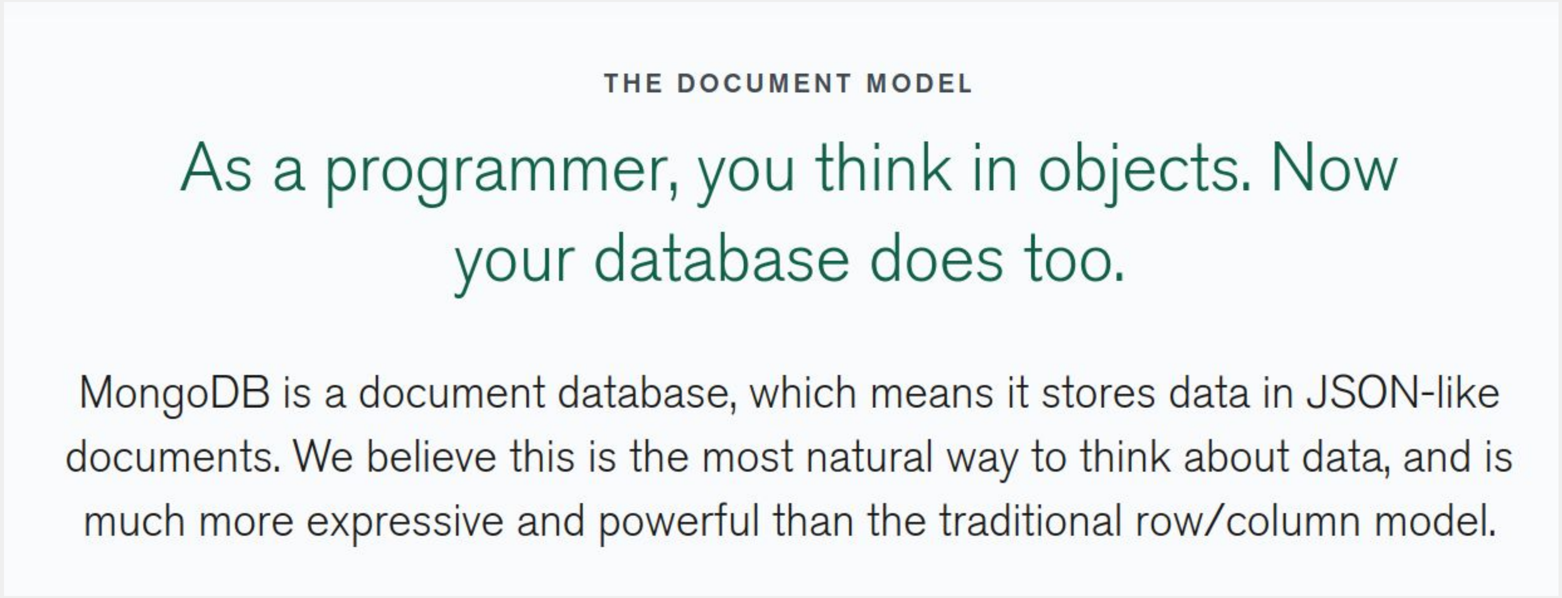
Workshop 6 - Databases

Jay Hilton and Joyce Yoon

What is MongoDB?

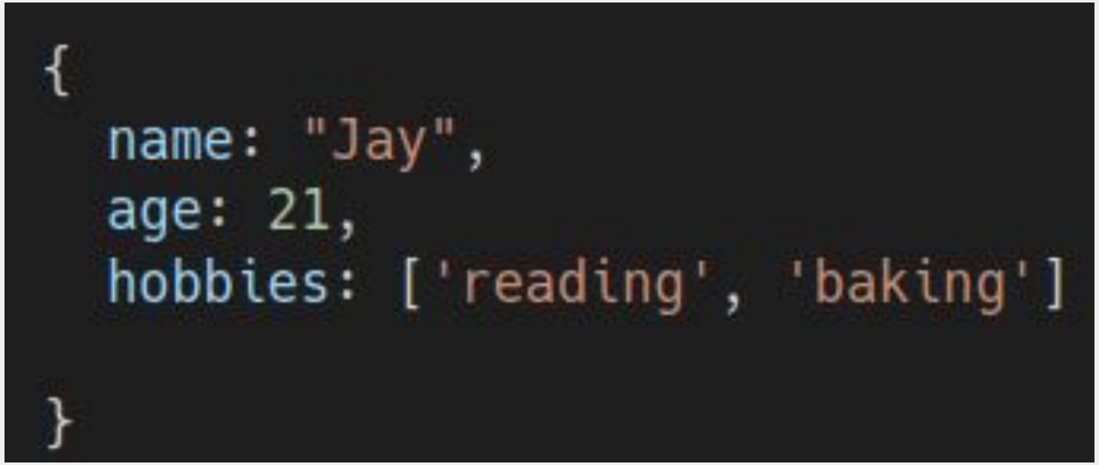


THE DOCUMENT MODEL

As a programmer, you think in objects. Now your database does too.

MongoDB is a document database, which means it stores data in JSON-like documents. We believe this is the most natural way to think about data, and is much more expressive and powerful than the traditional row/column model.

What is MongoDB?



```
{
   name: "Jay",
   age: 21,
   hobbies: ['reading', 'baking']

}
```

Why use MongoDB?

- Efficient when we need to write a lot to the database
- The structure of the data is very prone to changes
    - NoSQL gives us flexibility
- Relatively easy to use

so webscale

wow

very flexible

much speed

mongoDB

Structure

- MongoDB Instance
  - Database
    - Collections
      - Documents
        - Fields

Structure

- MongoDB Instance
    - Database
        - Collections
            - Documents

Structure

- MongoDB Instance
  - Database
    - Collections

Structure

- MongoDB Instance
  - Database

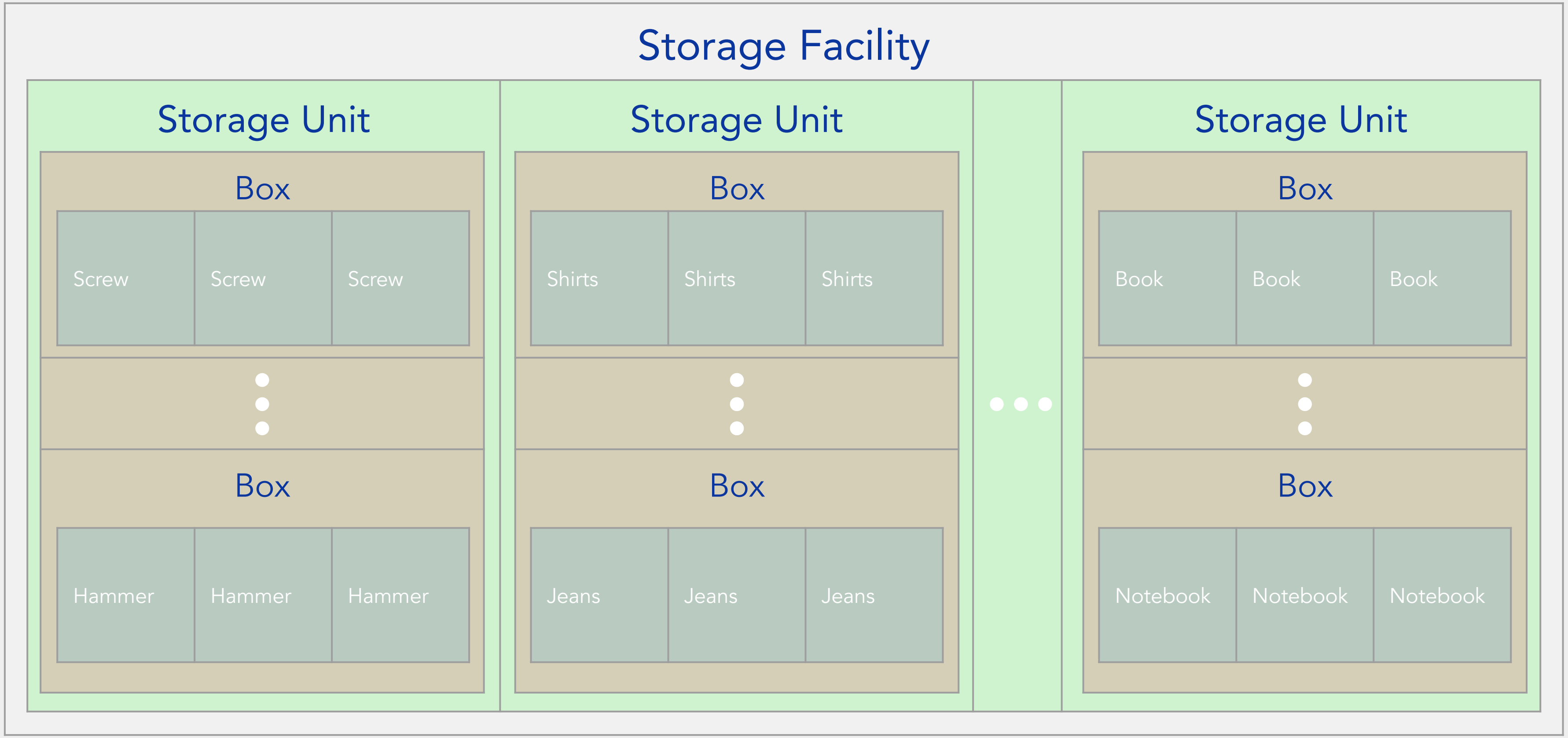
Structure

- MongoDB Instance

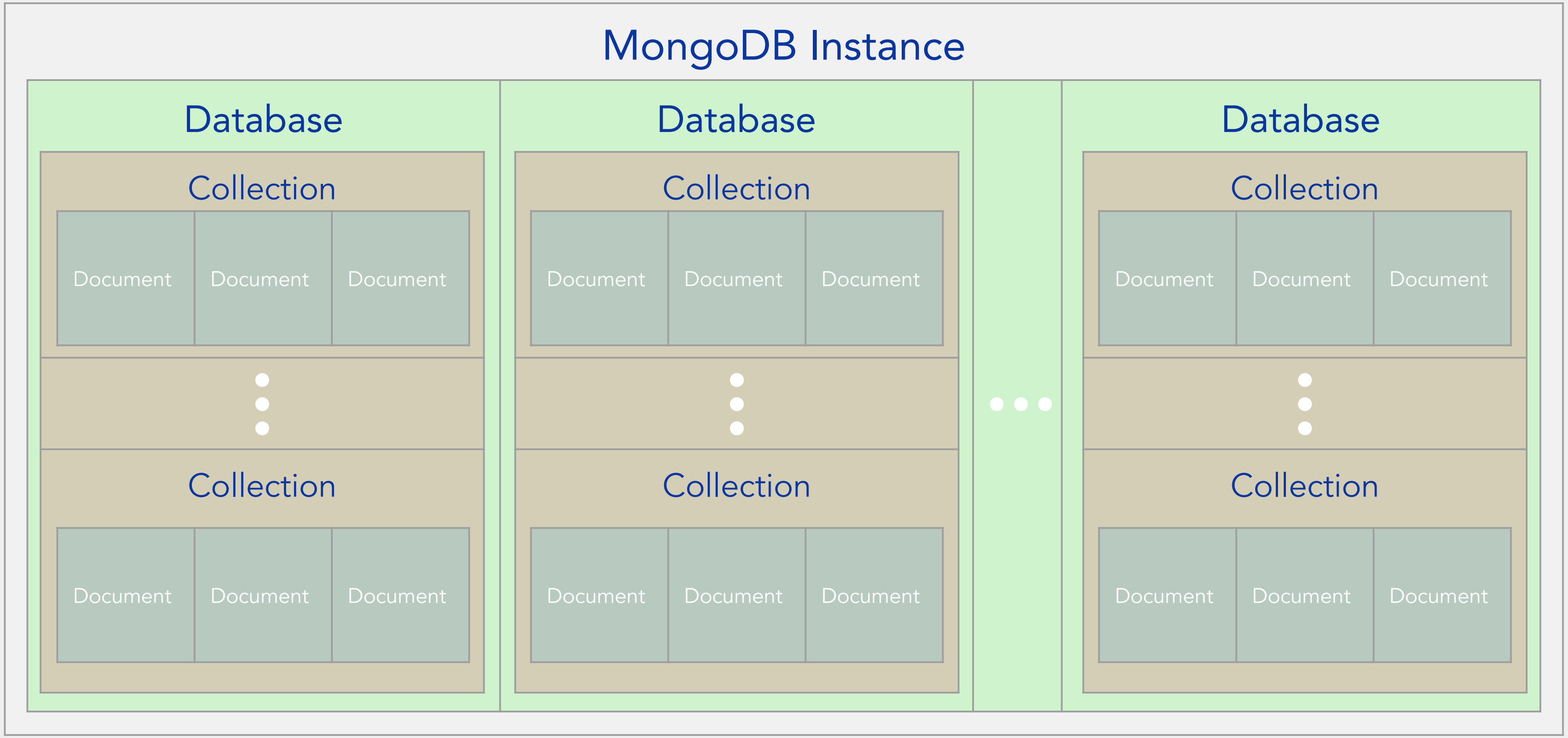
MongoDB Structure In Words

- **MongoDB Instance:** a group of databases
- **Database** (ex. Catbook database)**:** a group of collections, generally corresponds to one web application
- **Collection** (ex. Stories collection)**:** a group of very similar pieces of data. Ideally want all data in a given collection to have the same structure aka have the same keys and types
- **Document** (ex. Data for a single story)**:** a single JSON or Javascript object. A single piece of data in the the application, analogous to a row in SQL
- **Field** (ex. `content` property for a single story)**:** an attribute we want to record the value of, a key of the javascript object.

