Cubstart Lecture 8

Backend Continued

[start recording]

Administrivia

- HW 7: Quizlet-ish Part 2 due on Friday
- HW 8 will be due next Friday

Please check HW feedback and HW Ed posts!!!!

Administrivia: Final Project

- Team formation form extended to this Fri
  - https://forms.gle/2yAcxgQ67EUCkiFo8
- Use Partner Search Thread pinned on Ed!
  - https://edstem.org/us/courses/45098/discussion/3727491
- Draft Spec for Final Project released:
  - https://www.cubstart.com/#/hw/web/spec
  - Will be revised & finalized in the next few days

Administrivia: Final Project

- Part 1: Checkpoint due next Friday
  - Project proposal + design mockup
  - Will talk more in lab
- Part 2: Development + Presentation
  - Demo day will tentatively be Fri, Dec 1, 4-6 pm (lab)
  - Project due same week
- We'll focus on the project in the next few weeks to support you all!

Backends

Quick Poll:
Servers & Databases

Tools we use: Servers

Node.js: runs your JavaScript server code

npm: allows you to manage dependencies or packages to use in your server

express.js: a framework (“structure” and “toolkit”) for writing API servers

Middleware: pieces of code that we use in the API server that processes request/response

(i.e. body-parser for parsing POST request bodies)

Tools we use: Databases

MongoDB: document-based NoSQL database to store/retrieve data (as JSON)

Mongoose: JavaScript “Object Data Modeling” library to work with MongoDB

Project Setup

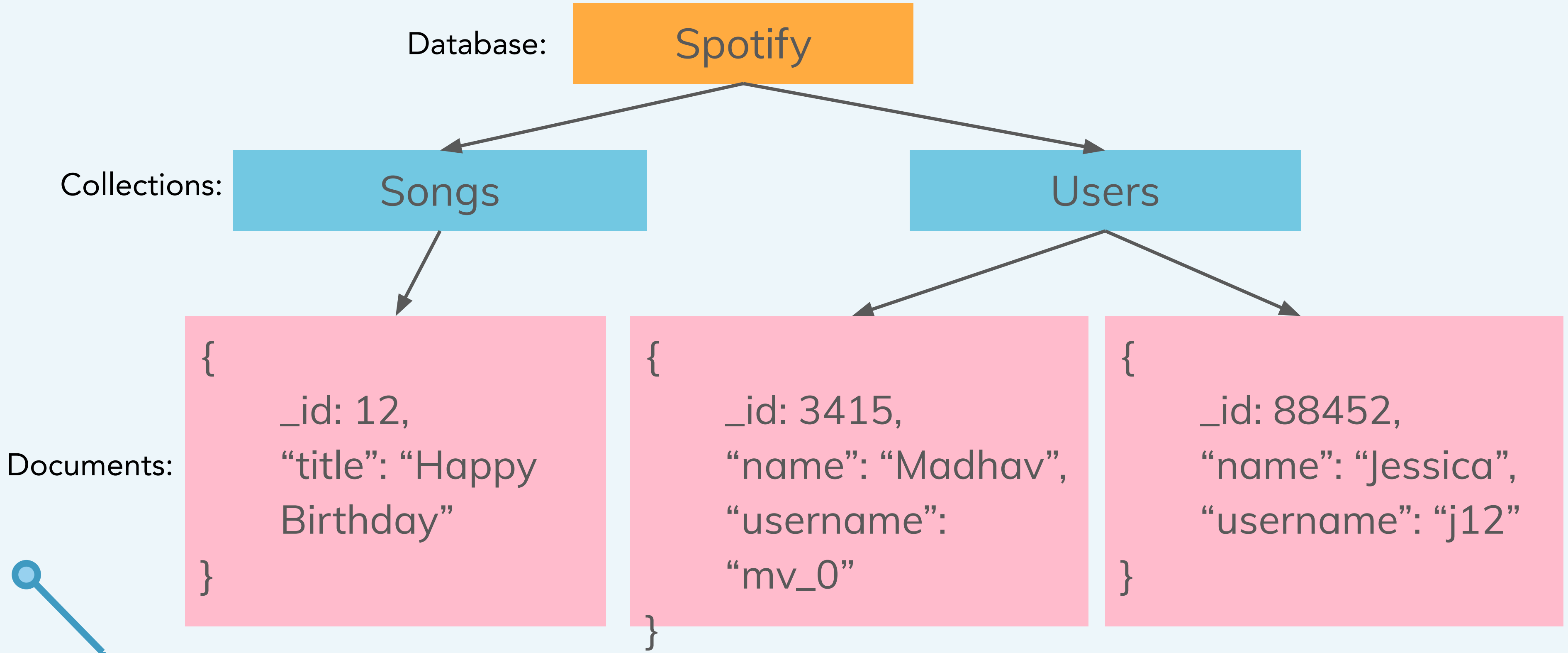
npm init to create npm project with a package.json

