://mlab.com/signup/)
mLab Database Example
List of collections

Manually add a collections

Collections

<table>
<thead>
<tr>
<th>NAME</th>
<th>DOCUMENTS</th>
<th>CAPPED?</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>images</td>
<td>0</td>
<td>false</td>
<td>7.98 KB</td>
</tr>
<tr>
<td>notifications</td>
<td>16</td>
<td>false</td>
<td>15.73 KB</td>
</tr>
<tr>
<td>requests</td>
<td>1</td>
<td>false</td>
<td>8.97 KB</td>
</tr>
<tr>
<td>travel_notices</td>
<td>23</td>
<td>false</td>
<td>30.63 KB</td>
</tr>
<tr>
<td>users</td>
<td>16</td>
<td>false</td>
<td>23.73 KB</td>
</tr>
</tbody>
</table>
Mini-Workshop:
Creating Your Catbook Database
Creating a new database

**MongoDB Deployments**

<table>
<thead>
<tr>
<th>NAME</th>
<th>PLAN</th>
<th>RAM</th>
<th>SIZE</th>
<th>SIZE ON DISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ds034807/rawr_db</td>
<td>Sandbox</td>
<td>shared</td>
<td>88.11 KB</td>
<td>80.00 MB</td>
</tr>
<tr>
<td>ds143900/rv_resources</td>
<td>Sandbox</td>
<td>shared</td>
<td>24.56 KB</td>
<td>16.00 MB</td>
</tr>
</tbody>
</table>

**Private Environments**

None exist at this time. Click "Create new" to create an mLab Private Environment (VPC) on AWS.
To connect using the mongo shell:

```bash
% mongo ds034807.mlab.com:34087/rawr_db -u <dbuser> -p <dbpassword>
```

To connect using a driver via the standard MongoDB URI (what's this?):

```
mongodb://<dbuser>:<dbpassword>@ds034807.mlab.com:34087/rawr_db
```

MongoDB URI
To connect using the mongo shell:

```
mongo ds034807.mlab.com:34807/rawr_db -u <dbuser> -p <dbpassword>
```

To connect using a driver via the standard MongoDB URI (what's this?):

```
mongodb://<dbuser>:<dbpassword>@ds034807.mlab.com:34807/rawr_db
```
MongoDB URI (Uniform Resource Identifier)

- The URI for your database uniquely identifies your resource (i.e. mongodb instance)
- Like a Storage Unit address with the Unit Number (referring to a specific Garage)
- Your URI should look like:

  `mongodb:<dbuser>:<dbpassword>@ds034807.mlab.com:34807/rawr_db`
MongoDB URI (Uniform Resource Identifier)

- The URI for your database uniquely identifies your resource (i.e. mongodb instance)
- Like a Storage Unit address with the Unit Number (referring to a specific Garage)
- Your URI should look like:

```
mongodb://<dbuser>:<dbpassword>@ds034807.mlab.com:34807/rawr_db
```

- credentials (allowed accessor): name and passcode
- Storage unit address
- Garage reference
Mini-Workshop: Add a User to Your Catbookdb

Remember your password!
MongoDB URI

Users of this database within this mongo instance

Add a database user
Add a Database User

Add new database user

- Database username*
- Database password*
- Confirm password*
- Make read-only

CANCEL  CREATE
Database: rawr_db

To connect using the mongo shell:

```
% mongo ds034807.mlab.com:34807/rawr_db -u <dbuser> -p <dbpassword>
```

To connect using a driver via the standard MongoDB URI (what's this?):

```
mongodb://<dbuser>:<dbpassword>@ds034807.mlab.com:34807/rawr_db
```

mongod version: 3.4.7 (MNAPv1)

⚠ Sandbox databases do not have redundancy and therefore are not suitable for production. Read our documentation on how to upgrade.