Structure


Storage Facility
Storage Unit
Box
Screw
Screw
Screw
Box
Hammer
Hammer
Hammer
Storage Unit
Box
Shirts
Shirts
Shirts
Box
Jeans
Jeans
Jeans
Storage Unit
Box
Book
Book
Book
Box
Notebook
Notebook
Notebook

Structure


MongoDB Instance
Database
Collection
Document
Document
Document
Collection
Document
Document
Document
Database
Collection
Document
Document
Document
Collection
Document
Document
Document
Database
Collection
Document
Document
Document
Collection
Document
Document
Document

Questions?
weblab.is/questions

Mongoose

NodeJS library that allows MongoDB integration

What is Mongoose?

wrapper that allows you to interact with MongoDB API

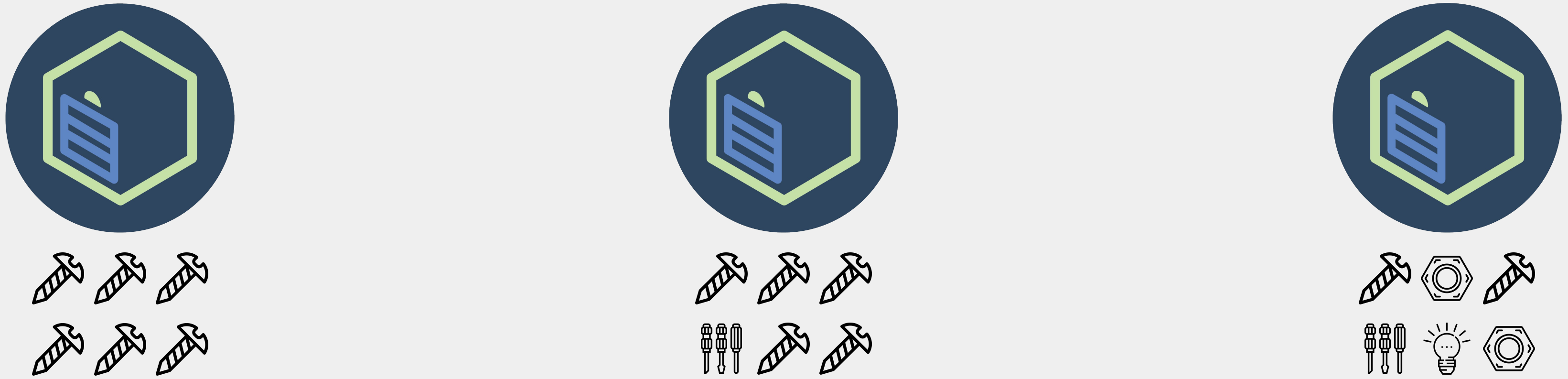
What does Mongoose do?

- Connects to cluster
  - We’ll cover code in the workshop
- Creates documents
- Interacts with databases
  - Create, Read, Update, Delete and more!

Why do we need Mongoose?

Mongoose vs Vanilla Mongo

- Mongo does not guarantee all documents in a collection have the same structure.

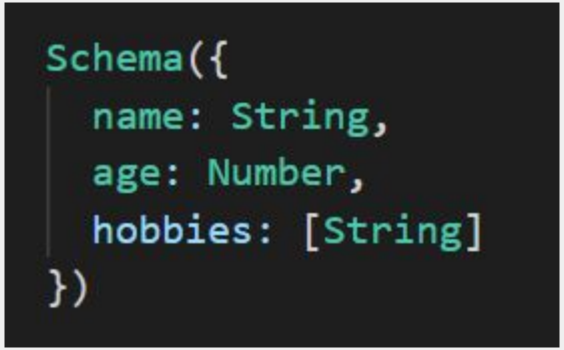


Schemas!

What is a Schema?

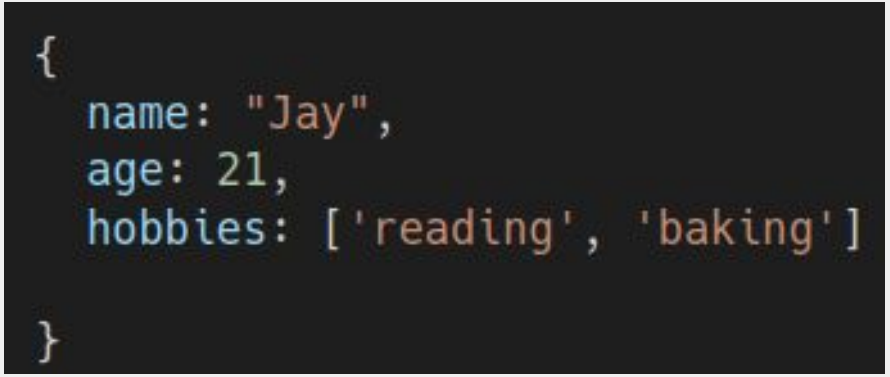
- Schemas define the **structure** of your documents
- Define the **keys** and **types** of the values corresponding to the keys
- Organization is key!

Mongoose Schema Example



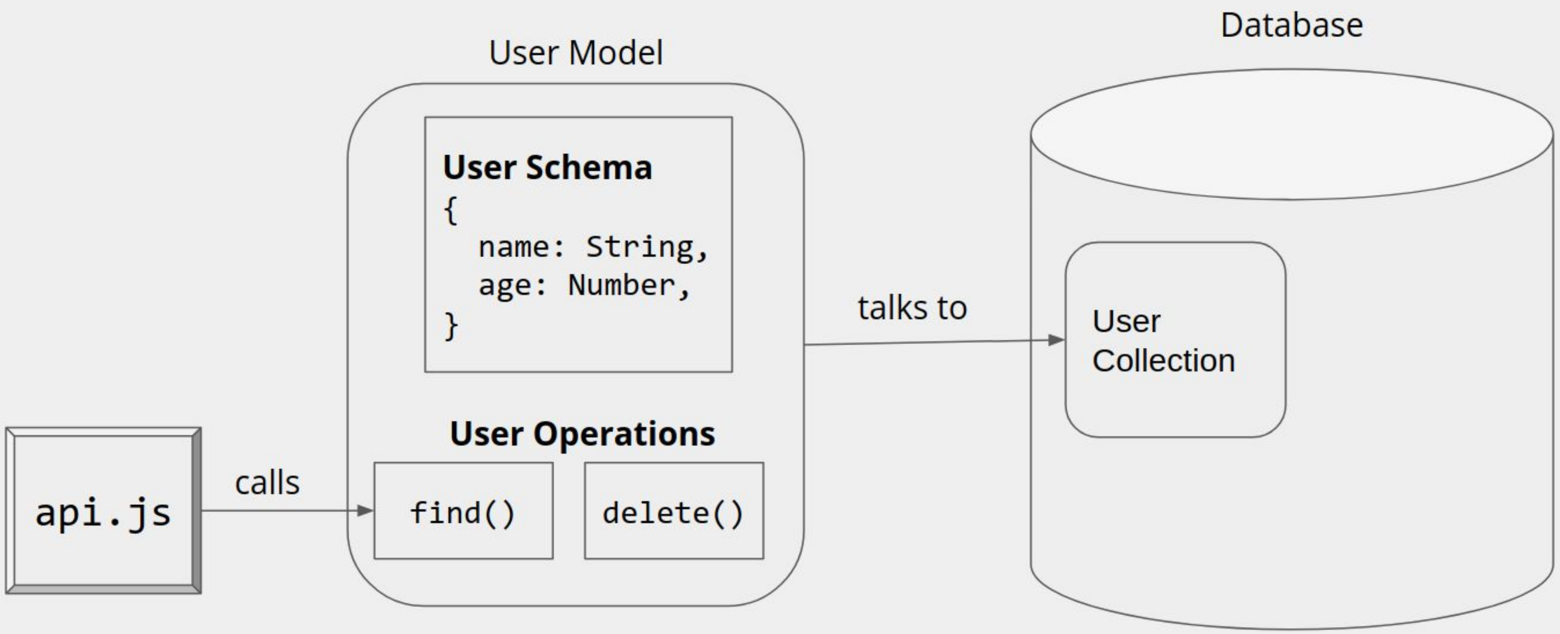
```
Schema({
  name: String,
  age: Number,
  hobbies: [String]
})
```

$\longrightarrow$



```
{
  name: "Jay",
  age: 21,
  hobbies: ['reading', 'baking']

}
```

Mongoose Structure



```
{
  name: String,
  age: Number,
}
```

“Models are responsible for creating and reading documents from the underlying MongoDB database.”

Mongoose Schemas: Processing Documents

- Means of structuring MongoDB documents
  - Specify fields within a document
- Each collection *should* have a schema

Mongoose Schema types

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

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

Mongoose Models

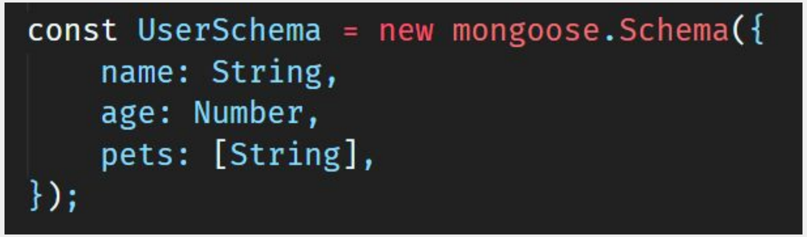
Models let you:

- Construct documents
- Get documents fitting the model
- Post documents
- ...or anything with documents fitting the model!

Models are like objects, but we can also use them to query or modify the database!

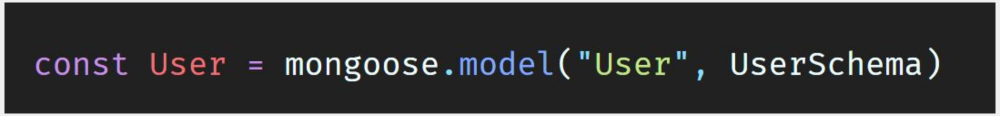
Creating a Mongoose Model (Generally)

1. Create a mongoose.Schema



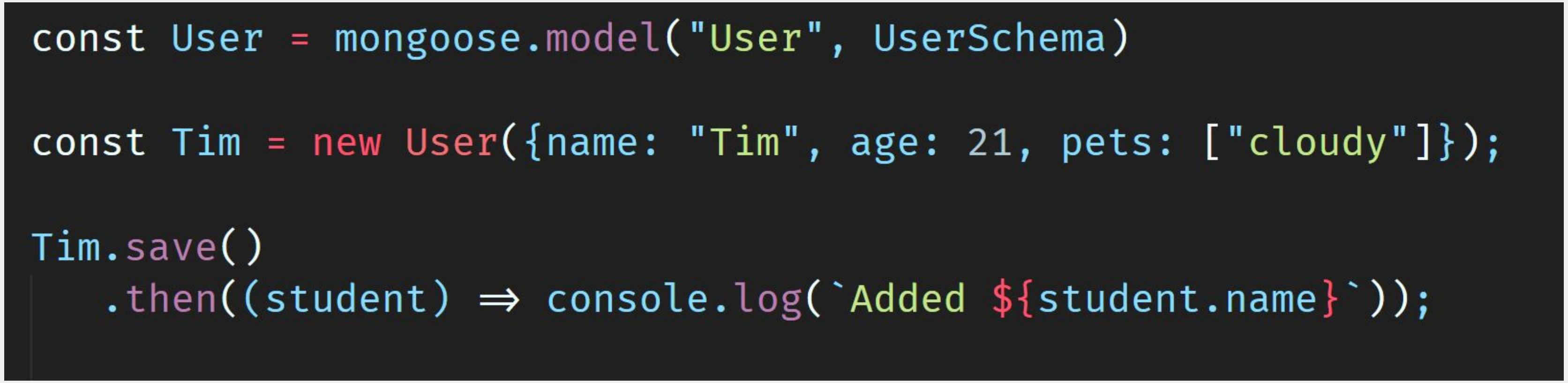
```
const UserSchema = new mongoose.Schema({
    name: String,
    age: Number,
    pets: [String],
});
```

