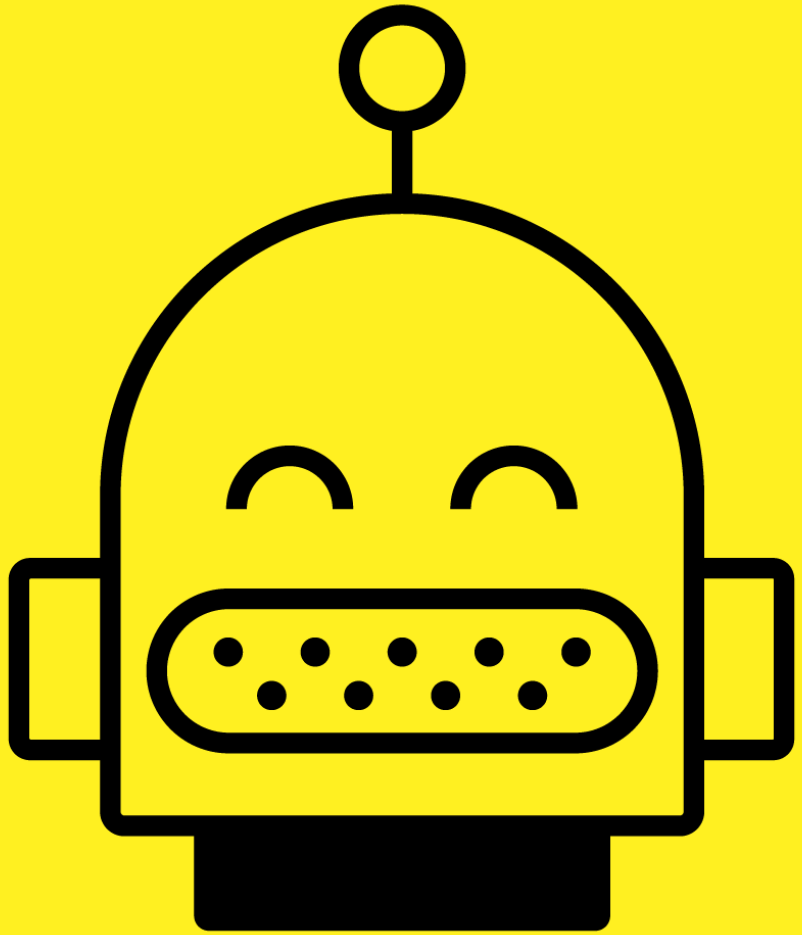
Communication and Multimedia Design | Hogeschool van Amsterdam

Project Tech

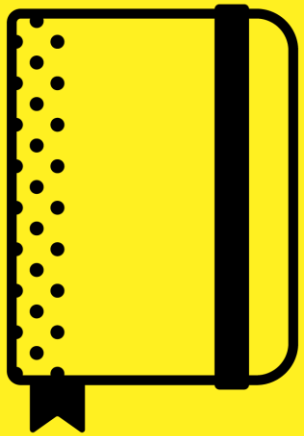
Verdieping Backend #3

Forms & Database



Collegejaar 23/24

Creating Tomorrow

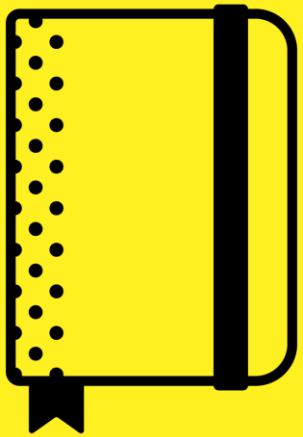


Recap theorie

Node.js

Laat je server code zien

Welke vragen zijn er nog?



