

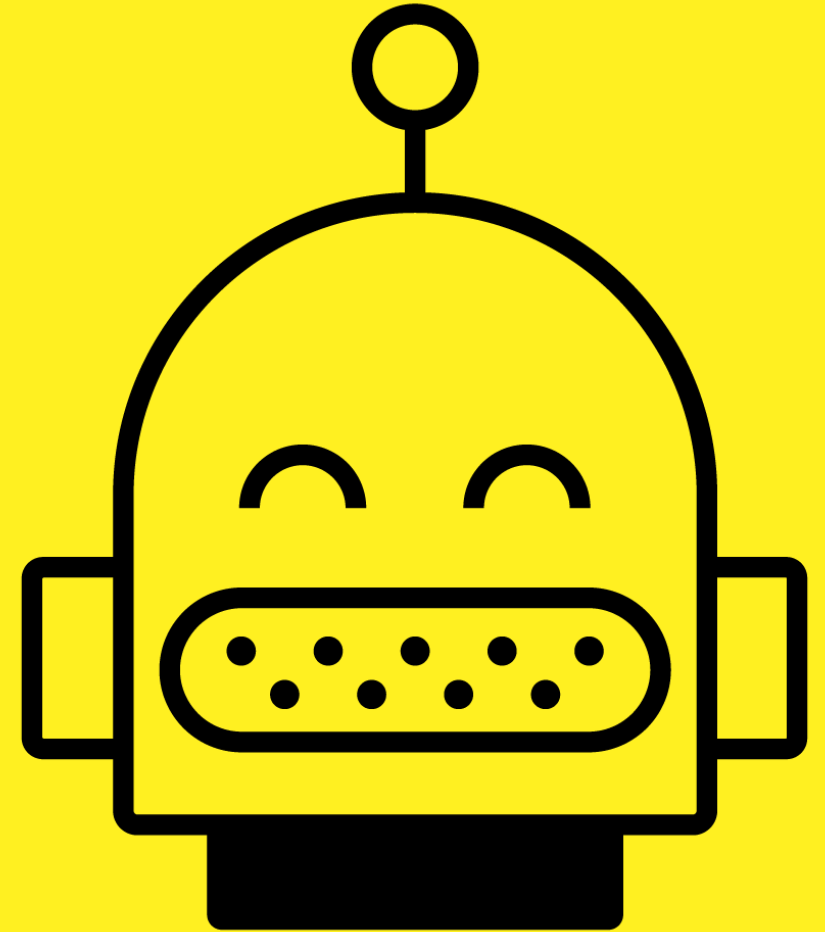


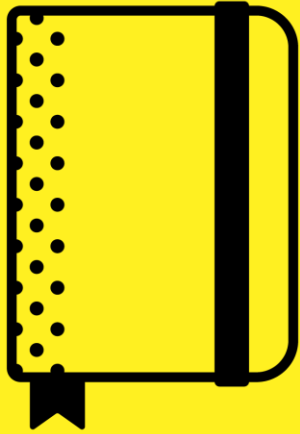
Project Tech

Verdieping Backend #2

Forms & Database

Collegejaar 23/24





Recap theorie

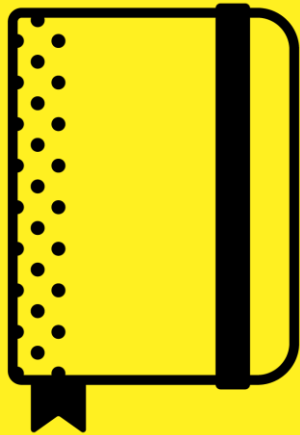
Routing & templating



Laat je server code zien



Welke vragen zijn er nog?



Theorie

Posting Forms

localhost:8000/add

Add a new movie

Title

Plot

Description

Add Movie

view/add.ejs

```
<%- include('head.ejs') %>

<h1>Add a new movie</h1>
<form action="/add-movie" method="POST">
  <label>Title</label>
  <input type="text" name="title">

  <label>Plot</label>
  <input type="text" name="plot">

  <label>Description</label>
  <textarea name="description"></textarea>

  <input type="submit" value="Add Movie">
</form>
```

Send a HTTP **POST** request to the URL specified in the **ACTION...**

...when the form is submitted

server.js

```
const express = require('express')
const app = express()

app.use(express.urlencoded({extended: true}))

app.get('/add', showAddForm)
app.post('/add-movie', addMovie)

function showAddForm(req, res) {
  res.render('add.ejs')
}

function addMovie(req, res) {
  res.send(`Thanks for adding the movie with:
    title: ${req.body.title},
    plot: ${req.body.plot},
    and description: ${req.body.description}
  `)
}
```

Middleware: parses form data

Route to handle the **post** request to **/add-movie**

Parsed form data is stored in **req.body**

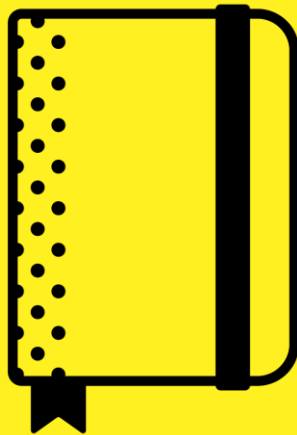
The properties of **req.body** match the **name** attributes of the inputs in the form



Opdracht: formulier posten

Breid je `node.js` server uit met code om gepost formulier te verwerken. Je kunt het inlogformulier uit de vorige les gebruiken, maar haal dan eerst de front-end JavaScript er uit. We gaan deze functionaliteit nu immers in de back-end bouwen!

1. Maak een route en een view om je formulier te tonen
2. Geef het formulier een action en een POST-method
3. Maak een route om de POST request af te handelen als het formulier wordt verstuurd
4. Maak een view om een HTTP response op de POST request te sturen. Laat in deze view de ontvangen formulier data zien.



Theorie

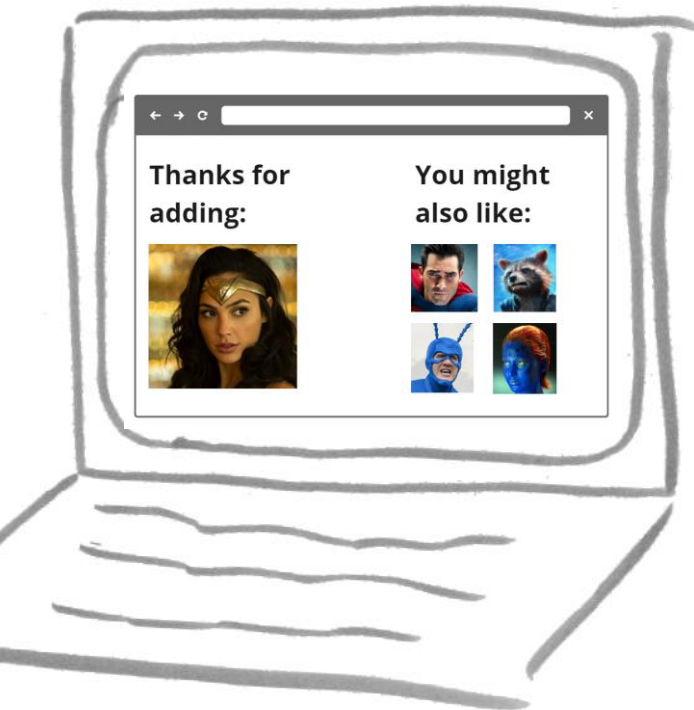
Database

Storing data in db

Client
(browser)

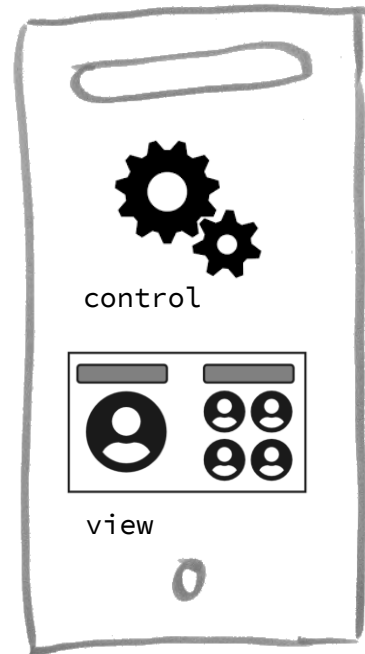
Server
(Webserver met Node.js)

Database
(MongoDB)



1. Browser: here's
form data about a
new movie

HTTP POST request



2. Server: please
store this info



3. Database: I did!



4. Server: now get me
some related movies

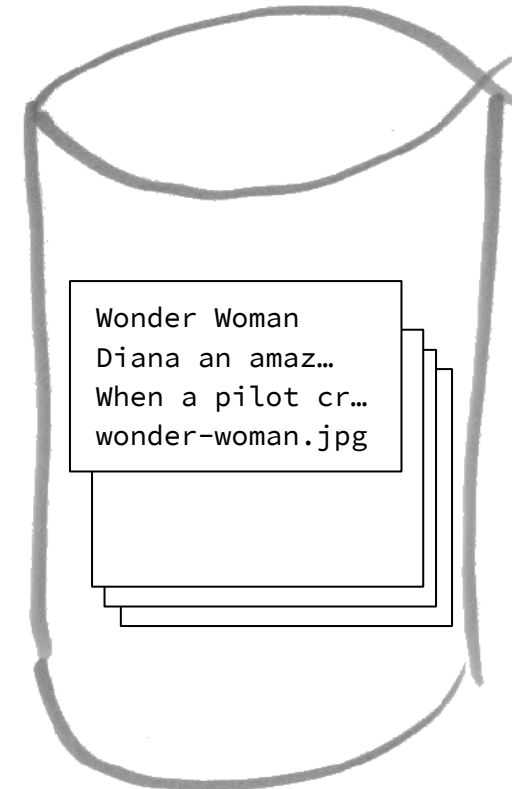


5. Database: here
are the details



Server: here's a
new page

6. HTTP response





MongoDB

MongoDB (from humongous) is a free and **open-source** cross-platform document-oriented database program. Classified as a **NoSQL** database program, MongoDB uses **JSON-like documents** with schemas. MongoDB is developed by MongoDB Inc. [...]

(wikipedia.org)



Waarom MongoDB?

- Het werkt goed samen met JavaScript / Node.js
- Je kunt je data flexibel structureren op een JSON-achtige manier, zonder SQL of formeel database ontwerp te hoeven leren
- Het is gratis ;)

DATABASES: 5 COLLECTIONS: 17

REFRESH

+ Create Database

Q NAMESPACES

- ▶ sample_airbnb
- ▶ sample_geospatial
- ▼ sample_mflix
 - comments
 - movies**
 - sessions
 - theaters
 - users
- ▶ sample_training
- ▶ sample_weatherdata

sample_mflix.movies

COLLECTION SIZE: 61.82MB TOTAL DOCUMENTS: 45993 INDEXES TOTAL SIZE: 37.95MB

Find Indexes

INSERT DOCUMENT

FILTER {"filter":"example"}

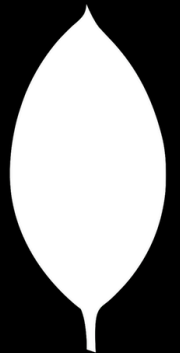
Find

Reset

QUERY RESULTS 1-20 OF MANY

```
_id: ObjectId("573a1390f29313caabcd4132")
title: "Carmencita"
year: 1894
runtime: 1
> cast: Array
poster: "http://ia.media-imdb.com/images/M/MV5B
plot: "Performing on what looks like a small
fullplot: "Performing on what looks like a
lastupdated: "2015-08-26 00:03:45.040000000"
type: "movie"
> directors: Array
> imdb: Object
> countries: Array
rated: "NOT RATED"
> genres: Array
```

Keys & values



DATABASES: 5 COLLECTIONS: 17 REFRESH

+ Create Database

Q NAMESPACES

- ▶ sample_airbnb
- ▶ sample_geospatial
- ▼ sample_mflix
 - comments
 - movies**
 - sessions
 - theaters
 - users
- ▶ sample_training
- ▶ sample_weatherdata

sample_mflix.movies

COLLECTION SIZE: 61.82MB TOTAL DOCUMENTS: 45993 INDEXES TOTAL SIZE: 37.95MB

Find Indexes

INSERT DOCUMENT

FILTER {"filter":"example"} Find Reset

QUERY RESULTS 1-20 OF MANY

```
_id: ObjectId("573a1390f29313caabcd4132")
title: "Carmencita"
year: 1894
runtime: 1
> cast: Array
poster: "http://ia.media-imdb.com/images/M/MV5P
plot: "Performing on what looks like a small
fullplot: "Performing on what looks like a
lastupdated: "2015-08-26 00:03:45.040000000"
type: "movie"
> directors: Array
```

Keys & values

Think about how you want to structure you data.
This is called **data modelling**.