**Database Users**

<table>
<thead>
<tr>
<th>NAME</th>
<th>READ ONLY?</th>
</tr>
</thead>
<tbody>
<tr>
<td>rawr_db</td>
<td>false</td>
</tr>
</tbody>
</table>
Structure (AGAIN!)
Questions?
Mongoose
NodeJS library that allows MongoDB integration

How Do I Mongo with NodeJS?
How to Install Mongoose Into Your Project

In your terminal, run:

- `npm install --save mongoose`
- This will install `mongoose` in `./node_modules/` and saves to `package.json`
Workshop Warm-up:
Example Code Snippets with Mongoose

go.6148.io/mongo-snippets
Example File Structure

```javascript
root {
  package.json, .gitignore,
  src {
    models { student.js }
    routes { api.js },
    views { index.html, profile.html },
    app.js, db.js
  },
  public { css {}, js {}, ... }
}
```
Setting Up MongoDB with Mongoose

```javascript
// filepath: ./src/db.js
const mongoose = require("mongoose");
const mongoURL = "mongodb://robert:qwerty123@ds034807.mlab.com:34807/rawr_db";
const options = { useMongoClient: true };

// connects to MongoDB
mongoose.connect(mongoURL, options);
mongoose.Promise = global.Promise; // <- will explain what this is next week
const dbConnection = mongoose.connection;

// error handler
dbConnection.on('error', console.error.bind(console, 'connection error:'));
// optional: run when connection is successful
dbConnection.on('connected', function() {
    console.log('database connected');
});
```

View Code Snippet: [go.6148.io/mongo-snippets](go.6148.io/mongo-snippets)
Mongoose Schemas/Models: Processing Documents

- Means of structuring your Mongo documents
  - They specify the fields within a document
  - They allow adding stricter structures such as required fields or restrictions
- Each collection *should* have a schema
  - Best for organized collections
Mongoose Schema Example

```javascript
[
    {
        "name": "Robert",
        "age": 12
    },
    {
        "name": "Rupayan",
        "age": 13
    },
    ...
]
```

Schema:
```
    "name": String,
    "age": Number
```
Mongoose Schema types

String
Number
Date
Buffer
Boolean
Mixed
ObjectId
Array

Read more about schema types: http://mongoosejs.com/docs/schematypes.html
Mongoose Models

If (mongoose.Schema === skeleton)

then mongoose.model === body

Mongoose Models provide functionalities!
Creating a Mongoose Model

```javascript
64 // filepath: ./src/models/student.js
65 const mongoose = require('mongoose');
66 const student_schema = new mongoose.Schema({
67     name: String,
68     course: Number
69 });
70
71 const Student = mongoose.model("Student", student_schema);
72 // finally, export the model
73 module.exports = Student;
```

View Code Snippet: go.6148.io/mongo-snippets
Creating a Mongoose Model (In General)

- Create a mongoose.Schema first:
  ```javascript
  let db_schema = new mongoose.Schema({field:type, ..., field:type});
  ```
- Then create a mongoose.model:
  ```javascript
  let model = mongoose.model(model_name, model_schema);
  ```
- Make sure that the types are
  - `model_name:`String,
  - `model_schema:`mongoose.Schema
Structure (AGAIN!): Model === Fancier Collection
Creating a Mongoose Model With Constraints

```javascript
117 // filepath: ./src/models/student.js
118 const mongoose = require('mongoose');
119 const studentSchema = new mongoose.Schema({
120     name: {type: String, required: true, unique: true},
121     course: {type: Number, required: true, min: 1, max: 30}
122 });
123
124 const Student = mongoose.model("Student", student_schema);
125 // finally, export the model
126 module.exports = Student;
```

View Code Snippet: go.6148.io/mongo-snippets
Mongoose Parameters

http://mongoosejs.com/docs/schematypes.html (from “All Schema Types”)

More advanced: http://mongoosejs.com/docs/validation.html

More advanced: http://mongoosejs.com/docs/guide.html
Creating a Document with Mongoose

```javascript
134 // filepath: ./src/routes/api.js
135 const Student = require("../models/student.js");
136
137 const studentRobert = new Student({
138   'name': 'Robert',
139   'course': 6
140 });
141
142 studentRobert.save();
```
Finding One Document in Our Database

```javascript
// filepath: ./src/routes/api.js
const Student = require("../models/student.js");

Student.findOne({name: "Robert"}, function(err, studentRobert) {
  if (err) {
    // handle the error
    console.log("An error occurred: ", err.message);
  } else if (studentRobert === null) {
    // handle the case when no student in the database
    // matches the given parameters
    console.log(\"No student under name \"Robert\" found.\");
  } else {
    // this means we found the student under name \"Robert\"
    console.log(\"The student's course is ", studentRobert.course);
  }
});
```