Theorie

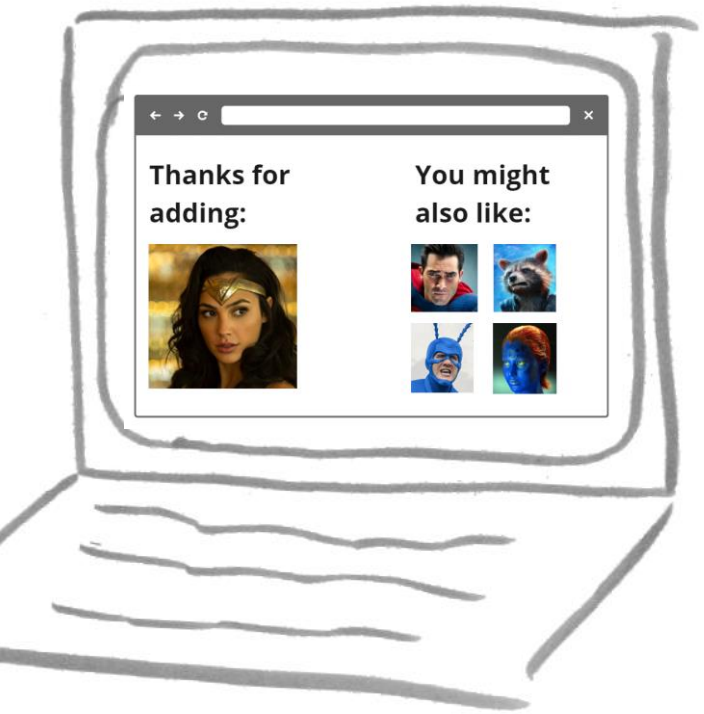
Database - CRUD

Storing data in db

Client
(browser)

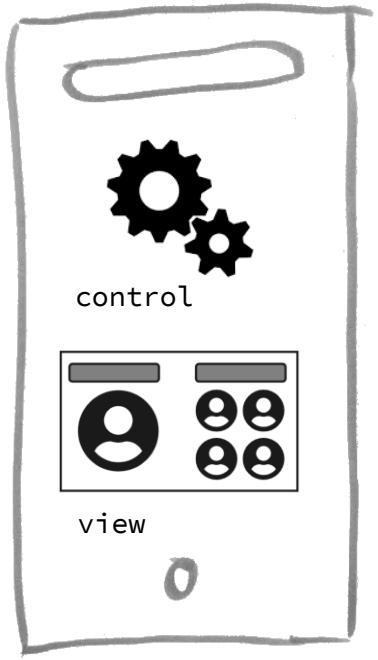
Server
(Webserver met Node.js)

Database
(MongoDB)



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

HTTP POST request

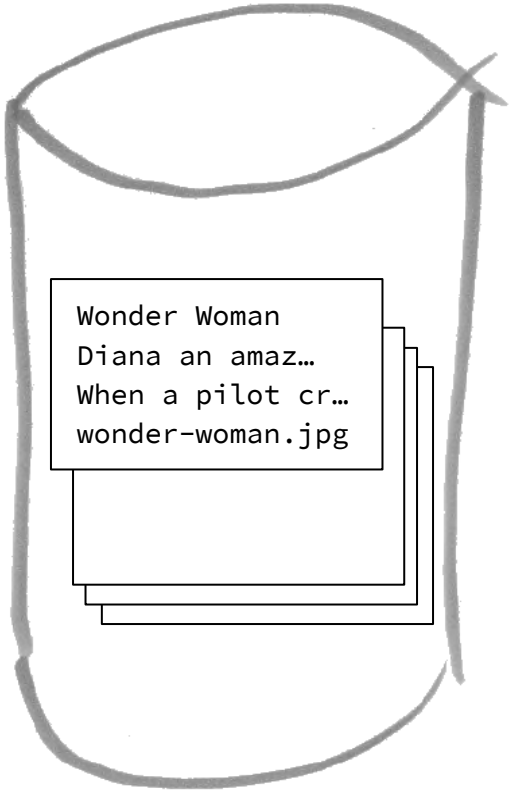


2. Server: please store this info

3. Database: I did!

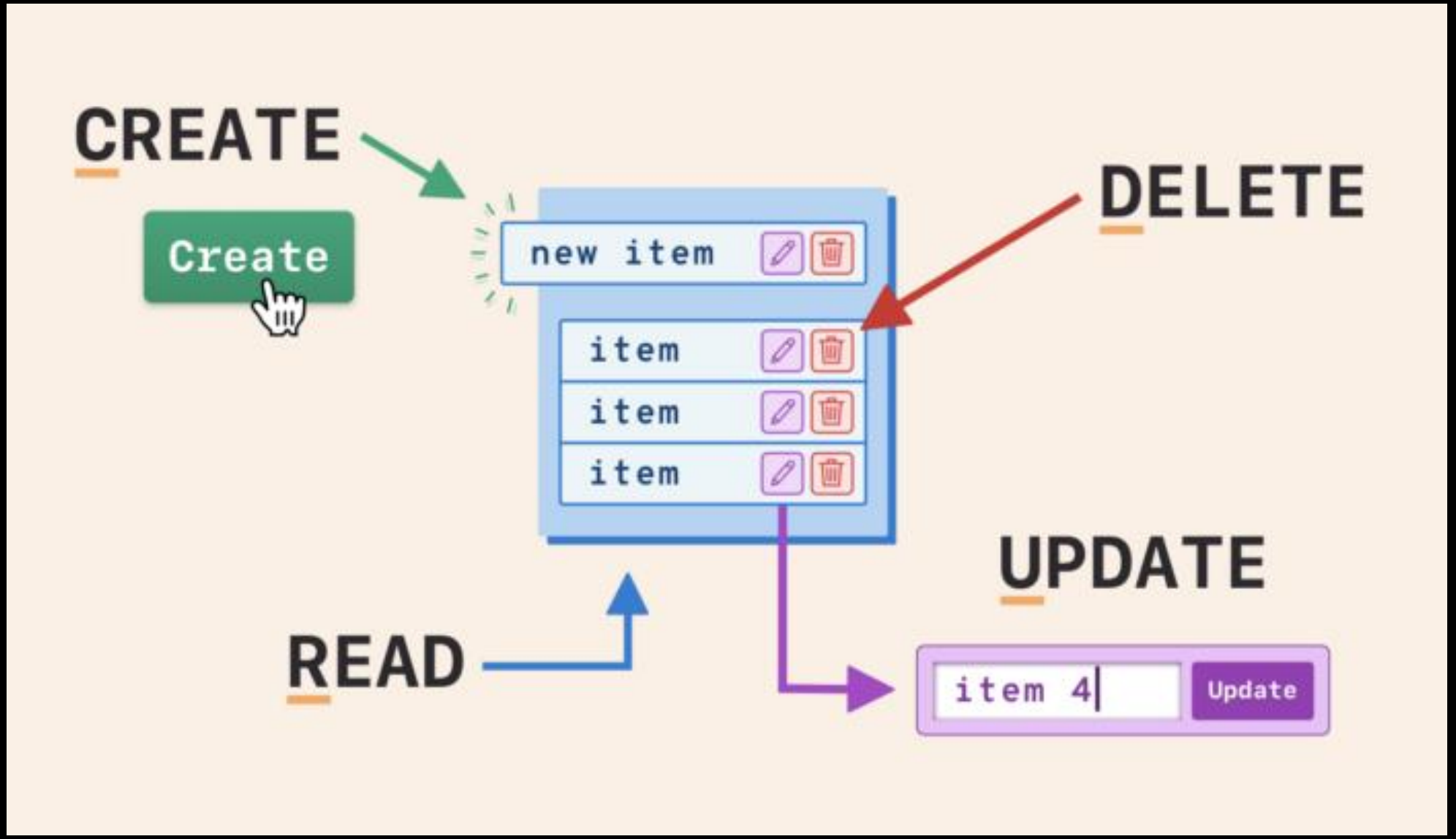
4. Server: now get me some related movies

5. Database: here are the details



6. Server: here's a new page

HTTP response

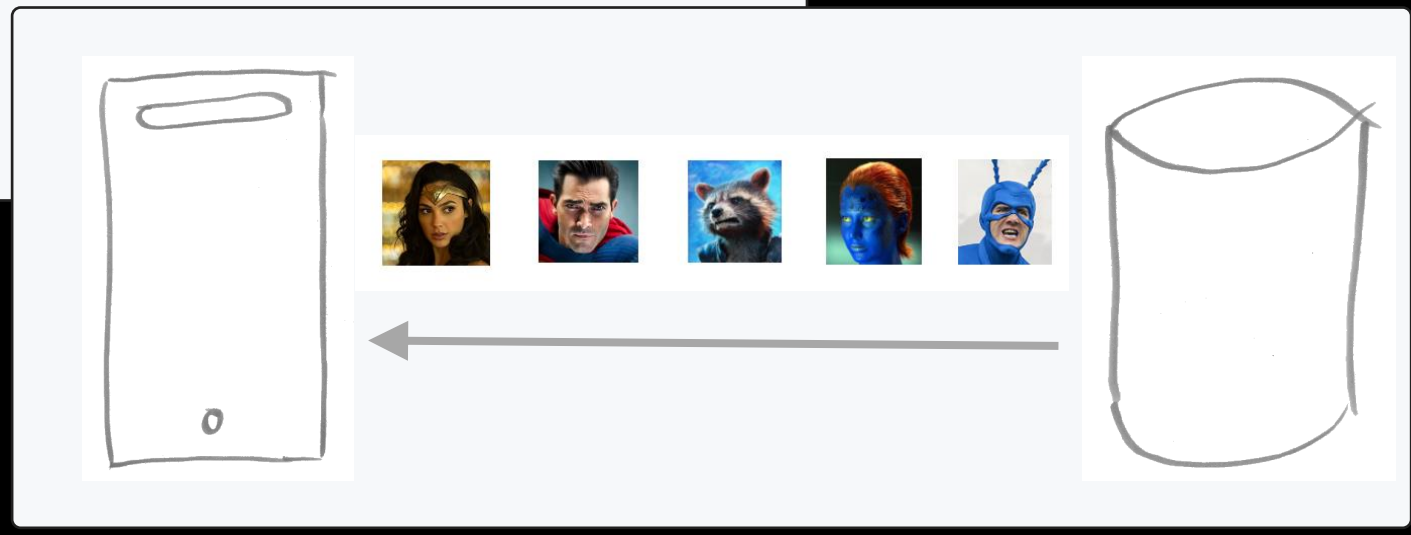


Source: CRUD Operations Explained by Avelon Pang, June 14, 2021

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})
}

…
```



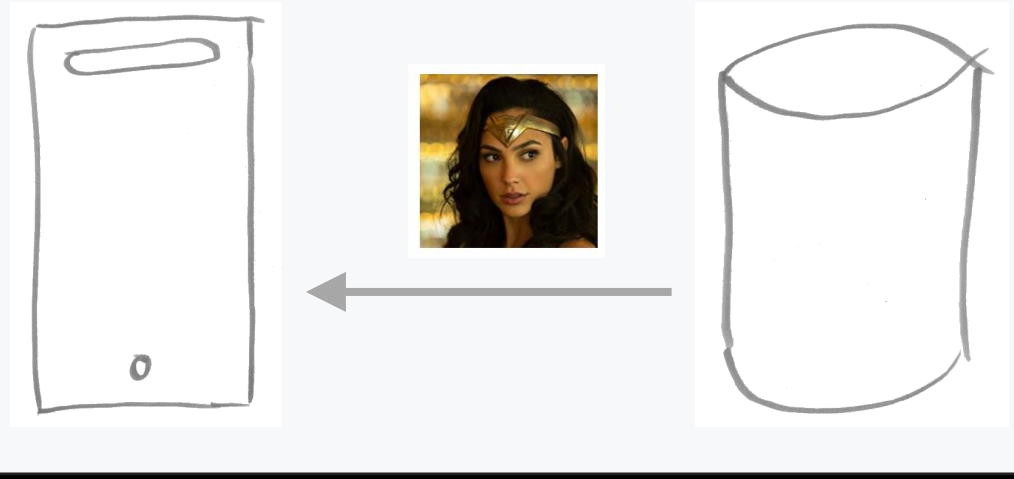
findOne

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})
}

…
```