2. Create a mongoose.model



```
const User = mongoose.model("User", UserSchema)
```

Creating Documents

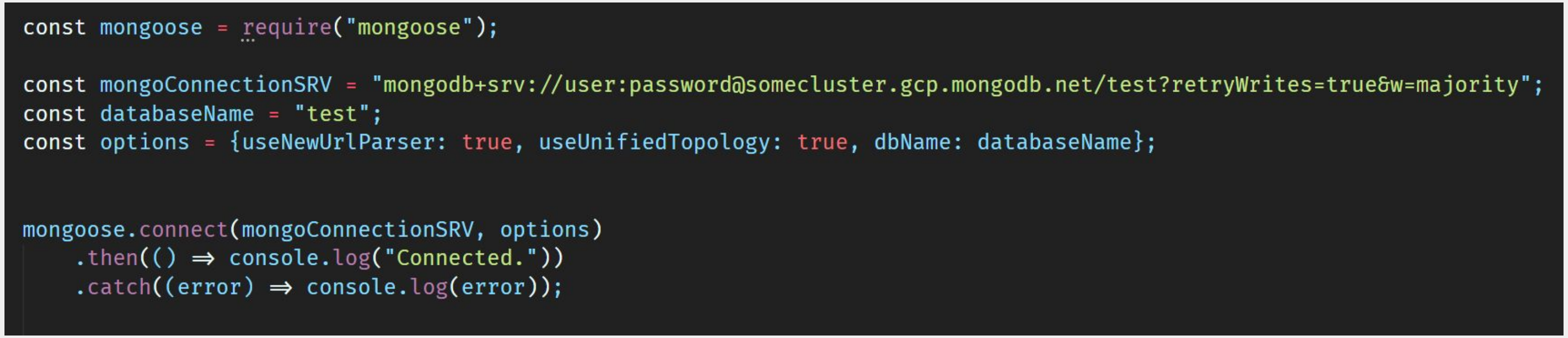


```javascript
const User = mongoose.model("User", UserSchema)

const Tim = new User({name: "Tim", age: 21, pets: ["cloudy"]});

Tim.save()
    .then((student) => console.log(`Added ${student.name}`));
```

All together

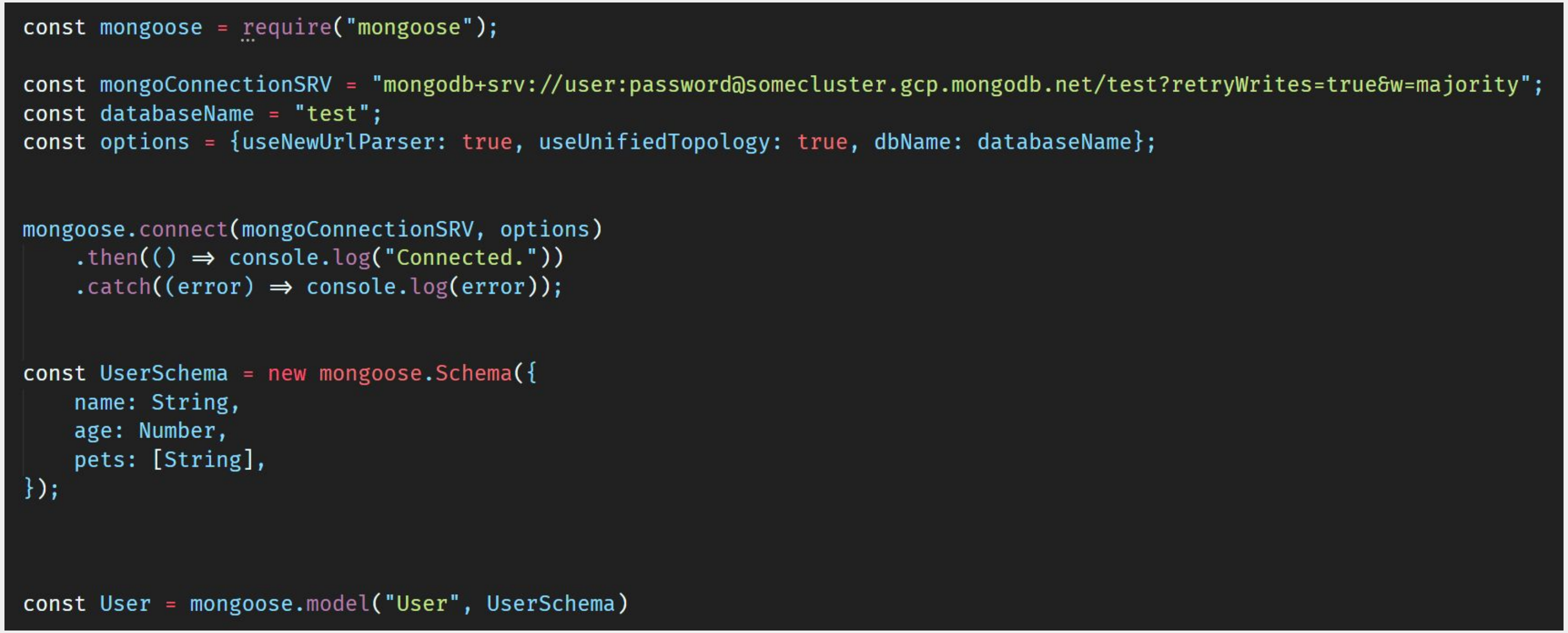
```
const mongoose = require("mongoose");

const mongoConnectionSRV = "mongodb+srv://user:password@somecluster.gcp.mongodb.net/test?retryWrites=true&w=majority";
const databaseName = "test";
const options = {useNewUrlParser: true, useUnifiedTopology: true, dbName: databaseName};
```

All together



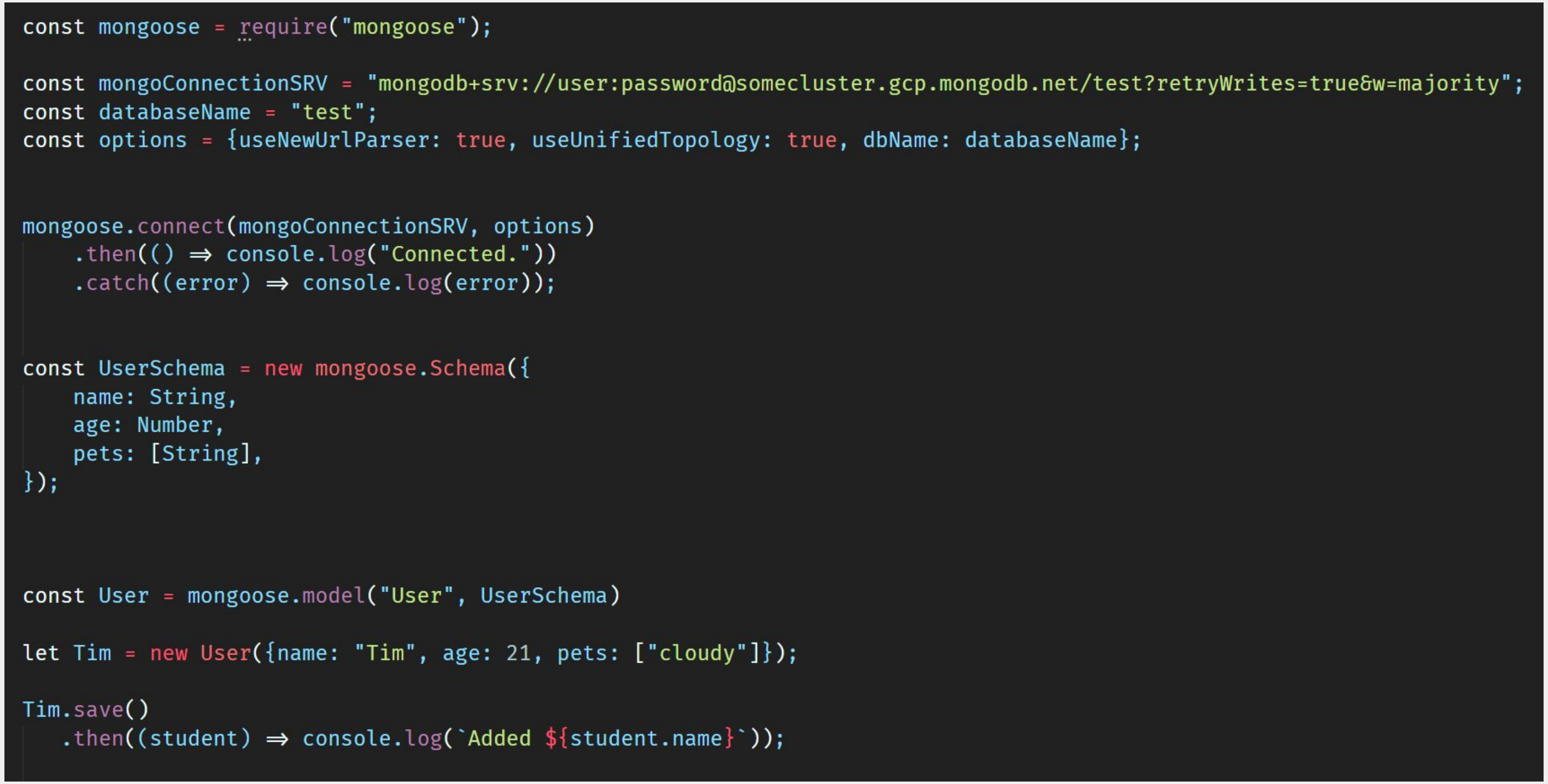
```
const mongoose = require("mongoose");

const mongoConnectionSRV = "mongodb+srv://user:password@somecluster.gcp.mongodb.net/test?retryWrites=true&w=majority";
const databaseName = "test";
const options = {useNewUrlParser: true, useUnifiedTopology: true, dbName: databaseName};


mongoose.connect(mongoConnectionSRV, options)
    .then(() => console.log("Connected."))
    .catch((error) => console.log(error));
```

All together



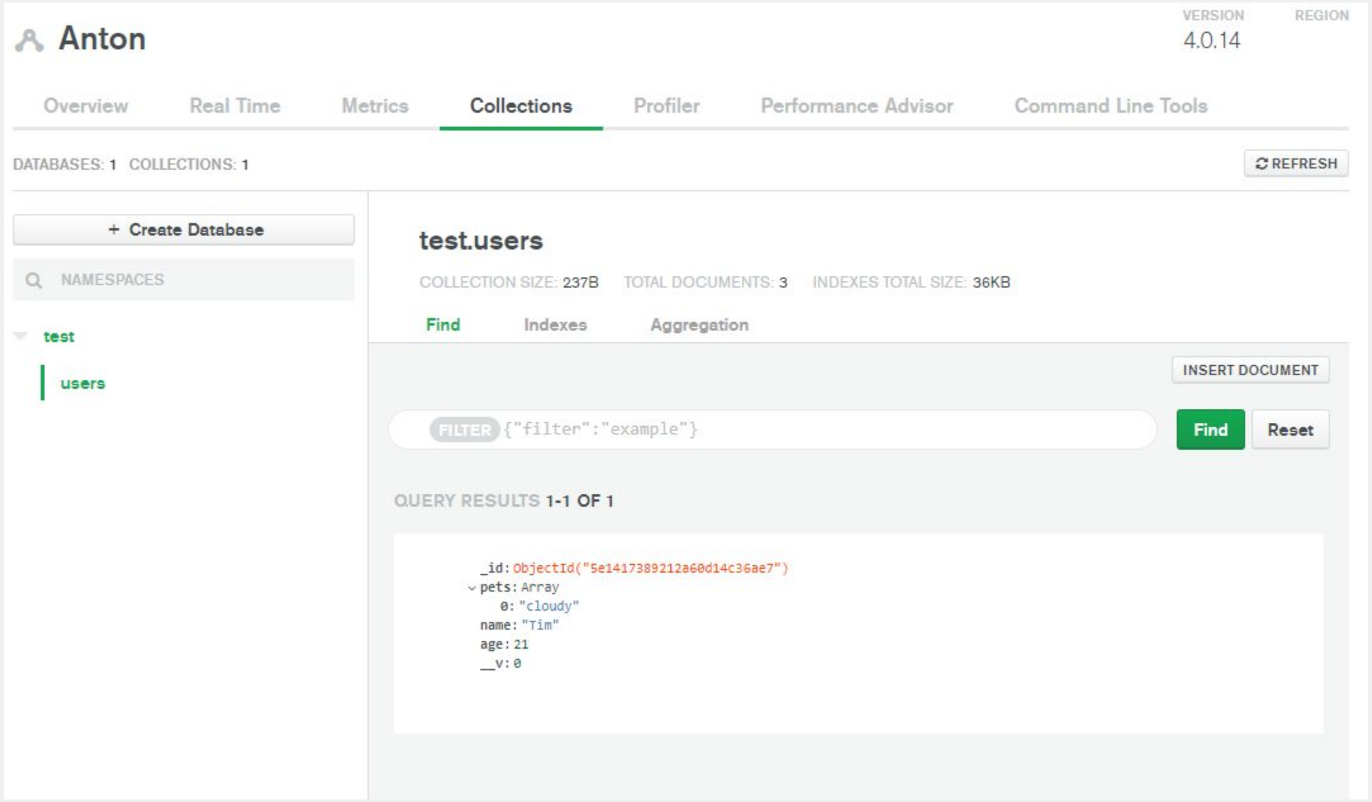
```
const mongoose = require("mongoose");

const mongoConnectionSRV = "mongodb+srv://user:password@somecluster.gcp.mongodb.net/test?retryWrites=true&w=majority";
const databaseName = "test";
const options = {useNewUrlParser: true, useUnifiedTopology: true, dbName: databaseName};


mongoose.connect(mongoConnectionSRV, options)
    .then(() => console.log("Connected."))
    .catch((error) => console.log(error));


const UserSchema = new mongoose.Schema({
    name: String,
    age: Number,
    pets: [String],
});



const User = mongoose.model("User", UserSchema)
```

All together

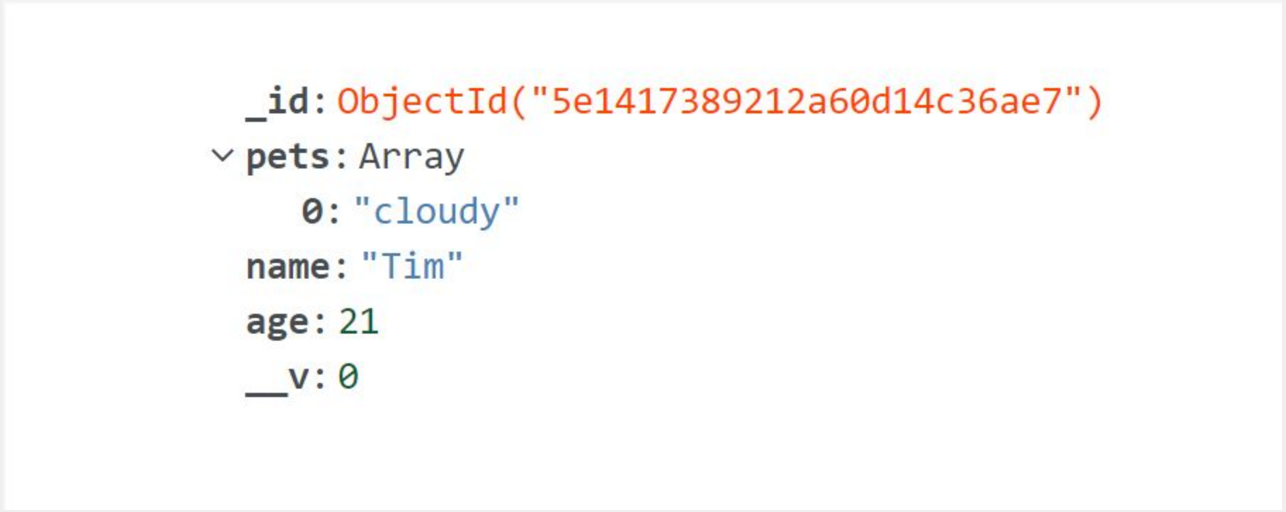


```javascript
const mongoose = require("mongoose");

const mongoConnectionSRV = "mongodb+srv://user:password@somecluster.gcp.mongodb.net/test?retryWrites=true&w=majority";
const databaseName = "test";
const options = {useNewUrlParser: true, useUnifiedTopology: true, dbName: databaseName};


mongoose.connect(mongoConnectionSRV, options)
    .then(() => console.log("Connected."))
    .catch((error) => console.log(error));


const UserSchema = new mongoose.Schema({
    name: String,
    age: Number,
    pets: [String],
});



const User = mongoose.model("User", UserSchema)

let Tim = new User({name: "Tim", age: 21, pets: ["cloudy"]});

Tim.save()
   .then((student) => console.log(`Added ${student.name}`));
```

Meanwhile on Atlas…

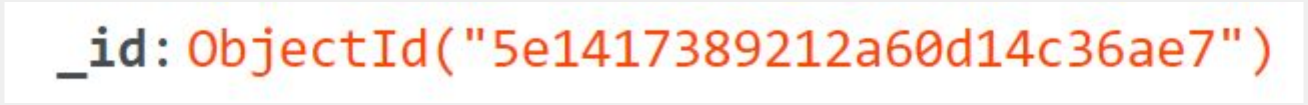


```
_id: ObjectId("5e1417389212a60d14c36ae7")
⌄ pets: Array
    0: "cloudy"
  name: "Tim"
  age: 21
  __v: 0
```



```
  _id: ObjectId("5e1417389212a60d14c36ae7")
v pets: Array
     0: "cloudy"
  name: "Tim"
  age: 21
  __v: 0
```

Wait

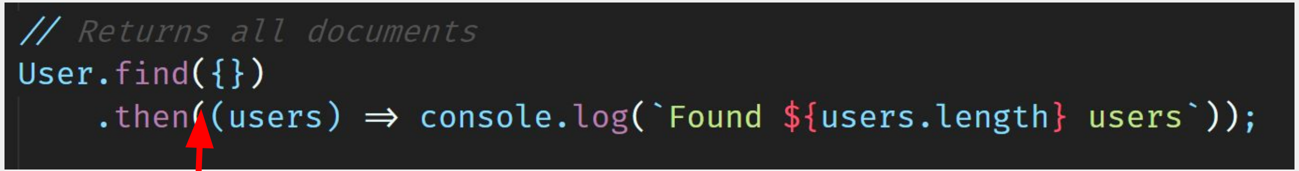


```
_id: ObjectId("5e1417389212a60d14c36ae7")
```

_id

- Every document is automatically assigned a unique identifier
- The identifier is assigned under the “_id” field.
- Useful when there’s a relationship between documents

Finding Documents

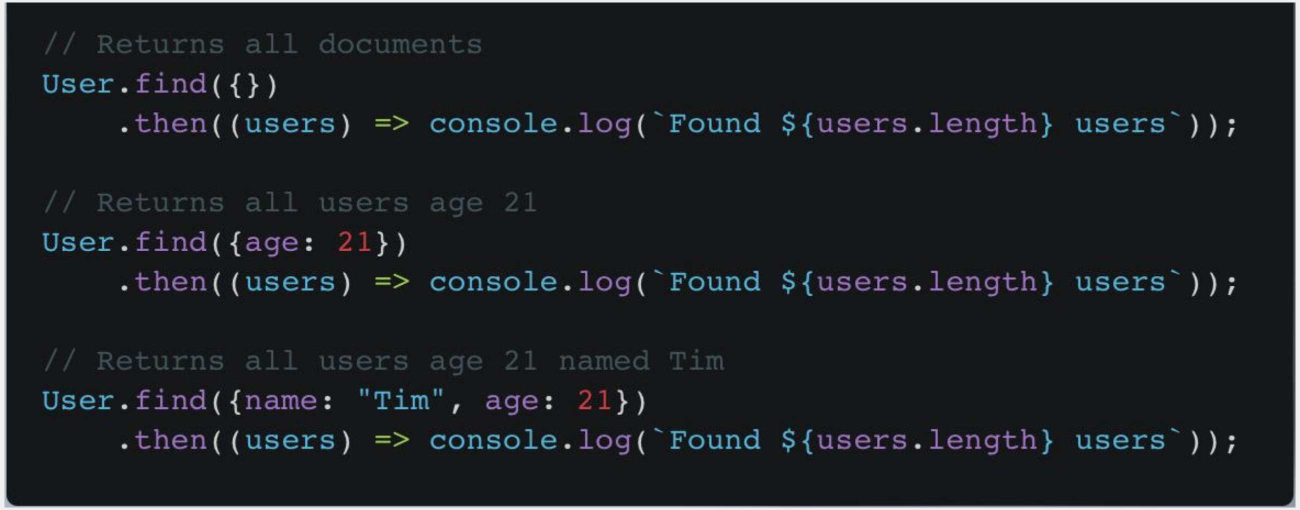


```
// Returns all documents
User.find({})
    .then((users) => console.log(`Found ${users.length} users`));
```

The first argument describes how to filter the collection

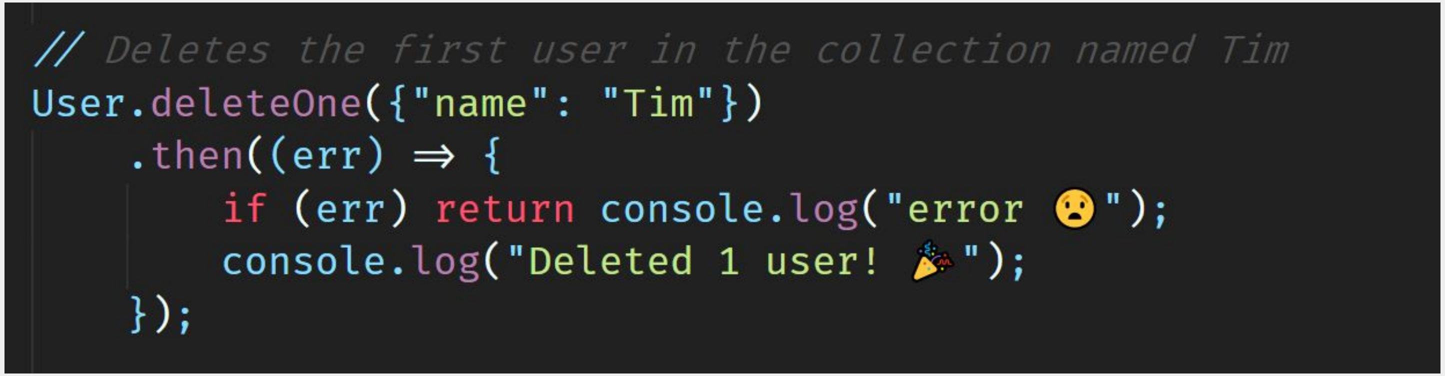
Finding Documents

- You can add as many parameters as you want to the filter. This is very useful!



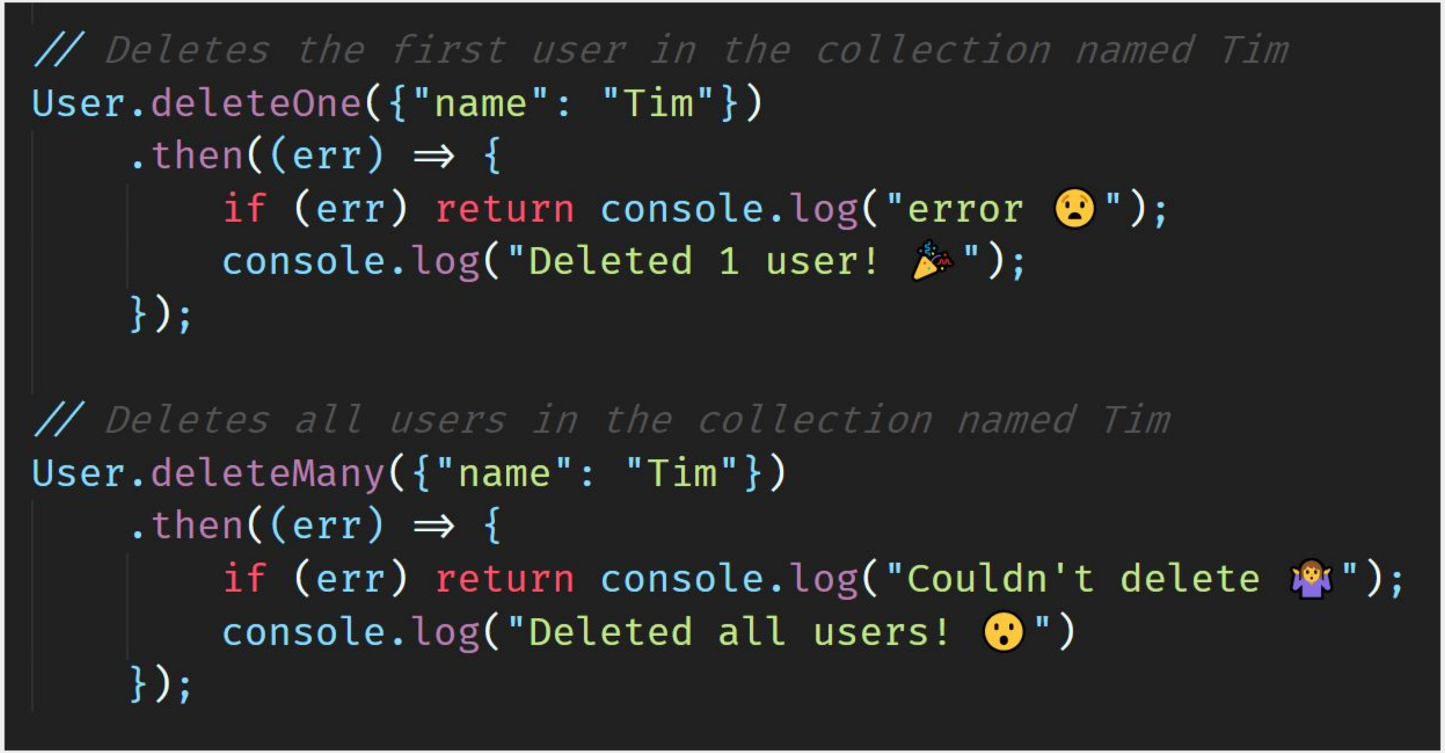
```
// Returns all documents
User.find({})
    .then((users) => console.log(`Found ${users.length} users`));

// Returns all users age 21
User.find({age: 21})
    .then((users) => console.log(`Found ${users.length} users`));

// Returns all users age 21 named Tim
User.find({name: "Tim", age: 21})
    .then((users) => console.log(`Found ${users.length} users`));
```

Deleting Documents



```
// Deletes the first user in the collection named Tim
User.deleteOne({"name": "Tim"})
    .then((err) => {
        if (err) return console.log("error 😟");
        console.log("Deleted 1 user! 🎉");
    });
```

Deleting Documents



```
// Deletes the first user in the collection named Tim
User.deleteOne({"name": "Tim"})
    .then((err) => {
        if (err) return console.log("error 😟");
        console.log("Deleted 1 user! 🎉");
    });

// Deletes all users in the collection named Tim
User.deleteMany({"name": "Tim"})
    .then((err) => {
        if (err) return console.log("Couldn't delete 🤷‍♀️");
        console.log("Deleted all users! 😮")
    });
```

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

Workshop:
Hook Database to Your Catbook App

Workshop Plan

- Hook back-end server up with mongo database
- Create models for our comments & stories
- Modify our API endpoints to use our Mongoose models

For sample code, see:
weblab.is/mongo-snippets

STEP -1:

Connect Your App to MongoDB with Mongoose

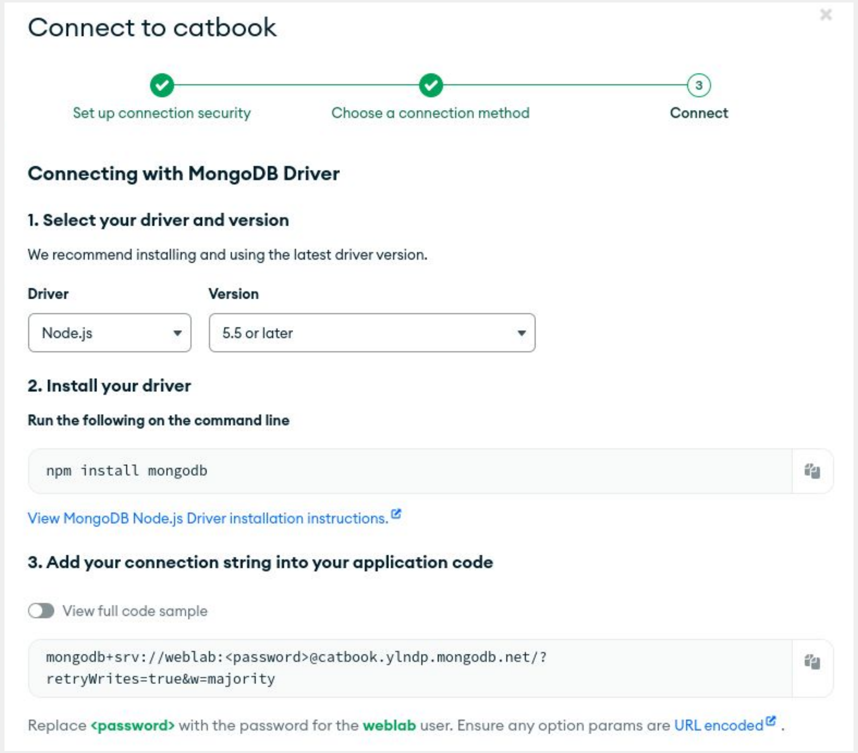
SETUP:

```
git fetch
git reset --hard
git checkout w6-starter
```

Connect Your App to Your Mongo DBMS

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

Enter your SRV from MongoDB Atlas where it says to do it in the comments.



Connect to catbook

Connecting with MongoDB Driver

1. Select your driver and version

We recommend installing and using the latest driver version.

Driver: Node.js

Version: 5.5 or later

2. Install your driver

Run the following on the command line

```
npm install mongodb
```

View MongoDB Node.js Driver installation instructions.

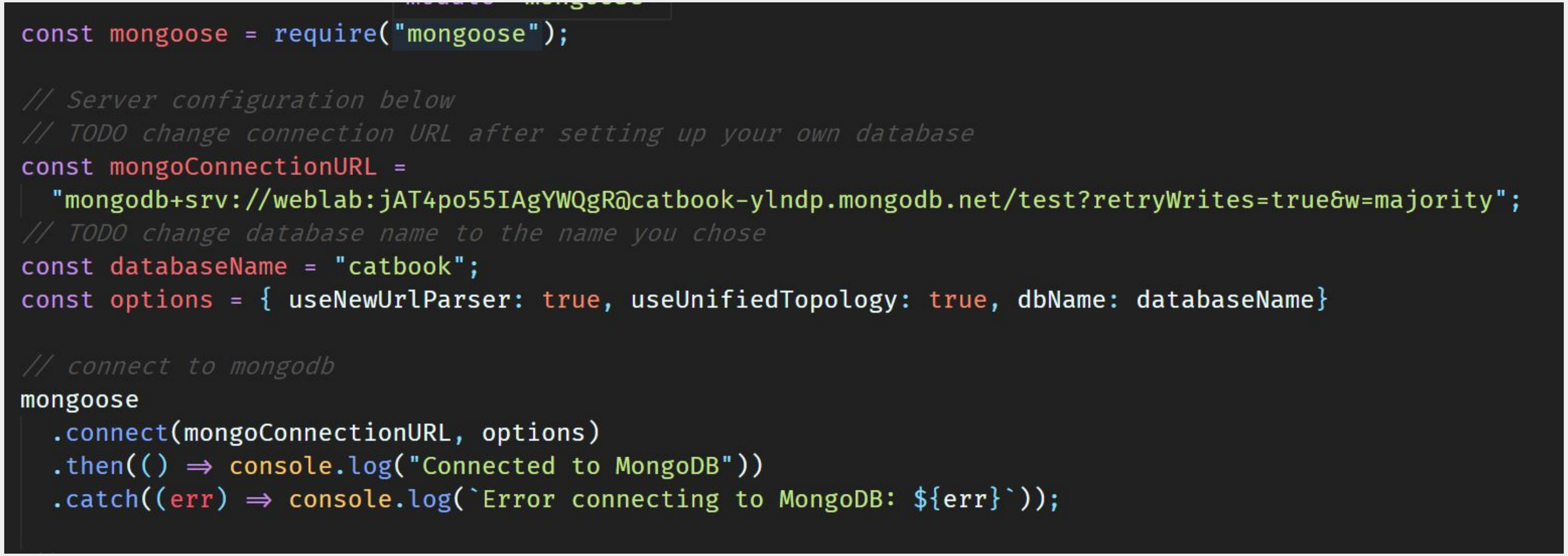
3. Add your connection string into your application code

```
mongodb+srv://weblab:<password>@catbook.ylndp.mongodb.net/?
retryWrites=true&w=majority
```

Replace **<password>** with the password for the **weblab** user. Ensure any option params are URL encoded.

Setting Up MongoDB with Mongoose

`server.js`.



```
const mongoose = require("mongoose");

// Server configuration below
// TODO change connection URL after setting up your own database
const mongoConnectionURL =
  "mongodb+srv://weblab:jAT4po55IAgYWQgR@catbook-ylndp.mongodb.net/test?retryWrites=true&w=majority";
// TODO change database name to the name you chose
const databaseName = "catbook";
const options = { useNewUrlParser: true, useUnifiedTopology: true, dbName: databaseName}

// connect to mongodb
mongoose
  .connect(mongoConnectionURL, options)
  .then(() => console.log("Connected to MongoDB"))
  .catch((err) => console.log(`Error connecting to MongoDB: ${err}`));
```

If you're having trouble, make sure you included your username & password

Connect Your App to Your Mongo DBMS (“solution”)

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

```
const mongoConnectionURL =
"mongodb+srv://weblab:jAT4po55IAgYWQgR@catbook-yln
dp.mongodb.net/test?retryWrites=true&w=majority";
```

(in one line!)

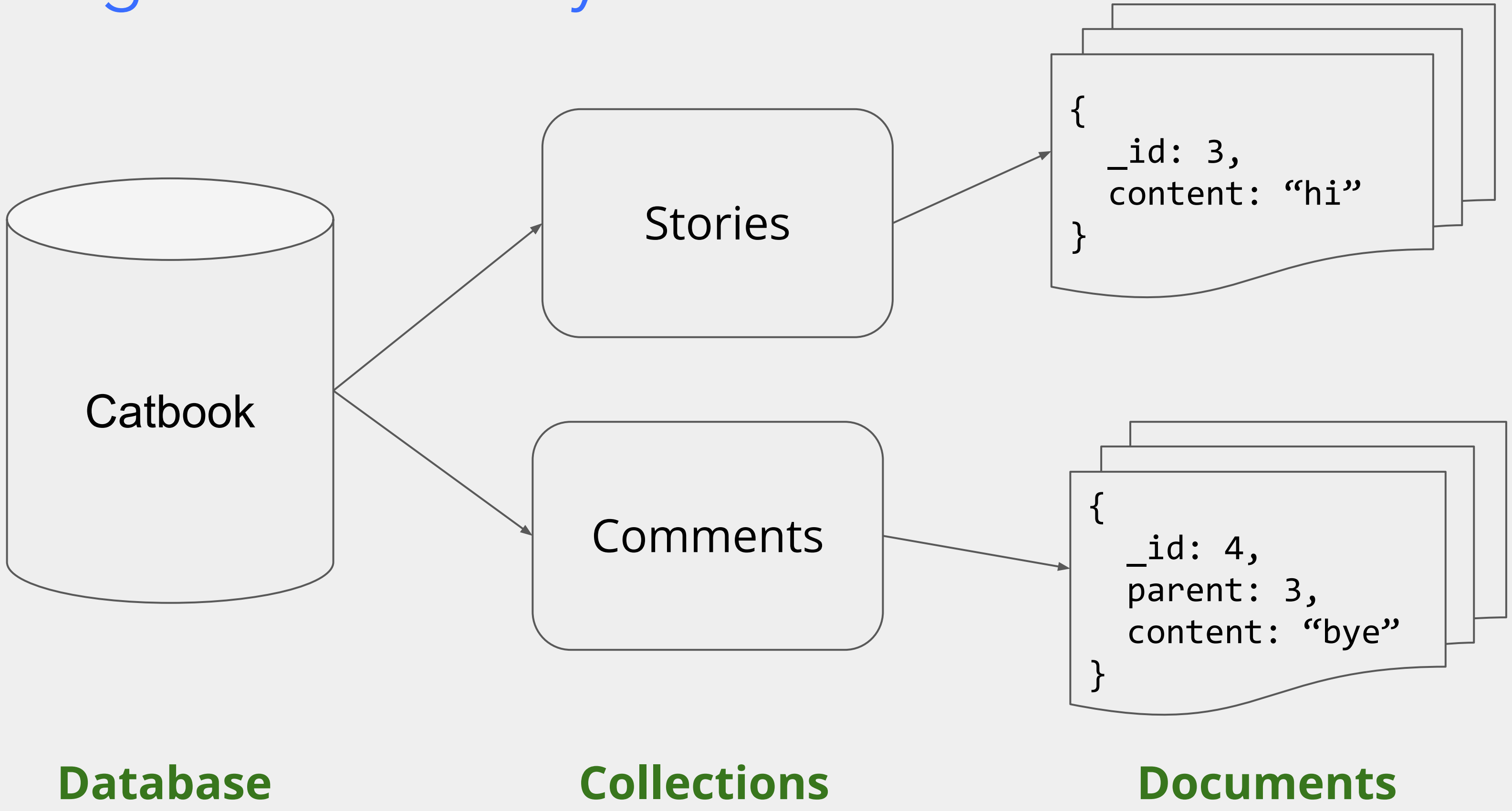
Run It From Your Root Directory

- `npm install`
- `npm start`
- If you run it now, you should get a “Connected to MongoDB” message.

STEP 0:

Create Comment and Story Mongoose Models.

MongoDB Hierarchy



```
{
  _id: 3,
  content: “hi”
}
```

```
{
  _id: 4,
  parent: 3,
  content: “bye”
}
```

Add Comment and Story Mongoose Models: Story

In the `models` directory, open `story.js`.

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

Any idea how we can do this?

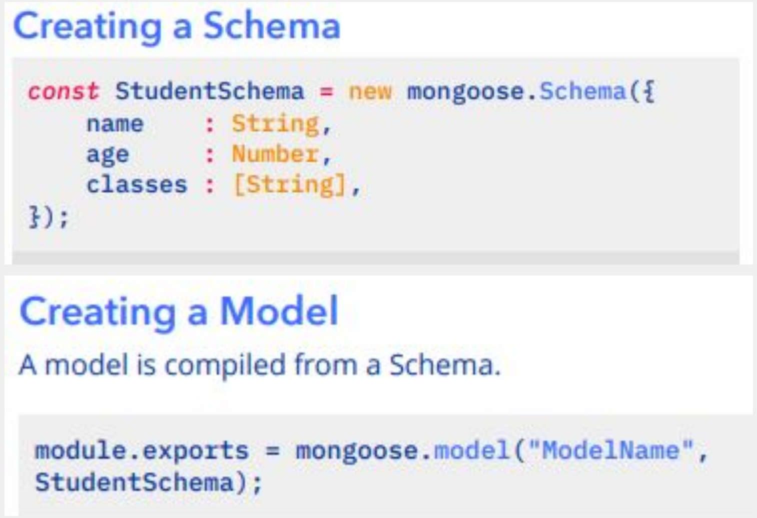
Add Comment and Story Mongoose Models: Story

In the `models` directory, open `story.js`.

We want each story to have a `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!



Creating a Schema

```
const StudentSchema = new mongoose.Schema({
    name    : String,
    age     : Number,
    classes : [String],
});
```

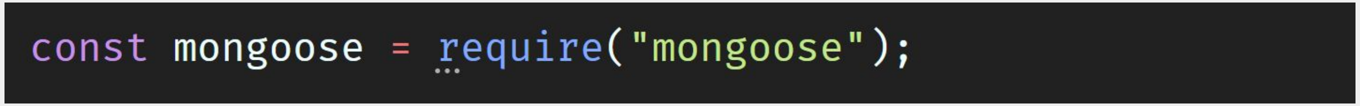
Creating a Model

A model is compiled from a Schema.

```
module.exports = mongoose.model("ModelName",
StudentSchema);
```

Add Story Mongoose Model

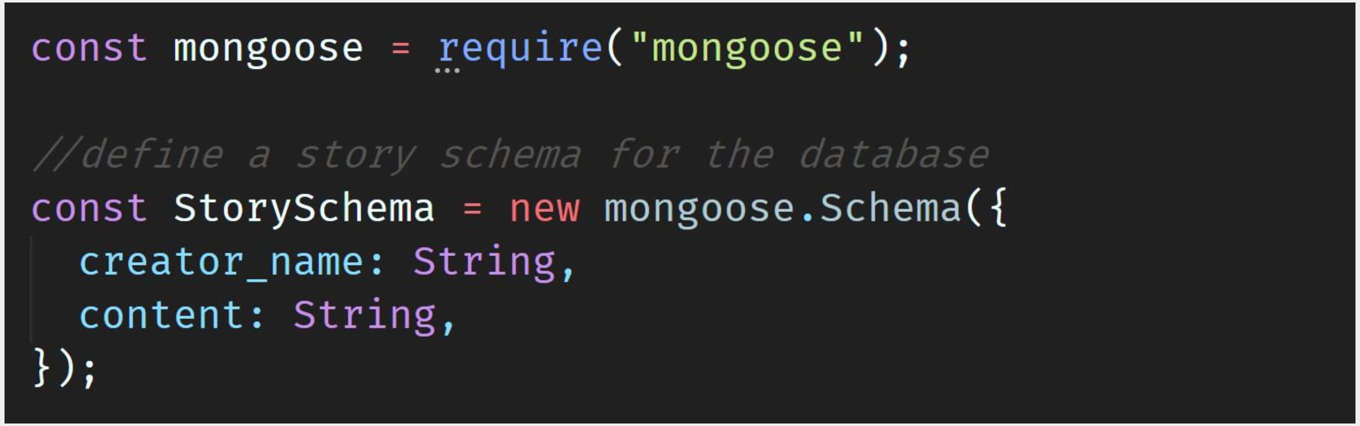
Enter the following into **`story.js`**.



```
const mongoose = require("mongoose");
```

Add Story Mongoose Model

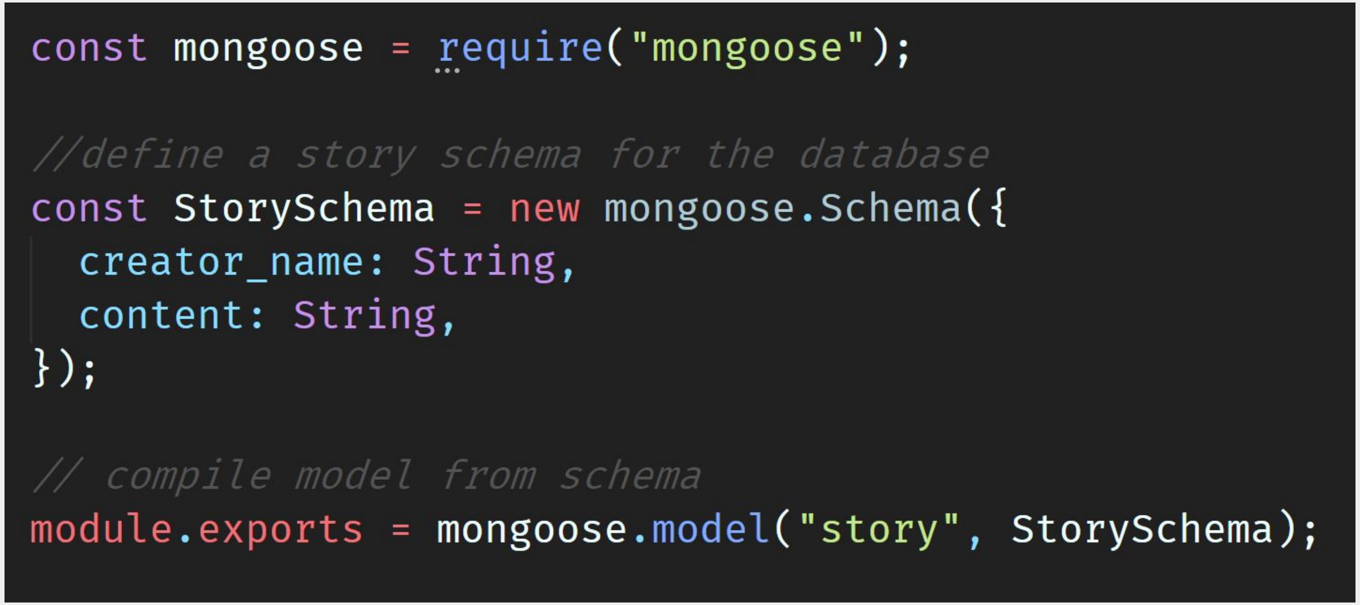
Enter the following into **`story.js`**.



```
const mongoose = require("mongoose");

//define a story schema for the database
const StorySchema = new mongoose.Schema({
  creator_name: String,
  content: String,
});
```

Add Story Mongoose Model

Enter the following into `story.js`.



```
const mongoose = require("mongoose");

//define a story schema for the database
const StorySchema = new mongoose.Schema({
  creator_name: String,
  content: String,
});

// compile model from schema
module.exports = mongoose.model("story", StorySchema);
```

Add Comment Mongoose Models (Your Turn)

Create the comment model for story comments in `comment.js`.

We want the model for comment to have

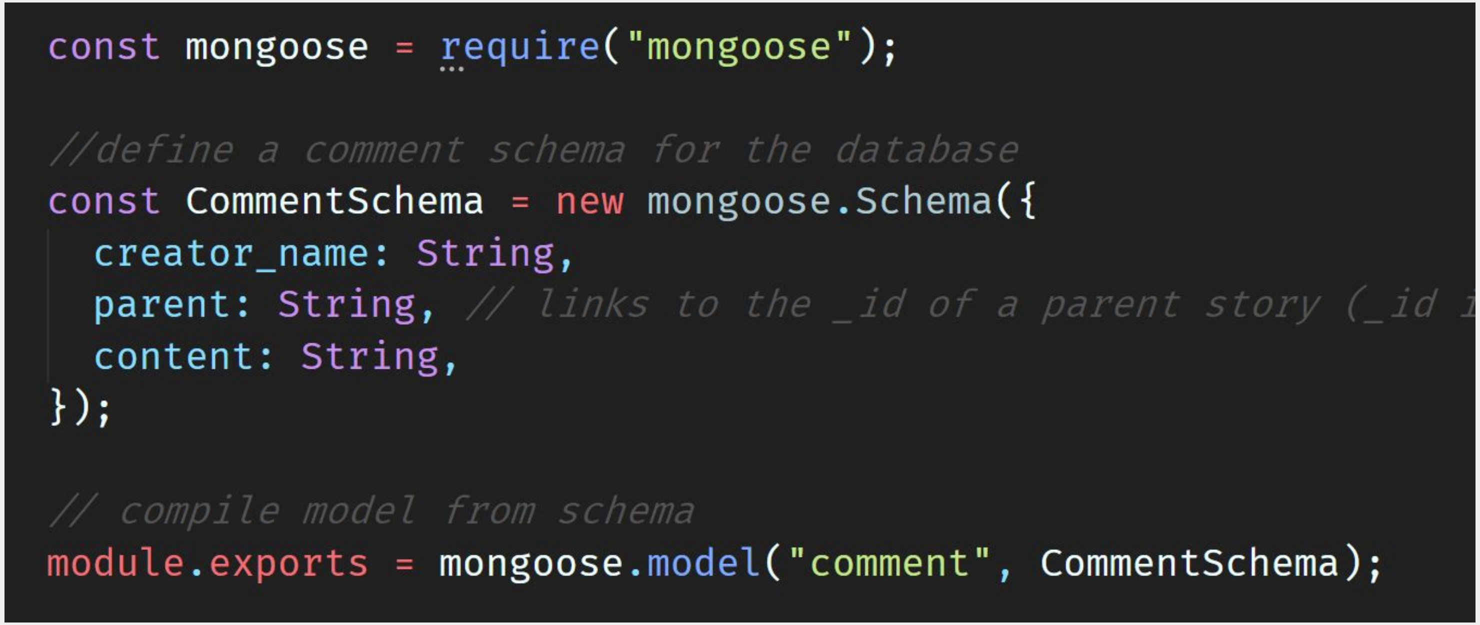
- `creator_name`
- `parent` (which describes the story this `comment` is going into)
- `content`

We want all these fields to be `String`s.

Make sure to include the `module.exports` statement.

Add Comment Mongoose Models (Solution)

Enter the following into `comment.js`.



```javascript
const mongoose = require("mongoose");

//define a comment schema for the database
const CommentSchema = new mongoose.Schema({
  creator_name: String,
  parent: String, // links to the _id of a parent story (_id
  content: String,
});

// compile model from schema
module.exports = mongoose.model("comment", CommentSchema);
```

STEP 1:

Link the Backend with our Newly Implemented MongoDB database (Atlas)

STEP 1 SETUP:

```
git reset --hard
git checkout w6-step1
```

Recopy your SRV into server.js if it disappeared

Use `api` Route for Database Requests

Open `api.js` from the `./server` directory.

Part 1: Update `require` path

This allows us to use the exported models!

Within **`api.js`**, import the comment model below "**`const Story = require("./models/story.js");`**"

Now, import the Comment model (use the path for story.js as an example).

Part 1: Update `require` path

This allows us to use the exported models!

Within **`api.js`**, import the comment model below "**`const Story = require("./models/story.js");`**"

Now, import the Comment model (use the path for story.js as an example).

```
const Comment = require('./models/comment');
```

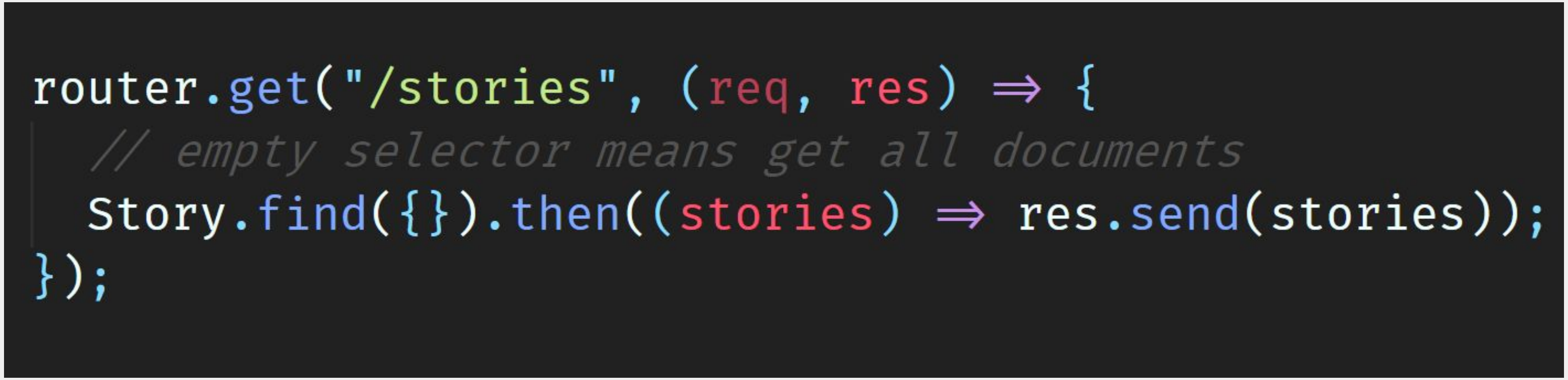
Part 2: Get all the stories via `GET /stories`

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

How would we do this?

Hint: try to find relevant code in weblab.is/mongo-snippets

Part 2: GET /stories (solution)

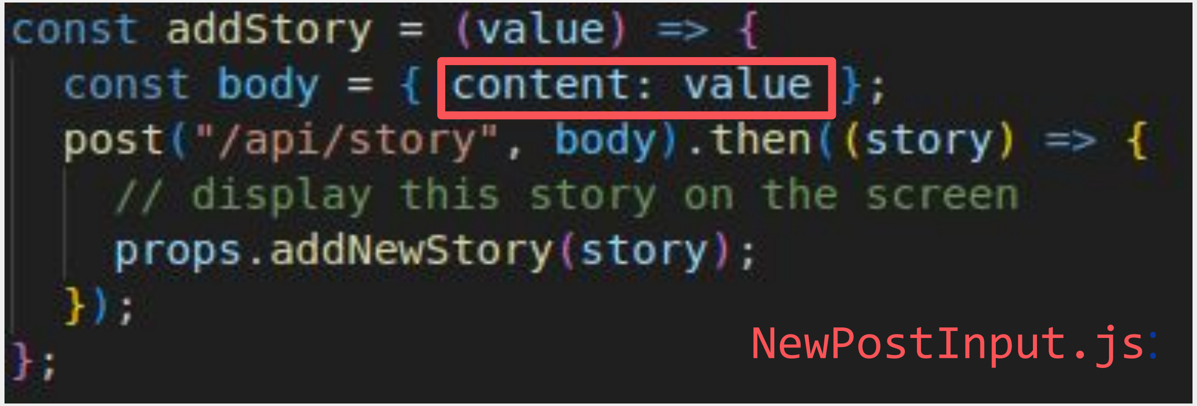


```javascript
router.get("/stories", (req, res) => {
  // empty selector means get all documents
  Story.find({}).then((stories) => res.send(stories));
});
```

Part 3: Implement `POST /story`

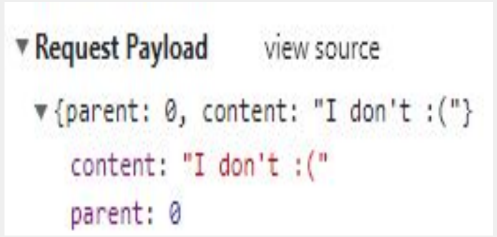
This server creates a new story based on the "`content`" parameter given in the request.

Where do we get the content? `req.body.content`



```
const addStory = (value) => {
  const body = { content: value };
  post("/api/story", body).then((story) => {
    // display this story on the screen
    props.addNewStory(story);
  });
};
```

NewPostInput.js:



```
▼ Request Payload     view source
  ▼ {parent: 0, content: "I don't :("}
      content: "I don't :("
      parent: 0
```

`req.query` vs. `req.body`

For `GET` requests:

Use `req.query`

E.g. `req.query.content`

For `POST` request:

Use `req.body`

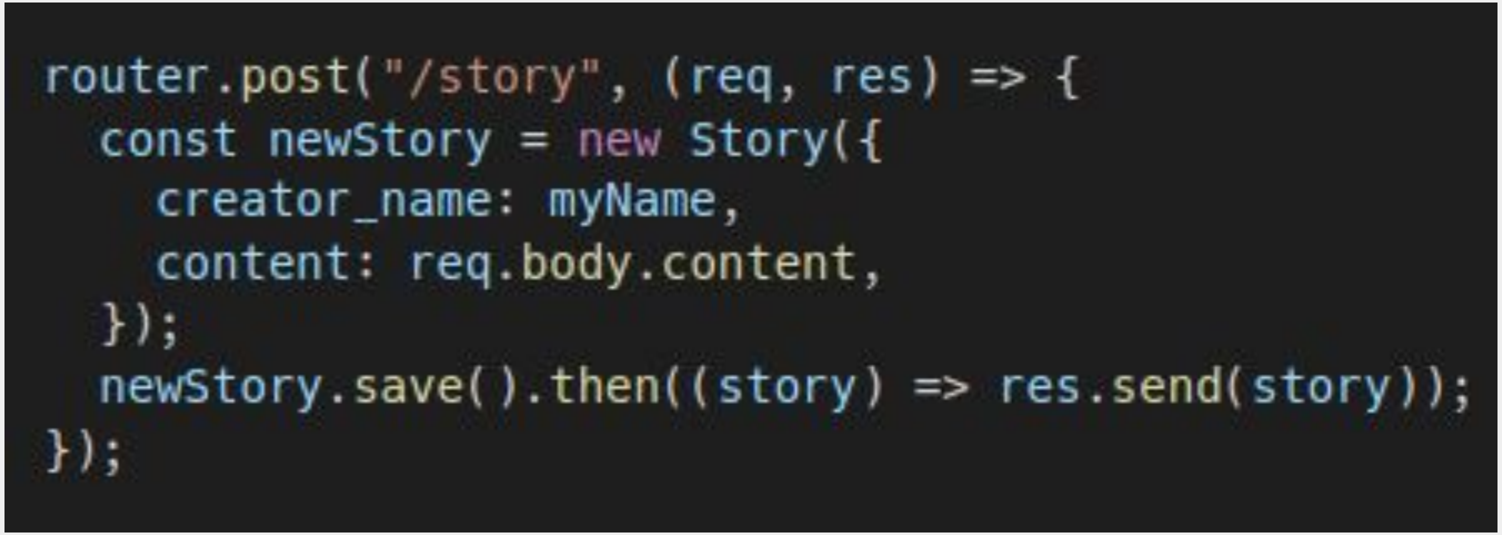
`req.body.content`

`How would you implement /story?`

`Note: You want to use the constant myName as the creator_name since we do not have access to the creator name yet.`

Hint: try to find relevant code in weblab.is/mongo-snippets

Part 3: POST /story (solution)



```
router.post("/story", (req, res) => {
  const newStory = new Story({
    creator_name: myName,
    content: req.body.content,
  });
  newStory.save().then((story) => res.send(story));
});
```

Let’s test post a story!

In one terminal:

```
npm start
```

In *another* terminal:

```
npm run hotloader
```

… and go check `localhost:5050` in your browser!

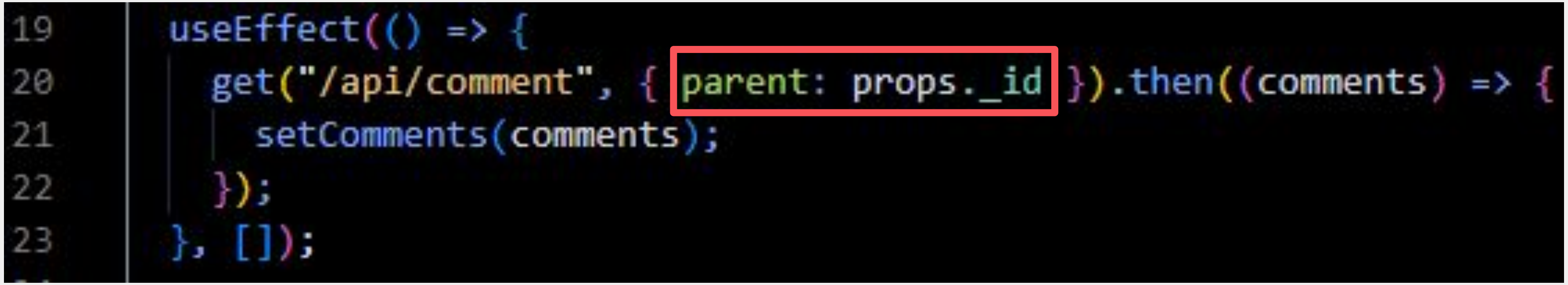
STEP 2 SETUP:

```
git reset --hard
git checkout w6-step2
```

Recopy your SRV into server.js

The GET body

We included the parent story's _id prop when we made the GET from the frontend!



```
useEffect(() => {
  get("/api/comment", { parent: props._id }).then((comments) => {
    setComments(comments);
  });
}, []);
```

How can we access this from the backend? (Hint: req)

req.query.parent

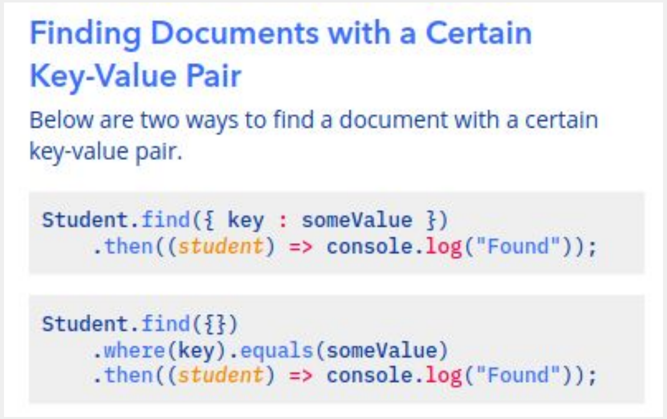
Your turn! Implement `GET /comment`

Choose the right parent to use!

Find “`/* input the parent parameter here */`” in the code and put your response there.

Hint 1: req.query has the content of the get request

Hint 2: weblab.is/mongo-snippets has hints on how to filter



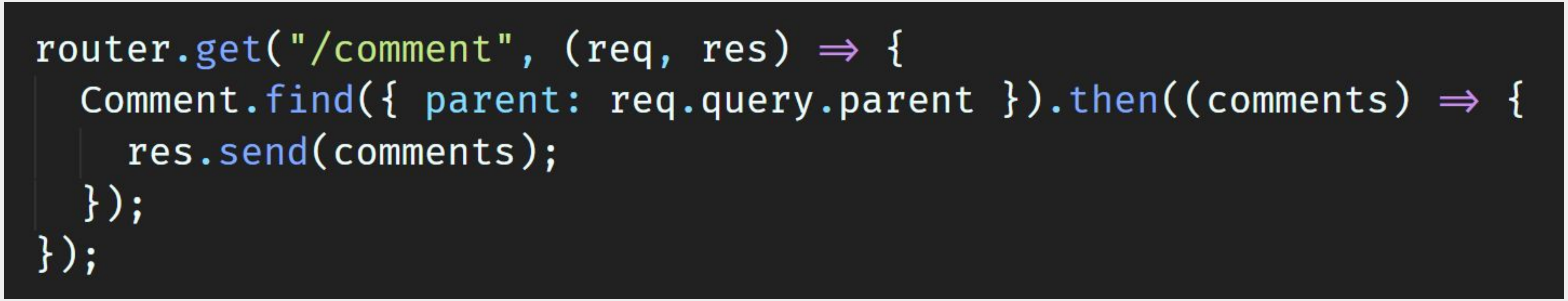
Finding Documents with a Certain Key-Value Pair

Below are two ways to find a document with a certain key-value pair.

```
Student.find({ key : someValue })
    .then((student) => console.log("Found"));
```

```
Student.find({})
    .where(key).equals(someValue)
    .then((student) => console.log("Found"));
```

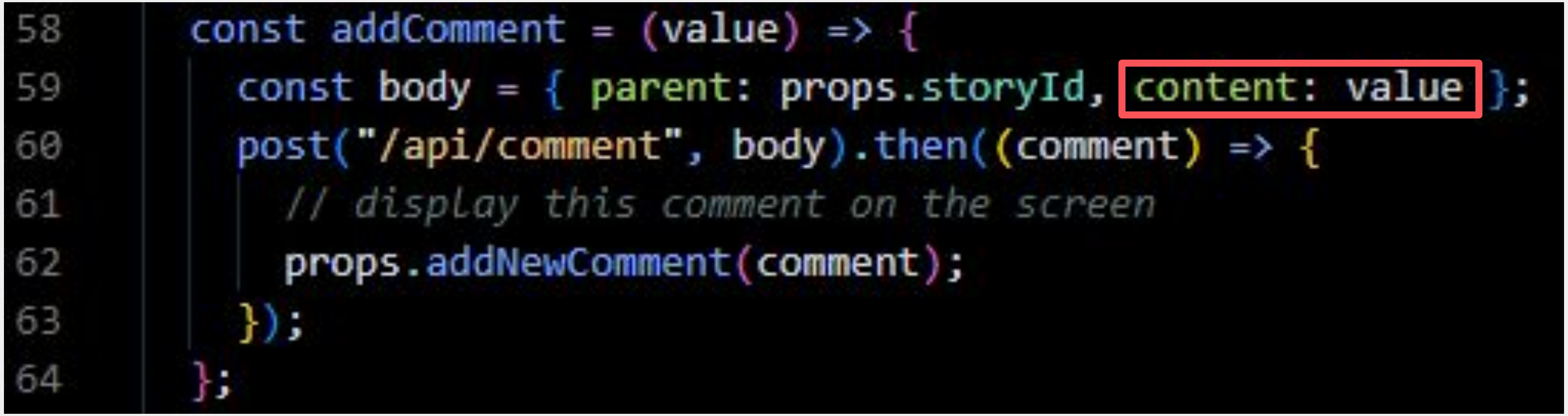
GET /comment (solution)



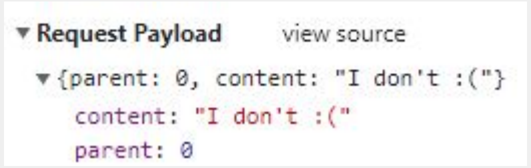
```javascript
router.get("/comment", (req, res) => {
  Comment.find({ parent: req.query.parent }).then((comments) => {
    res.send(comments);
  });
});
```

The POST body

From `NewPostInput.js`:



```
const addComment = (value) => {
  const body = { parent: props.storyId, content: value };
  post("/api/comment", body).then((comment) => {
    // display this comment on the screen
    props.addNewComment(comment);
  });
};
```



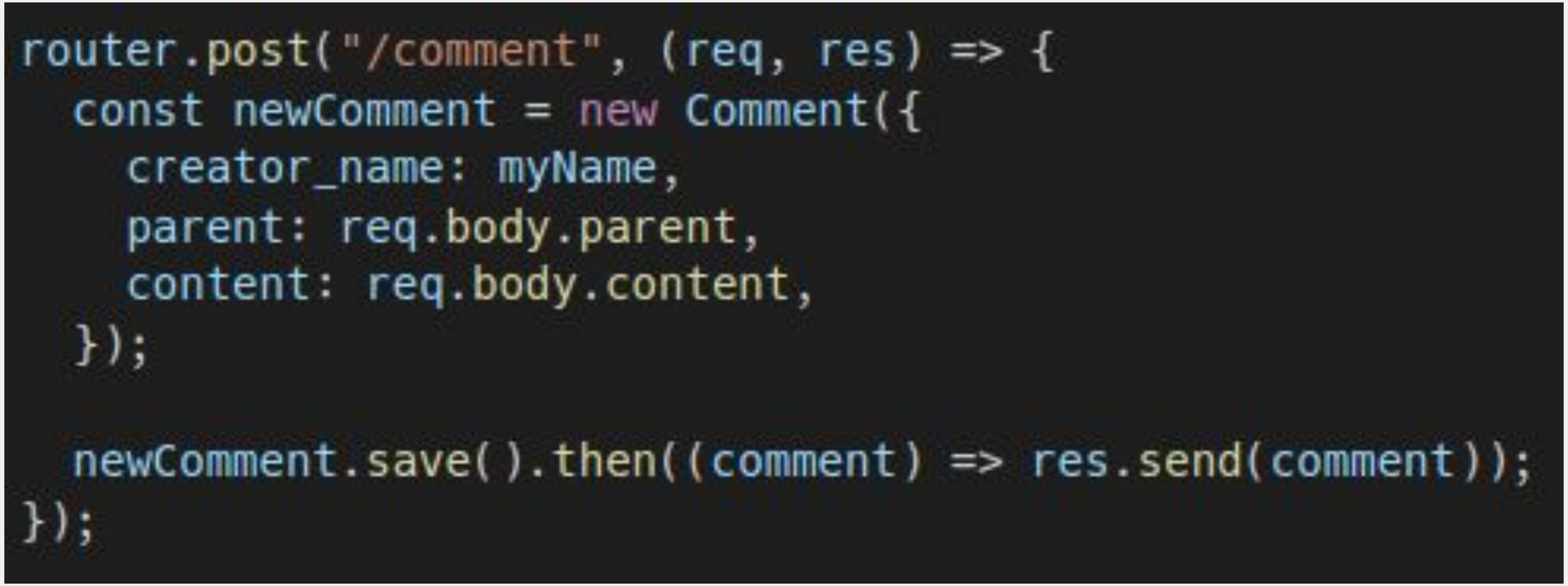
```
▼ Request Payload      view source
  ▼ {parent: 0, content: "I don't :("}
      content: "I don't :("
      parent: 0
```

Your turn! Implement `POST /comment`

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

Hint 1: Look at `POST /story` and weblab.is/mongo-snippets

Part 5: POST /comment (solution)



```
router.post("/comment", (req, res) => {
  const newComment = new Comment({
    creator_name: myName,
    parent: req.body.parent,
    content: req.body.content,
  });

  newComment.save().then((comment) => res.send(comment));
});
```

Let’s test post a comment!

In one terminal:

```
npm start
```

In *another* terminal:

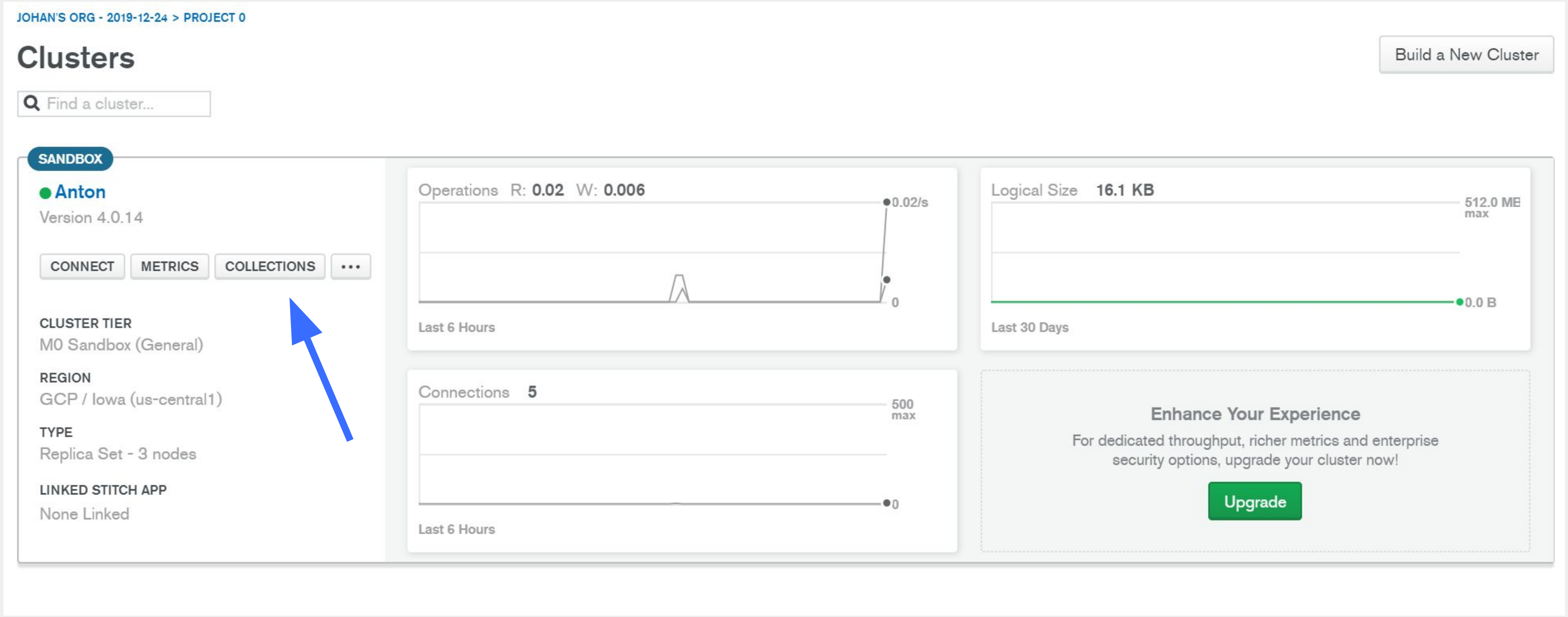
```
npm run hotloader
```

… and go check `localhost:5050` in your browser!

```
git reset --hard
git checkout w6-complete
```

Recopy your SRV into server.js

Testing!



JOHAN'S ORG - 2019-12-24 > PROJECT 0

Clusters

Build a New Cluster

Find a cluster...

SANDBOX

● Anton

Version 4.0.14

CONNECT METRICS COLLECTIONS •••

CLUSTER TIER
M0 Sandbox (General)

REGION
GCP / Iowa (us-central1)

TYPE
Replica Set - 3 nodes

LINKED STITCH APP
None Linked

Operations R: 0.02 W: 0.006
0.02/s
0
Last 6 Hours

Logical Size 16.1 KB
512.0 MB max
0.0 B
Last 30 Days

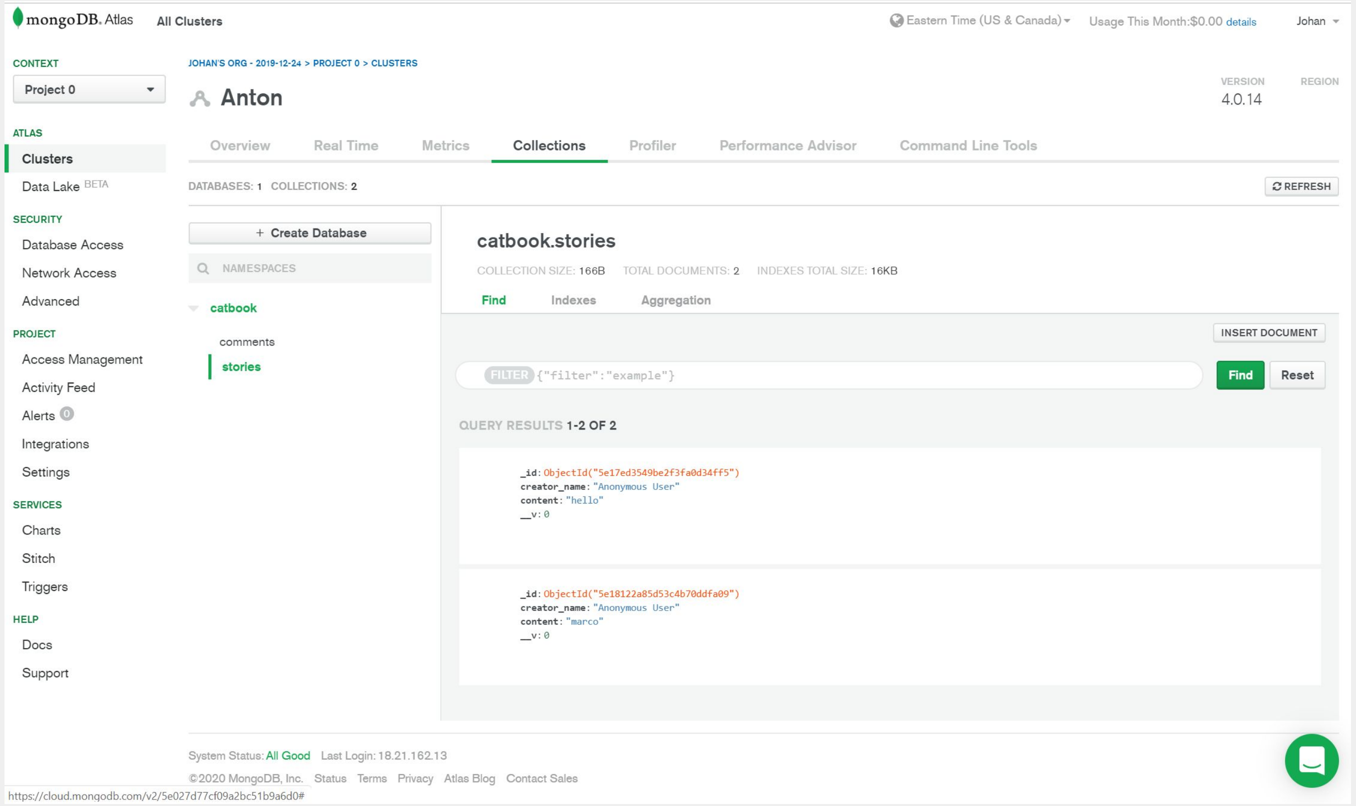
Connections 5
500 max
0
Last 6 Hours

Enhance Your Experience

For dedicated throughput, richer metrics and enterprise security options, upgrade your cluster now!

Upgrade

Testing!



```
_id: ObjectId("5e17ed3549be2f3fa0d34ff5")
creator_name: "Anonymous User"
content: "hello"
__v: 0
```

```
_id: ObjectId("5e18122a85d53c4b70ddfa09")
creator_name: "Anonymous User"
content: "marco"
__v: 0
```

System Status: All Good Last Login: 18.21.162.13
©2020 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales
https://cloud.mongodb.com/v2/5e027d77cf09a2bc51b9a6d0#

Recap

We learned to:

- Understand database structure, schemas, models
- Hook remote mongodb instances to our nodejs app
- Interact with database via an api
- Use that api in the frontend

THAT'S IT!

Mongoose Documentations & Further Readings

MongoDB Documentations: https://docs.mongodb.com

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

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

Atlas documentation: https://docs.atlas.mongodb.com/import/

Now, catbook can go w e b s c a l e