View Code Snippet: [https://github.mit.edu/gist/roberv/006ff73653355008cc2e8c291c4798d9](https://github.mit.edu/gist/roberv/006ff73653355008cc2e8c291c4798d9)
Finding One Document in Our Database (Traditional)

```javascript
// filepath: ./src/routes/api.js
const Student = require("./models/student.js");
const id = "5a4ede410f9e59d70a8f18e9";

Student.findById(id, function(err, student_by_id) {
  if (err) {
    // handle the error
    console.log("An error occurred: ", err.message);
  } else if (studentRobert === null) {
    // handle the case when no student in the database
    // matches the given id
    console.log("No student under name with the given id found.");
  } else {
    // this means we found the student under name "Robert"
    console.log("The student's name is ", student_by_id.name);
    console.log("The student's course is ", student_by_id.course);
  }
});
```

View Code Snippet: go.6148.io/mongo-snippets
Finding MANY Documents in Our Database

```javascript
219 // filepath: ./src/routes/api.js
220 const Student = require("../models/student.js");
221
222 Student.find({name: "Robert"}, function(err, students_named_robert) {
223     if (err) {
224         // handle the error
225         console.log("An error occurred: ", err.message);
226     } else if (students_named_robert.length === 0) {
227         // handle the case when no student in the database
228         // matches the given parameters
229         console.log("No students under name "Robert" found.");
230     } else {
231         // this means we found "many" students under name "Robert"
232         console.log("The number of students named Robert is ", students_named_robert.length);
233     }
234 });
```

View Code Snippet: [go.6148.io/mongo-snippets](go.6148.io/mongo-snippets)
Finding MANY Documents in Our Database

In our previous example, keep in mind that you can add multiple parameters in your search. For example:

```javascript
Student.find({name: "Robert", course: 6}, callback);
```
Editing Existing Documents

```
255 // filepath: ./src/routes/api.js
256 const Student = require("../models/student.js");
257 // first, retrieve the document with student.findOne
258 // filepath: ./src/routes/api.js
259 const Student = require("../models/student.js");
260 // first, retrieve the document with student.findOne
261 Student.findOne({name: "Robert"}, function(err, studentRobert) {
  if (err) {
    // handle the error
    console.log("An error occurred: ", err.message);
  } else if (studentRobert === null) {
    // handler the case when no student in the database
    // matches the given parameters
    console.log("No students under name "Robert" found.");
  } else {
    // this means we found "many" students under name "Robert"
    // now we can edit "Robert"
    studentRobert.course = 21;
    // save the changes made to studentRobert
    studentRobert.save();
  }
});
```
Editing Existing Documents (Another Way)

THIS SLIDE WAS NOT SHOWN IN PRESENTATION. IT’S A MORE ADVANCED WAY OF UPDATING.

```javascript
289  // filepath: ./src/routes/api.js
290  const Student = require("../models/student.js");
291  // first, retrieve the documents and update them
292  Student.update({name: "Robert"}, {$set: {course: 21}}, function(err, results) {
293     if (err) {
294         // handle the error
295         console.log("An error occurred: ", err.message);
296     } else {
297         // do something with the results if you wish
298         // usually looks like { n: 1, nModified: 0, ok: 1 }
299         console.log(results);
300     }
301 });
```

View Code Snippet: go.6148.io/mongo-snippets
Removing Existing Documents

```javascript
// filepath: ./src/routes/api.js
const Student = require("./models/student.js");

Student.remove({name: "Robert"}, function(err, results) {
    if (err) {
        // handle the error
        console.log("An error occurred", err.message);
    } else {
        // do something with results if you want
        // usually looks like { n: 1, nModified: 0, ok: 1 }
        console.log(results);
    }
});
```

View Code Snippet: [go.6148.io/mongo-snippets](go.6148.io/mongo-snippets)
Workshop: Hook Database to Your Catbook App
Workshop Plan

- Quickly review our repository
- Hook back-end server up with mongo database
- Gear up front-end to use implemented back-end database functionalities: save comments and stories to our own mongo database
Clone the codebase:
go.6148.io/workshop3
STEP 1:
Connect Your App to MongoDB with Mongoose
STEP 1 SETUP:

- git checkout step0
- npm install
- npm install --save mongoose
File Structure After Changes

catbook-workshop3 {
    package.json, .gitignore,
    src {
        models { comment.js, story.js },
        routes { views.js, api.js },
        views { index.html, profile.html },
        app.js, db.js
    },
    public { css {}, js[5]{ ..., feed.js, ... }, ... }
}
Let’s Look at app.js

- Open app.js:
  - Where did our routes (‘/’, ‘/u/profile’) go?
  - `app.use(‘/’, views)` creates a “guide” for the endpoint ‘/’ and uses definitions from `view.js`.

- Open views.js:
  - Contains our previous routes :)

- What’s this 404 business?

- What about “next(err)”?

- Finally, what’s this error handler stuff?
Connect Your App to Your Mongo DBMS

Use Mongoose to Connect to your database in `db.js`:

Enter your mongodbURI from mlab where it says to do it in the comments.
Connect Your App to Your Mongo DBMS ("solution")

Use Mongoose to Connect to your database in `db.js`. It should look like:

```javascript
const mongoURL = 
"mongodb://robertv:qwerty123@ds251287.mlab.com:51287/catbookdb";
```

(in one line!)
Run It From Your Root Directory

- npm start
Run It From Your Root Directory

- npm start
- IT DOES NOT LOG ANYTHING...
Our App Isn’t Aware of db.js (Yet)

That’s because our app isn’t aware of db.js yet. We need to fix this in app.js.

Add the following line directly before (or above) “const views = require(‘./routes/views’);” or directly below the comment “// local dependencies” in your ./src/app.js:

```javascript
const db = require(‘./db’);
```

If you run it now, you should get a “database connected” message.
STEP 2: Create Comment and Story Mongoose Models.
Add Comment and Story Mongoose Models: Story

cd into the models directory we created earlier. We’ll start with story.js.

We want each story to have a creator_id, creator_name, and content, and we want each of these to be of type String.

Any idea how we can do this?
Add Comment and Story Mongoose Models: Story move into the models directory. We’ll start with story.js.

We want each story to have a creator_id, creator_name, and content, and we want each of these to be of type String.

Any idea how we can do this?

*We use schemas and mongoose models!*
Add Story Mongoose Model

Enter the following into story.js.

```
1  // import node modules
2  const mongoose = require('mongoose');
3
4  // define a schema
5  const StoryModelSchema = new mongoose.Schema({
6    creator_id : String,
7    creator_name : String,
8    content : String,
9  });
10
11  // compile model from schema
12  module.exports = mongoose.model('StoryModel', StoryModelSchema);
```
YOUR TURN! Add Comment Mongoose Models

Create the comment model story comments in comment.js.

We want the model for comment to have

- creator_id
- creator_name
- parent (which describes the story this comment is going into)
- content

We want all these fields to be Strings.

Make sure to include the module.exports statement.
Add Comment Mongoose Models (Solution)

```javascript
// import node modules
const mongoose = require('mongoose');

// define a schema
const CommentModelSchema = new mongoose.Schema({
creator_id : String,
creator_name : String,
parent : String,
content : String,
});

// compile model from schema
module.exports = mongoose.model('CommentModel', CommentModelSchema);
```
STEP 3:
Create Endpoints in our Server for Linking the Frontend and Backend with our Newly Implemented mongoDB database (mLab)
STEP 3 SETUP:

```bash
git reset --hard

git checkout step2

npm install
```

Recopy your `mongodbURI` from mlab
Use **api** Route for Database Requests

Open `api.js` from the `./src/routes` directory.
Part 1: Update `require` path

Within `api.js`, enter the correct path for `comment.js` where it says to do so.

(In line 6)
const Comment = require('../models/comment');
What do GET /whoami and GET /user do?

Both are not yet implemented. We won’t be implementing them either :P

/whoami: Let’s the using user know who he is.

/user: Allows you to find another user based on his or her id.
Part 2: Get all the stories via GET /stories

This endpoint asks the server to return ALL the stories saved in the database.

We implement this together.
Part 2: GET /stories (solution)

// add the following inside the get route
Story.find({}, function(err, stories) {
    res.send(stories);
});
Part 3: Implement POST /story

This server creates a new story based on the “content” parameter given in the request.

We do this together.
req.query vs. req.body

For **GET** requests:
Use `req.query`

For **POST** request:
Use `req.body`
Part 3: POST /story (solution)

// add the following inside the post route
const newStory = new Story({
    'creator_id': 'anonid',
    'creator_name': 'Anonmymous',
    'content': req.body.content,
});
newStory.save(function(err, story) {
    if (err) console.log(err);
});
res.send({});
Part 4: Implement GET /comment

We do this together.
Part 4: GET /comment (solution)

// replace /* input the parent parameter here */ with:

req.query.parent
Part 5: Implement POST /comment

Now your turn:

This endpoint saves a new comment into the database with both the "parent" and the "content" from the request.

*Hint:* Look at POST /story
// your code inside of the function in this route should be:
const newComment = new Comment({
  'creator_id': 'anonid',
  'creator_name': 'Anonymous',
  'parent': req.body.parent,
  'content': req.body.content,
});
newComment.save(function(err, comment) {
  if (err) console.log(err);
});
res.send({});
Part 6: Link `api.js` with `app.js`

We finally need to link `api.js` with `app.js`.

- Open `app.js` file, and right below the line "`const views = require('./routes/views');`", add the following:

  ```javascript
  const api = require('./routes/api');
  ```

- Then, scroll down till you see the "`app.use('/', views);`". Right below that, add the following:

  ```javascript
  app.use('/api', api);
  ```
Part 6: Reflect `api.js` changes in `app.js`

Now, your code should look like this:

```javascript
7   // local dependencies
8   const db = require('./db');
9   const views = require('./routes/views');
10  const api = require('./routes/api');
20  // set routes
21  app.use('/', views);
22  app.use('/api', api);
23  app.use('/static', express.static('public'));
```
STEP 4:
Make our Frontend Use the Our Own Database to Store Comments and Stories
STEP 4 SETUP:

- git reset --hard
- git checkout step3
- npm install

Recopy your mongodbURI from mlab
Look at Our Codebase First: ./public/js/

- Look at feed.js: There is new code and code skeleton added:
  - `newCommentDOMObject`: Submit new comments
  - `submitCommentHandler`: Handles what to do when we submit new comments
    - We will implement this!
  - `newStoryDOMObject`: Submit new story posts
  - `submitStoryHandler`: Handles what to do when we submit new stories
    - We will implement this!
  - `renderStories`: A new line added to display a new story add-box
Implement `submitStoryHandler`

This handler is what happens when we click on the submit button when trying to save a story.

We do this together.
Implement `submitStoryHandler` (solution)

```javascript
function submitStoryHandler() {
  const newStoryInput =
    document.getElementById('story-content-input');

  const data = {
    content: newStoryInput.value,
  };

  post('/api/story', data);
  newStoryInput.value = '';
}
```
Implement `submitCommentHandler`

This handler is what happens when we click on the submit button when trying to save add a comment to a story.

We do this together.
Implement `submitCommentHandler` (solution)

```javascript
function submitCommentHandler() {
    const commentInput = document.getElementById(this.getAttribute('story-id') + '-comment-input');
    const data = {
        content: commentInput.value,
        parent: this.getAttribute('story-id')
    };

    post('/api/comment', data);
    commentInput.value = '';
}
```
Recap

We learned to:

- Gain access to a mongodb instance via mlab
- Creating (free 500 mb) databases as you want
- Understand how databases work
- Hook remote mongodb instances to our nodejs app
- Save content to mongodb database with JavaScript on the front-end
THAT’S IT!
Mongoose Documentations & Further Readings

MongoDB Documentations: https://docs.mongodb.com/?ga=2.99200926.1225005866.1515560696-1152838387.1514762539

Mongoose Getting Started: http://mongoosejs.com/docs/

Documentations: http://mongoosejs.com/docs/guide.html