localhost/movie/wonder_woman

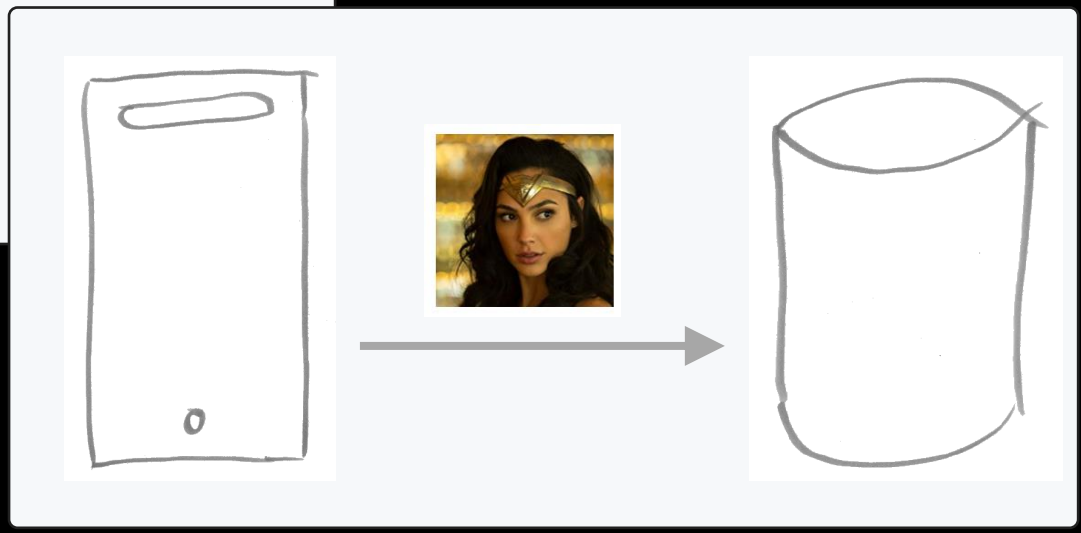
Wonder Woman



insertOne

server.js

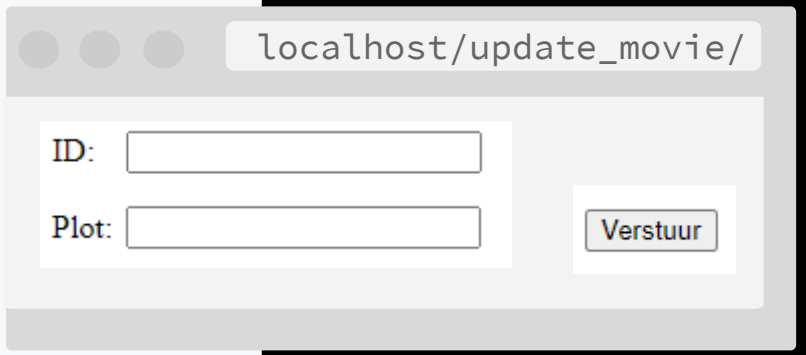
```
const db = client.db(process.env.DB_NAME)
const collection = db.collection(process.env.DB_COLLECTION)

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')
}

…
```

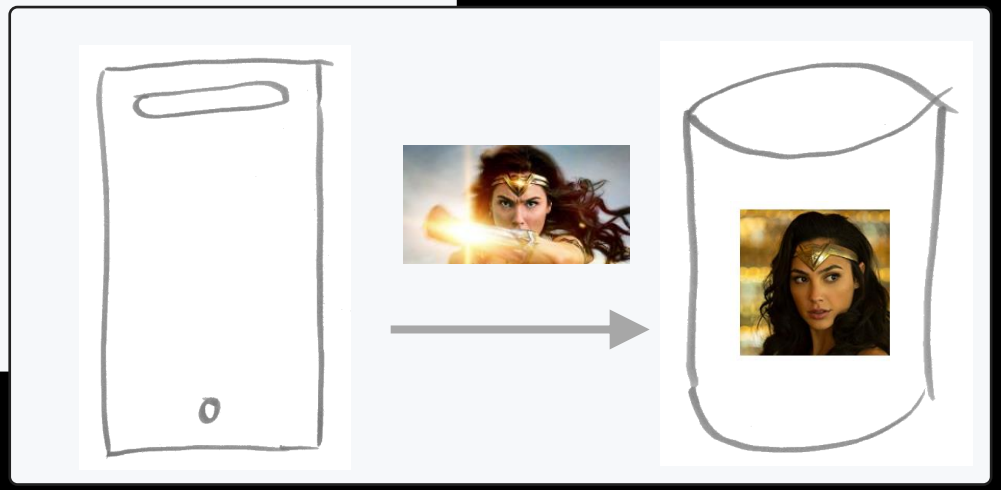


update

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})
}

...
```

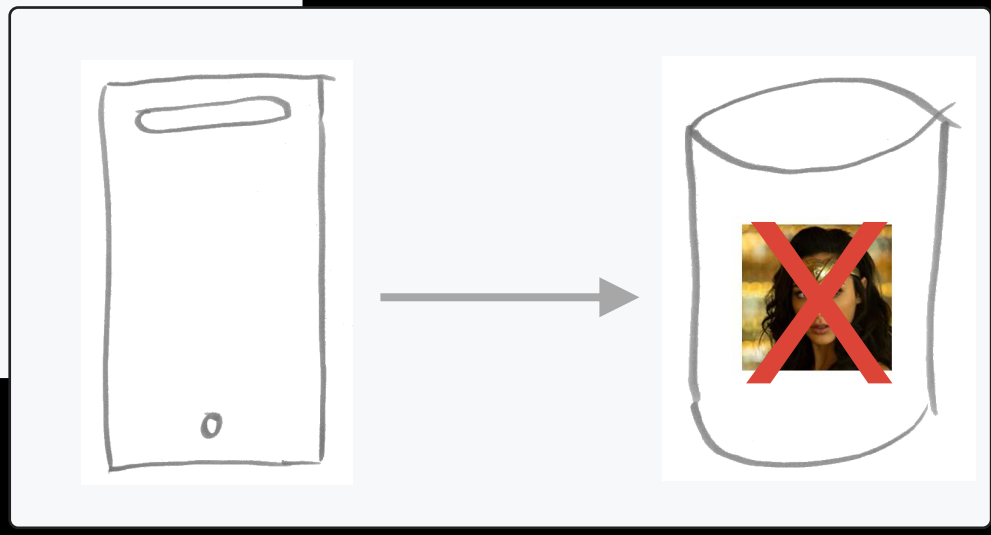


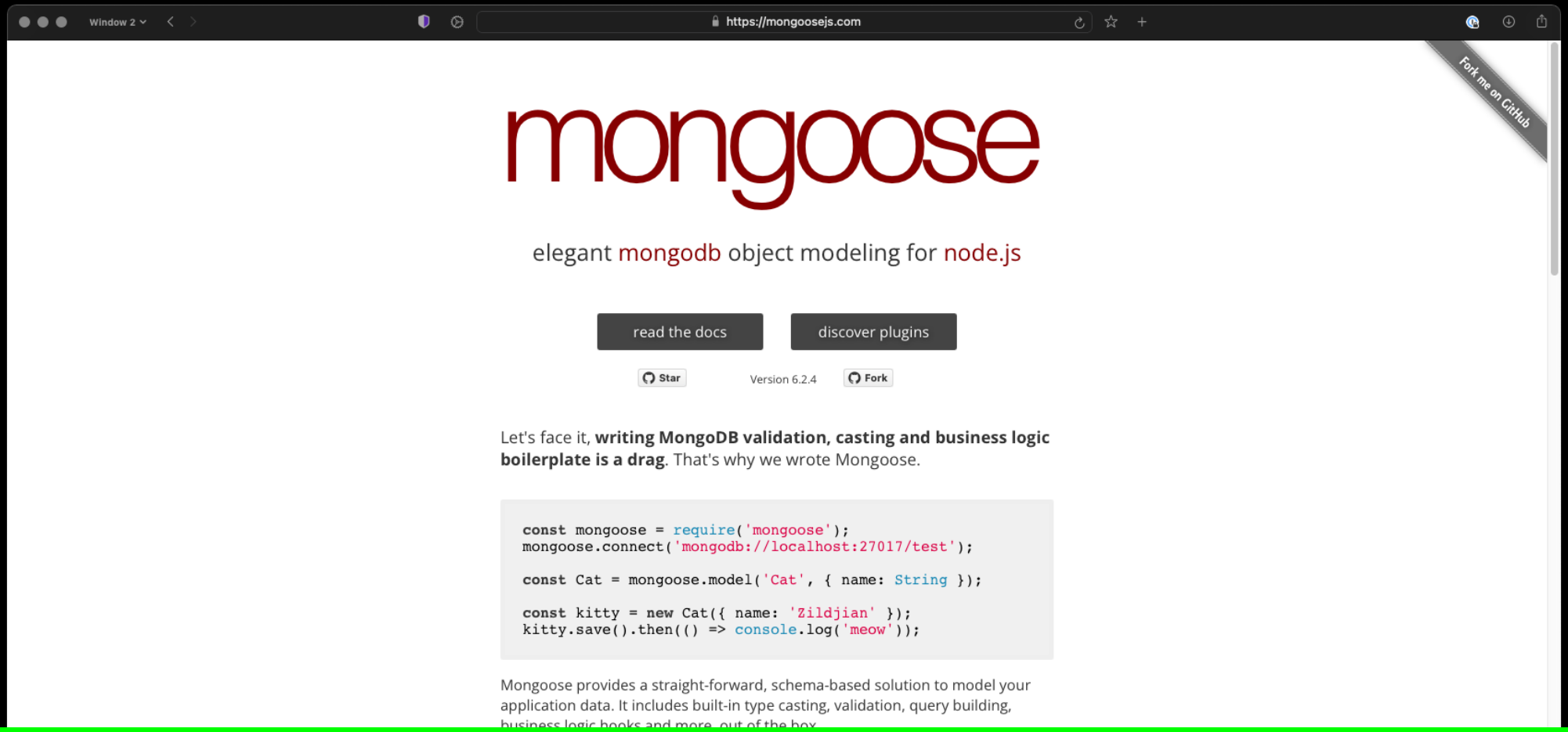


delete

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})
}

…
```





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

Vervolg opdracht: database

1. Maak een eigen MongoDB aan
2. Maak een `.env` file
3. Zet je `.env` in je `.gitignore`
4. Installeer `dotenv` en `mongodb`
5. Breid je `node.js` server uit met code om de database verbinding te openen
6. Zet met de hand wat testdata in je database
7. Voeg een nieuwe route toe aan je webserver, die met een find de testdata ophaalt en weergeeft

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}`)
    })
```

Vervolg 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.

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.
5. Breid deze route uit om de formulier data niet alleen te tonen, maar nu ook op te slaan in je eigen database