npm install express mongoose <...other packages>

create server.js, setup express.js app

Databases

Example: Spotify

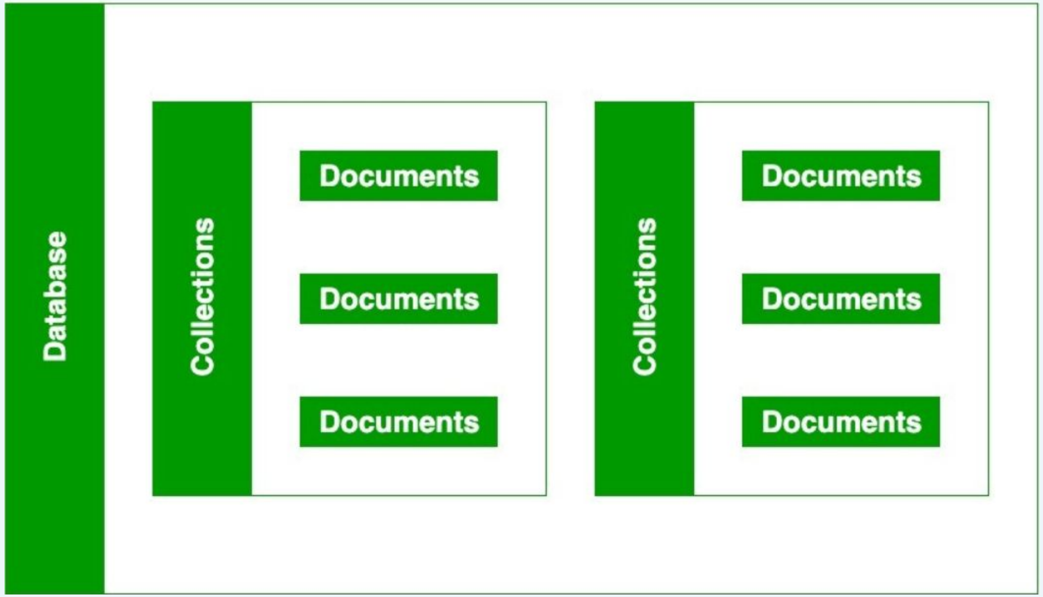


```
{
	_id: 12,
	"title": "Happy Birthday"
}
```

```
{
	_id: 3415,
	"name": "Madhav",
	"username": "mv_0"
}
```

```
{
	_id: 88452,
	"name": "Jessica",
	"username": "j12"
}
```

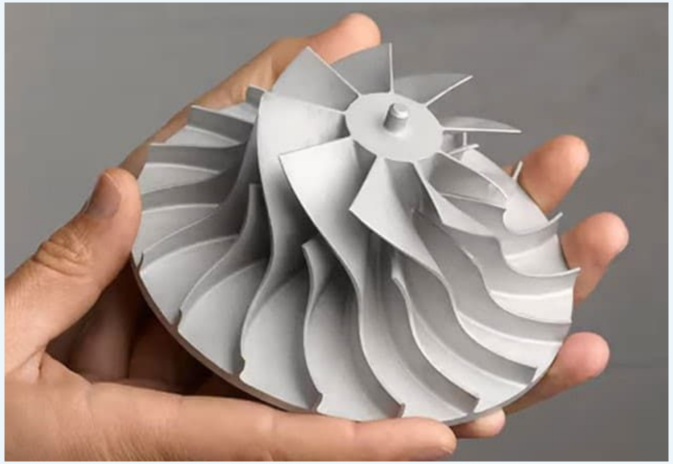
How is data stored in MongoDB?