Maar eerst: .env

- Je inloggegevens voor de database wil je niet in je code zetten
- Als ze op GitHub staan, kan de hele wereld bij je database (en dat gaan ze doen ook)
- Zelfs als ze alleen in een oude commit staan (dat was het fijne van Git)
- Daarom zetten we dit soort gegeven in een apart bestand, genaamd: `.env`
- En zetten `.env` in onze `.gitignore` (!!)
- Ook handig als er iets verandert: een nieuw database account, deployen naar een nieuwe webserver etc. Je hoeft dit dan maar op 1 plek aan te passen.

.env

```
DB_HOST=dbhost.somewhere.com  
DB_NAME=mydatabase  
DB_USERNAME=myusername  
DB_PASSWORD=mypassword  
DB_COLLECTION=mycollection
```

We gebruiken meestal hoofdletters

Geen " " (.env is geen JavaScript)

.gitignore

```
node_modules/  
.DS_Store  
.env
```



Opdracht: database

1. Maak een eigen database account aan op `mongodb.com` volgens de instructies op onze GitHub
2. Maak een `.env` file met je eigen gegevens
3. En zet je `.env` in je `.gitignore`
4. Installeer met `npm install` de modules `dotenv` en `mongodb`
5. Breid je `node.js` server uit met code om de database verbinding te openen

Werkt het niet? Kijk eens of er wat in je terminal staat?

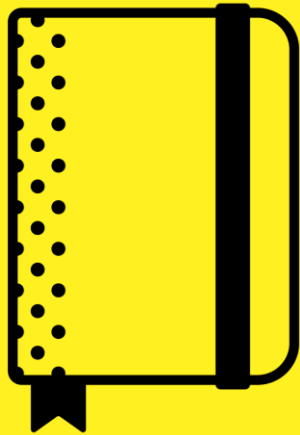
```
require('dotenv').config() // Add info from .env file to process.env

const { MongoClient, ServerApiVersion, ObjectId } = require('mongodb')

// Construct URL used to connect to database from info in the .env file
const uri = `mongodb+srv://${process.env.DB_USERNAME}:${process.env.DB_PASSWORD}@${process.env.DB_HOST}/${process.env.DB_NAME}?retryWrites=true&w=majority`

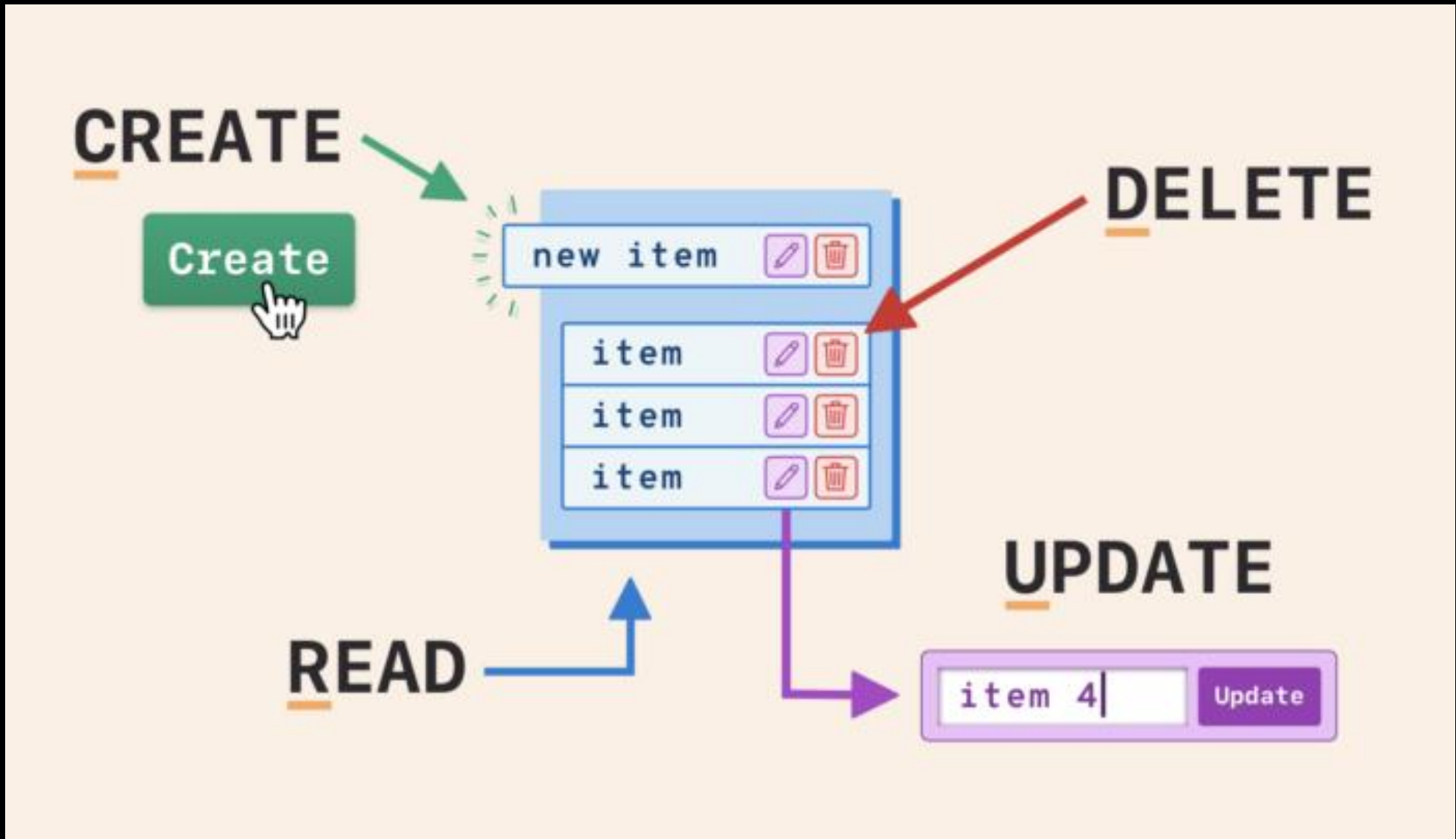
// Create a MongoClient
const client = new MongoClient(uri, {
  serverApi: {
    version: ServerApiVersion.v1,
    strict: true,
    deprecationErrors: true,
  }
})

// Try to open a database connection
client.connect()
  .then((res) => {
    console.log('Database connection established')
  })
  .catch((err) => {
    console.log(`Database connection error - ${err}`)
    console.log(`For uri - ${uri}`)
  })
```

Theorie

Database - CRUD



Source: [CRUD Operations Explained](#) by Avelon Pang, June 14, 2021

find

server.js

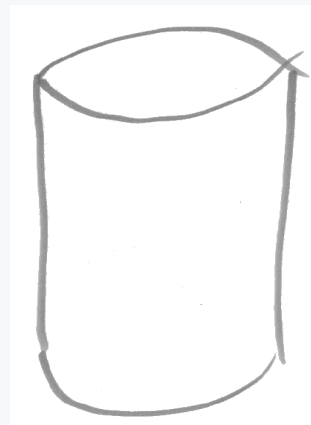
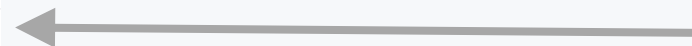
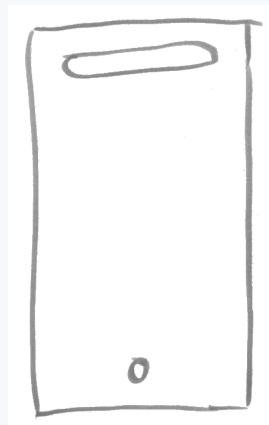
...

```
const db = client.db(process.env.DB_NAME)
const collection = db.collection(process.env.DB_COLLECTION)
```

```
async function listAllMovies(req, res) {
  data = await collection.find().toArray()

  res.render('list.ejs', {data: data})
}
```

...



server.js

...

```
const db = client.db(process.env.DB_NAME)
const collection = db.collection(process.env.DB_COLLECTION)
```

```
async function findMovie(req, res) {
  data = await collection.findOne({
    title: req.params.title
  })

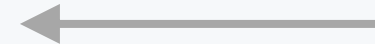
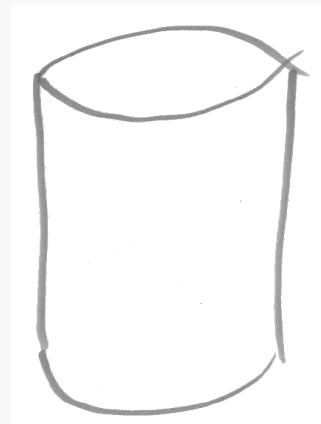
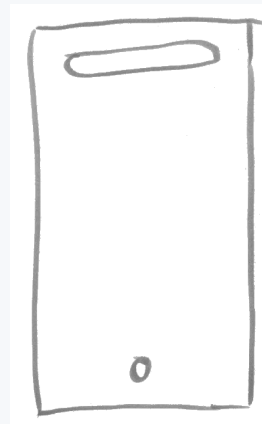
  res.render('detail.ejs', {data: data})
}
```

...

findOne

localhost/movie/wonder_woman

Wonder Woman



server.js

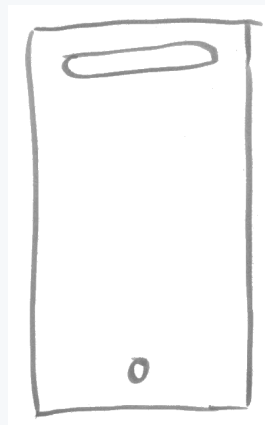
```
const db = client.db(process.env.DB_NAME)
const collection = db.collection(process.env.DB_COLLECTION)

async function addMovie(req, res) {
  result = await collection.insertOne({
    title: req.body.title,
    plot: req.body.plot,
    description: req.body.description
  })

  console.log(`Added with _id: ${result.insertedID}`)
  res.render('added.ejs')
}

...
```

insertOne



update

```
server.js

const db = client.db(process.env.DB_NAME)
const collection = db.collection(process.env.DB_COLLECTION)

async function updateMovie(req, res) {
  const result = await collection.updateOne(
    _id: new ObjectId(req.body.id),
    {$set: {plot: req.body.plot}}
  )

  console.log(`Items found: ${result.matchedCount},
    items updated: ${result.modifiedCount}`)
  res.render('movie_updated.ejs', {id: req.body.id})
}

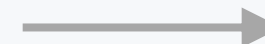
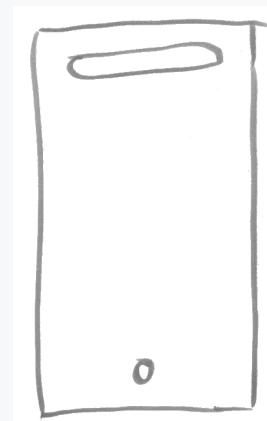
...
```

localhost/update_movie/

ID:

Plot:

Verstuur



server.js

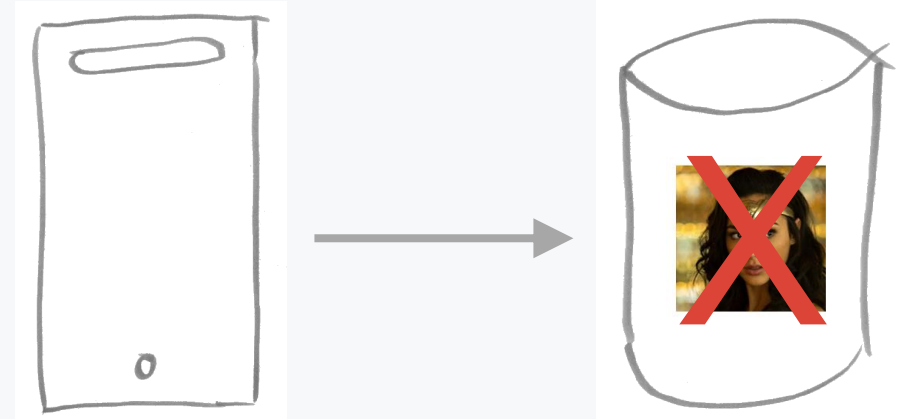
```
const db = client.db(process.env.DB_NAME)
const collection = db.collection(process.env.DB_COLLECTION)

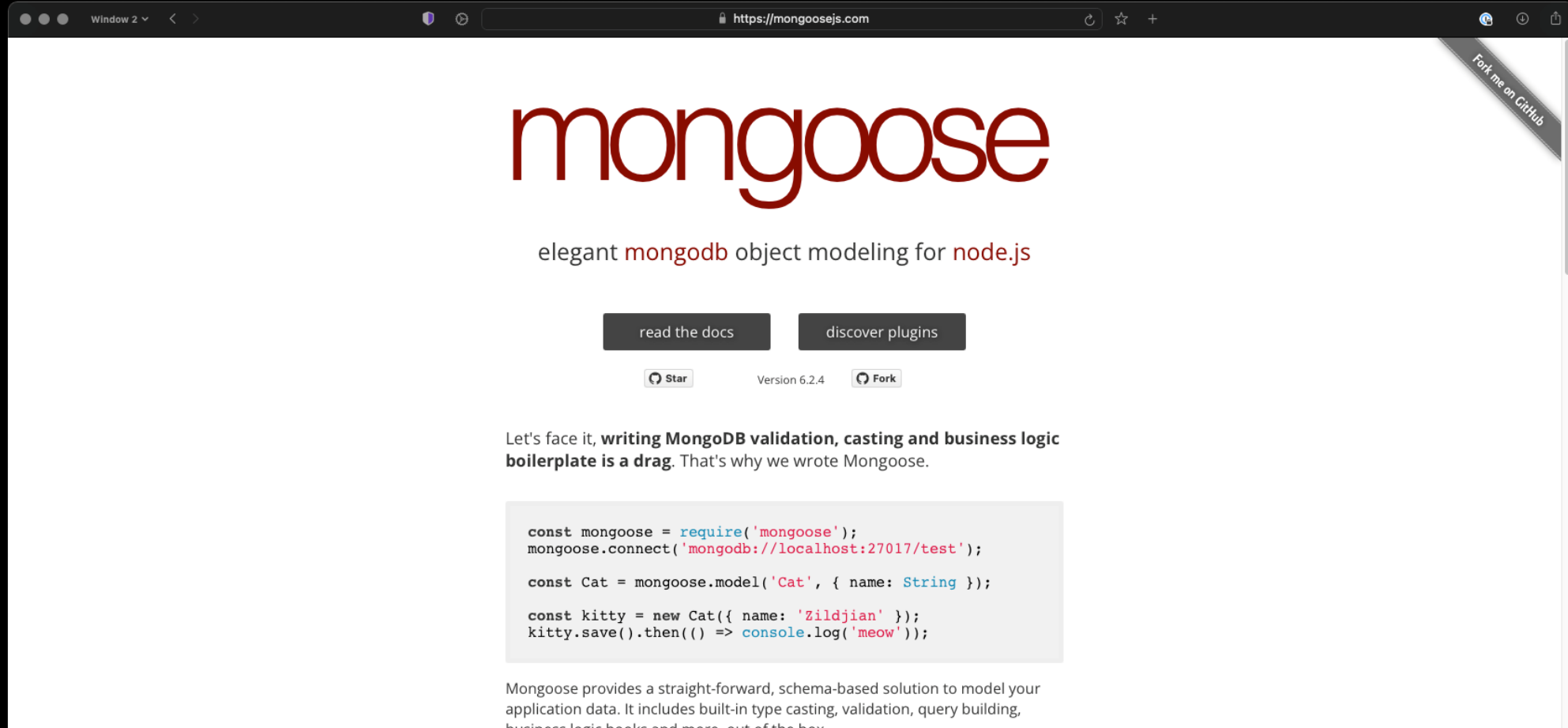
function deleteMovie(req, res) {
  const result = await collection.deleteOne({
    _id: new ObjectId(req.body.id)
  })

  console.log(`Items deleted: ${result.deletedCount}`)
  res.render('movie_deleted.ejs', {id: req.body.id})
}

...
```

delete





Pick **mongoDB (default driver)** over Mongoose.