Analogy: Shelf -> Binders -> Pages

Mongoose overview

mongoose



$$\begin{tikzcd}
\text{Blueprint} \arrow[r] & \text{Prototype} \arrow[r] & \text{Actual Thing!} \\
\text{Schema} \arrow[r] & \text{Model} \arrow[r] & \text{Document}
\end{tikzcd}$$

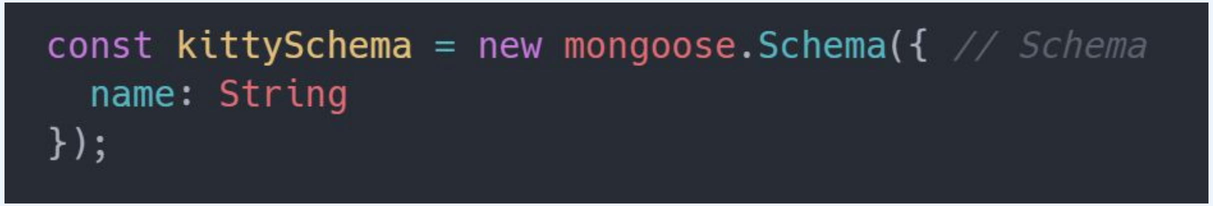
Mongoose overview

mongoose

Blueprint

Schema

Schemas define the structures and properties of a MongoDB document



```
const kittySchema = new mongoose.Schema({ // Schema
  name: String
});
```

Each key in our code `kittySchema` defines a property in our document which will be cast to its associated SchemaType. For example, we've defined a property `name` which will be cast to the String SchemaType.

Mongoose overview

mongoose

Blueprint

Schema

Schemas define the structures and properties of a MongoDB document

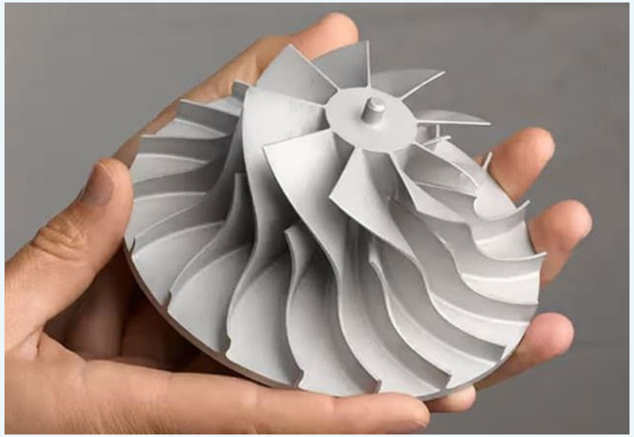
```
const kittySchema = new mongoose.Schema({ // Schema
  name: String
});
```

Here are a few permitted SchemaTypes:

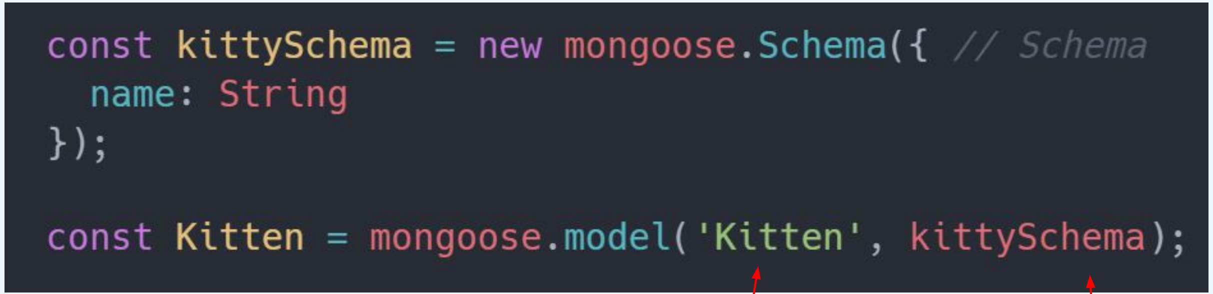
- String
- Number
- Boolean
- Array

Mongoose overview

mongoose



Models are compiled versions of Schemas that handle database operations such as creating, querying, updating, and deleting data in a collection



```
const kittySchema = new mongoose.Schema({ // Schema
  name: String
});

const Kitten = mongoose.model('Kitten', kittySchema);
```

Collection name

Schema name

Prototype

Model

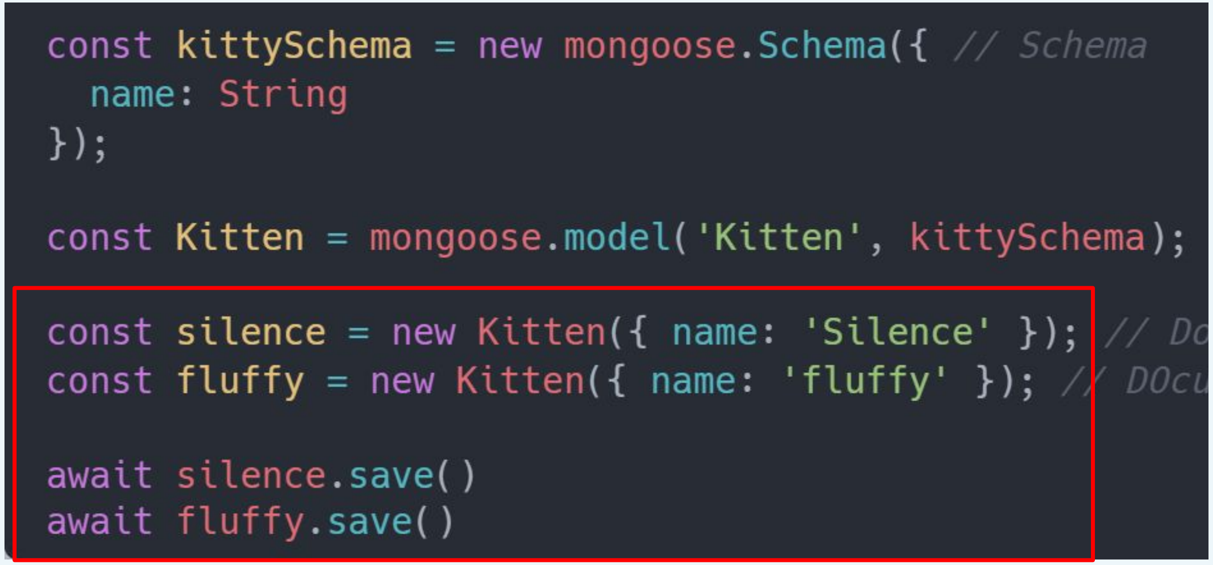
Mongoose overview

mongoose

Actual Thing!

Document

Documents store your actual data! They are instances of your model.



```
const kittySchema = new mongoose.Schema({ // Schema
  name: String
});

const Kitten = mongoose.model('Kitten', kittySchema);

const silence = new Kitten({ name: 'Silence' }); // Do
const fluffy = new Kitten({ name: 'fluffy' }); // DOcu

await silence.save()
await fluffy.save()
```

Demo

Live: database.js

```
const mongoose = require('mongoose')

const songSchema = new mongoose.Schema({
    title: String,
    artist: String,
    duration: Number
})

const Song = mongoose.model('songs', songSchema)
```

database.js (part 1 - create schema & model)

```
async function createSong(title, artist, duration) {
    const newSong = new Song({
        title: title,
        artist: artist,
        duration: duration,
    })

    await newSong.save()
}
```

database.js (part 2 - creating a document)

```
async function findSongs() {
    // could pass in object into .find() to filter based on keys
    // (i.e. find a particular song by title or duration)
    const allSongs = await Song.find()
    return allSongs
}
```

database.js (part 3 - finding documents)

```javascript
async function connectToDatabase() {
    await mongoose.connect('mongodb+srv://...<uri>...')
    console.log('connected to DB!')
}

module.exports = { createSong, findSongs, connectToDatabase }
```

database.js (part 4 - connecting to DB & exports)

Live: server.js

```
const express = require('express')
const bodyParser = require('body-parser')
const database = require('./database.js')

const app = express()

app.use(bodyParser.json()) // read POST body as JSON and put in req.body

app.get("/songs", async (req, res) => {
    const songs = await database.findSongs()
    res.json(songs)
})
```

server.js (part 1)

```
app.post("/songs", async (req, res) => {
    await database.createSong(req.body.title, req.body.artist, req.body.duration)
    res.json({ 'message': 'song created successfully' })
})

database.connectToDatabase().then(() => {
    console.log('database connected...')

    app.listen(3000, () => {
        console.log('server started!')
    })
})
```

server.js (part 2)

Live: client-side

Creating a song: POST /songs

```
await fetch('/songs', {
    method: 'POST',
    body: JSON.stringify({
        title: "Heal the World",
        artist: "Michael Jackson",
        duration: 3
    }),
    headers: {
        "Content-Type": "application/json"
    }
})
```

[end recording]

Attendance: Lecture 8

https://forms.gle/LAAZ28LAEzEcpfP59

Secret word: