Online Multimedia

Winter Semester 2019/20

Tutorial 08 – Databases and Authentication

Today's Agenda

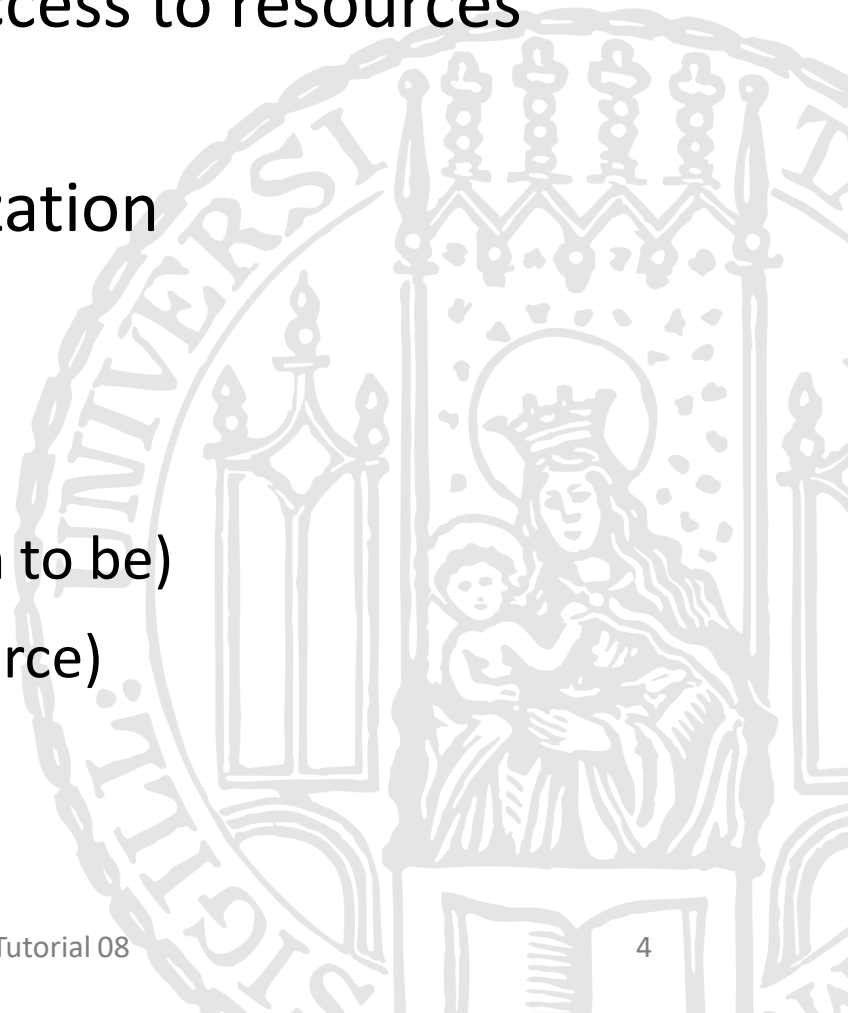
- Authentication and Authorization
- NodeJS + MongoDB

Digital Rights Management: Access Control

Restricted Content

- On many occasions, web apps need to restrict access to resources (e.g. media, sensitive data)
- Usual Solution: User authentication and authorization

- **Differentiation**:
  - Authentication: ≈ Verification (you are who you claim to be)
  - Authorization: ≈ Grant access (allowed to use a resource)



Quick Quiz

Are the following error messages related to authentication or authorization?

1. „Permission to URL denied to user johndoe12“
2. „Comments are disabled for this video“
3. „This site is marked private by its owner“
4. „Sorry, this action requires being logged in.“

Authentication in Web Apps

- Access Tokens, e.g. API Keys
  - URL Parameter
  - HTTP Header
  - HTTP Message Body
- Cookies
  - Contains (authentication) session ID
  - Server handles authorization based on this ID (deserialization)
- Third Party Authentication
  - OAuth
  - OpenID

Basic-Auth Middleware in NodeJS

```
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());

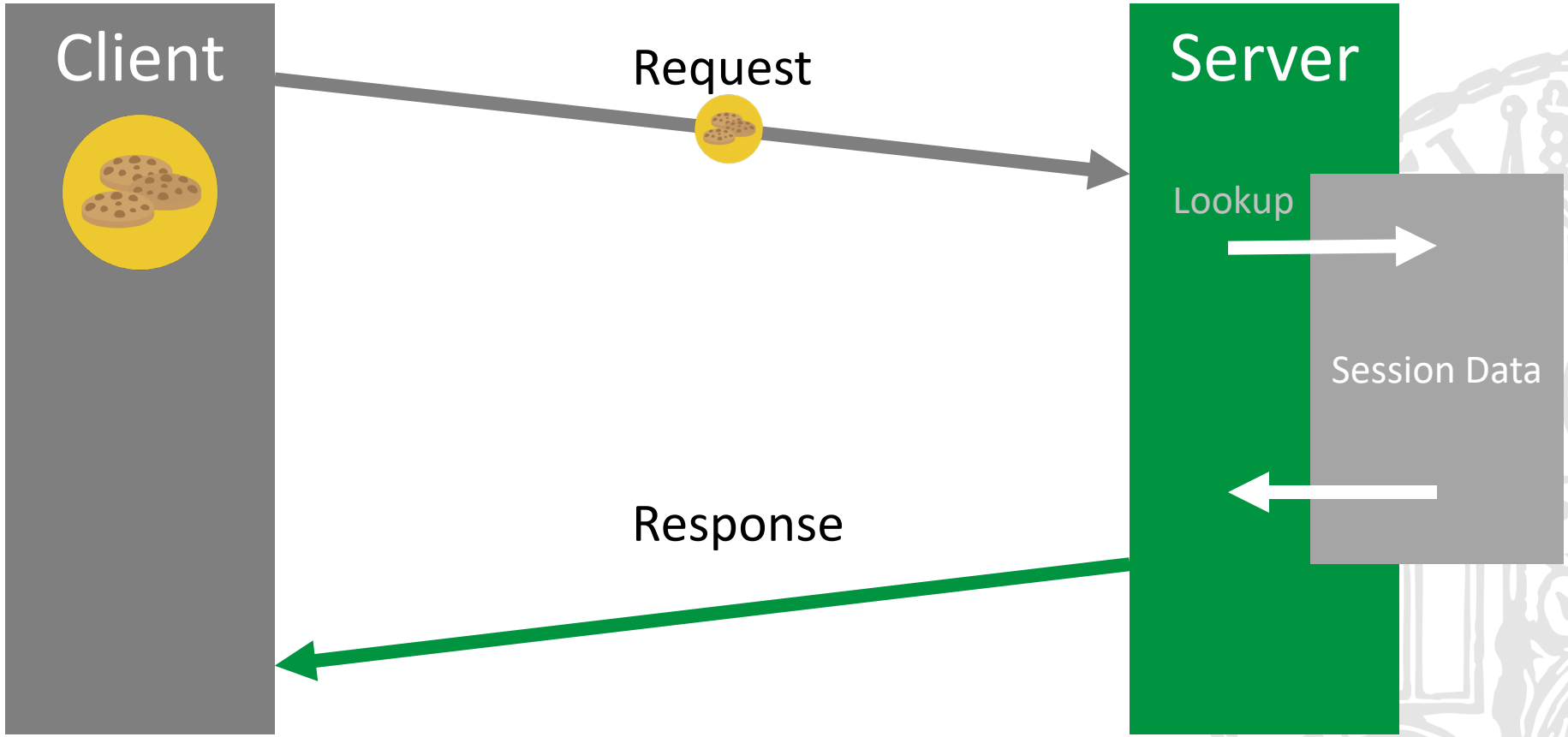

// ---- basic auth middleware ----
app.use((req,res,next) => {
  console.log(req.headers.authorization)
  if (req.headers.authorization !== 'Basic c3R1ZGVudDpvbW1pc2F3ZXNvbWU=') {
    res.set('WWW-Authenticate', 'Basic realm="401"')
    res.status(401).send()
    return
  }
  else {
    next();
  }
})

app.use(express.static(path.join(__dirname, 'public')));
app.use('/', indexRouter);
app.use('/users', usersRouter);
```

Base64 encoding of
student:ommisawesome

basic-auth-app/app.js

Traditional Authentication: with *Sessions*



Just Another Authentication Strategy

HTTP is stateless => further user properties (e.g. roles, permissions) have to be looked up on every request

JWT allows encoding of properties in a token

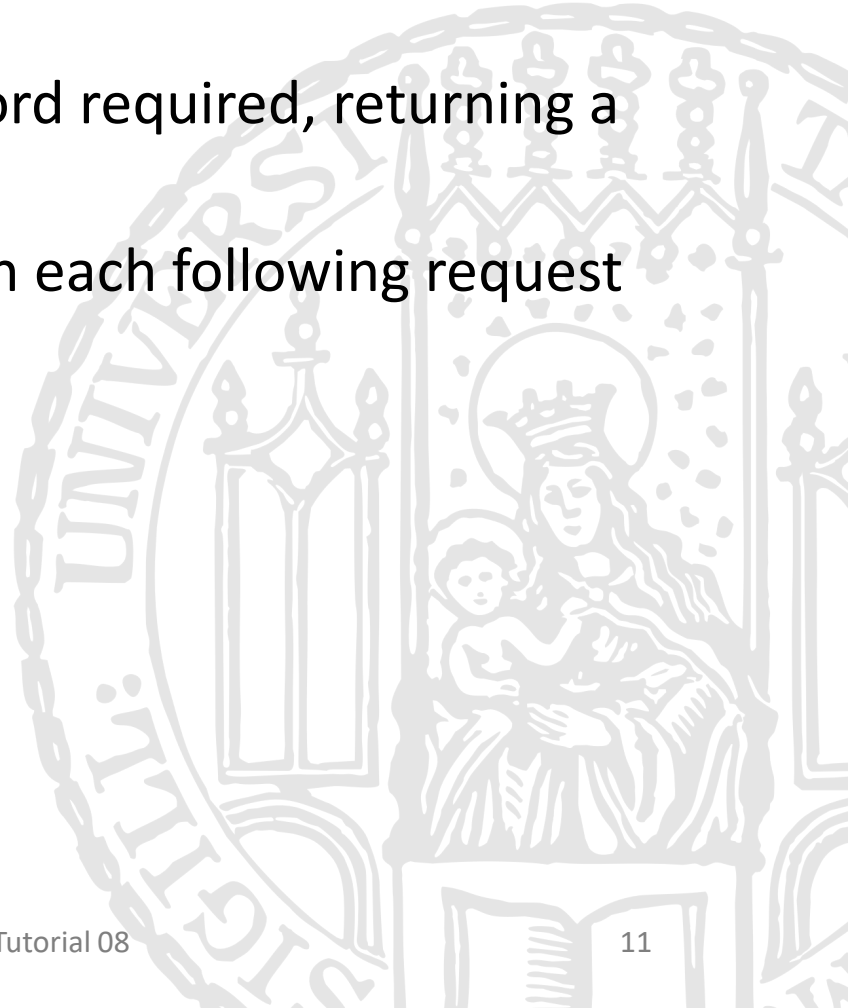
- The server encrypts user id, roles, … into a token and hands it to the client
- The client sends this token alongside each request
- The server encrypts it to authenticate the user and can ensure that the encoded properties are not manipulated

Code Along:
JWT Middleware

JWT Middleware

Two parts:

- Login route: BasicAuth login with username + password required, returning a JWT token
- Middleware: Verify the JWT token in a middleware on each following request



Step 1: Login route

- Create a new route matching /login
- Check for Basic-Auth credentials

Step 2: Create a JWT token

Import a library to do all the encryption stuff

Properties to be encrypted into the JWT token

```
const jwt = require('njwt')
const claims = {permission: 'read-data', username: 'student '}
const token = jwt.create(claims, 'something-top-secret')
token.setExpiration(new Date().getTime() + 60 * 1000)
const jwtTokenString = token.compact()
```

Encryption with a secret that should only be known by the server

Specify an expiration time (optional)

Step 3: Authorize endpoints by verifying the JWT token

- Similar to the Basic-Auth middleware in the previous example app
- Replace check for correct Authentication header with JWT token verification

Decryption, with the same secret

```
jwt.verify(jwtTokenString, 'something-top-secret', (err, verifiedJwt) => {
  if(err){
    // the given token is not valid – respond with 401!
  }else{
    // authentication successful! Continue in the middleware chain
  }
})
```

Object containing the encryted properties

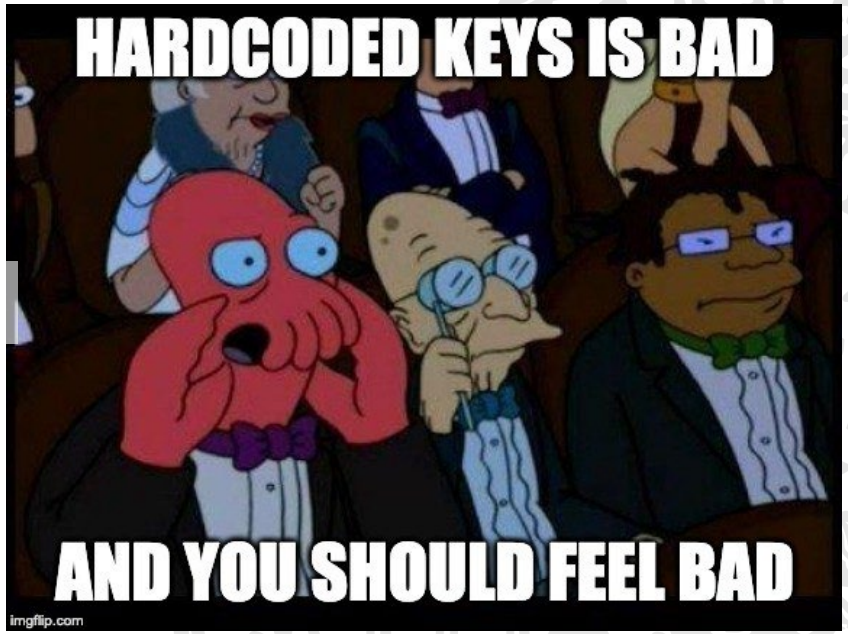
The `verifiedJwt` object, logged to the console:

```
Jwt {
  header: JwtHeader { typ: 'JWT', alg: 'HS256' },
  body:
   JwtBody {
     permission: 'read-data',
     username: 'student',
     jti: '27afc6fa-40ab-456e-8de7-c4264b9771a2',
     iat: 1574929725,
     exp: 1574929785 },
  toString: [Function] }
```

Using MongoDB with NodeJS

Why a database?

```
app.use('/login', (req,res,next) => {
if (req.headers.authorization !== 'Basic c3R1ZGVudDpvbW1pc2F3ZXNvbWU=,) {
   res.set('WWW-Authenticate', 'Basic realm="401"')
```

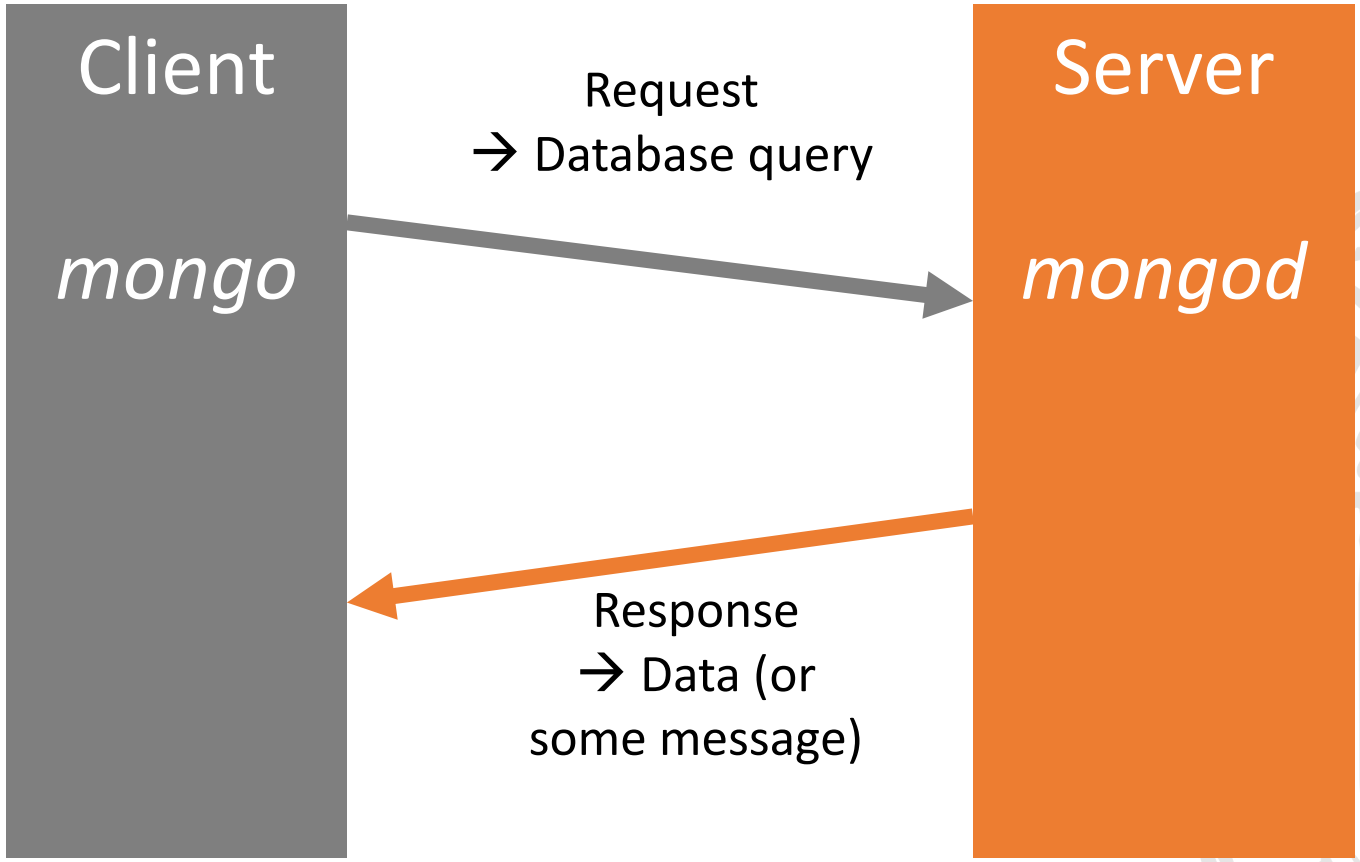


Motivation

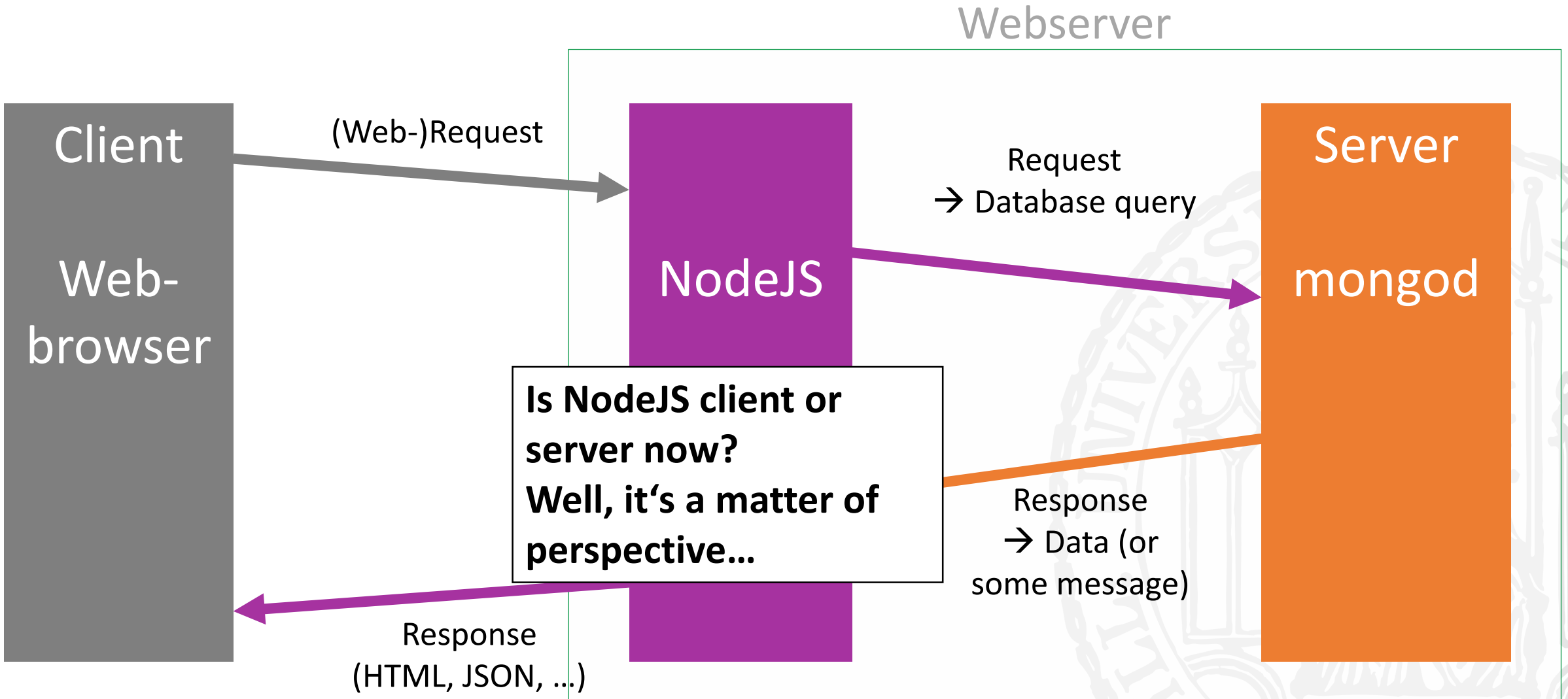
Web applications need a database to

- … store data that does not fit into working memory
- … keep data after application restart
- … structure and organize data

Again we have a client / server relationship



Is NodeJS client or server now?



NodeJS and MongoDB

- There are a couple of implementations for NoSQL/MongoDB middleware in NodeJS
- For MongoDB, the most prevalent examples are
  - monk
  - mongoose
- In the tutorial, we use monk, but mongoose works quite similar.

Using MongoDB middleware

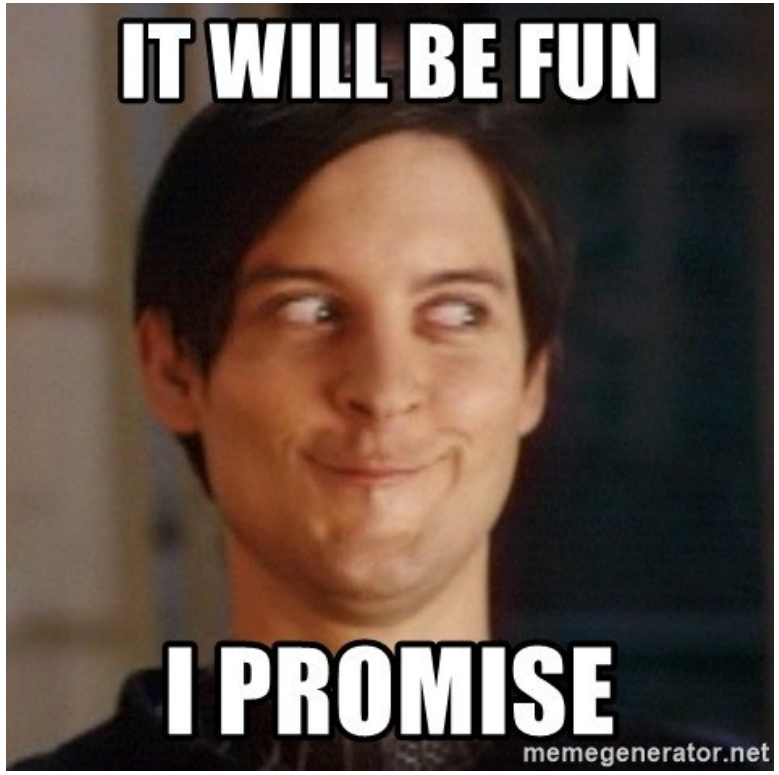
- **const *db*** = require(**'monk'**)(**'localhost/omm-1920'**);

- This has to come very early in the middleware chain:

```
app.use(function(req,res,next){  req.db = db;
        next();
});
```

On the CIP Pool: Not possible. But you could use an online-hosted database „Database-as-a-service“: https://www.mongodb.com/cloud/atlas

MongoDB using Monk



Accessing the database with monk

```
const express = require('express');
const router = express.Router();
router.get('/users',function(req,res){
1    const db = req.db;
2    const users = db.get('users');
3    users.find({},{})
       .then((docs) => res.json(docs))
       .catch((e) => res.status(500).send())
     })
module.exports = router;
```

Note: it’s not necessary to require monk here! Why?

Accessing the database with monk

```
const express = require('express');
const router = express.Router();
router.get('/users',function(req,res){
    const db = req.db;
    const users = db.get('users');
    users.find({username: 'student'},{})
      .then((docs) => res.json(docs))
      .catch((e) => res.status(500).send())
    })
module.exports = router;
```

Query "WHERE username = 'student'"

Options "SELECT *"

Note: it's not necessary to require monk here! Why?

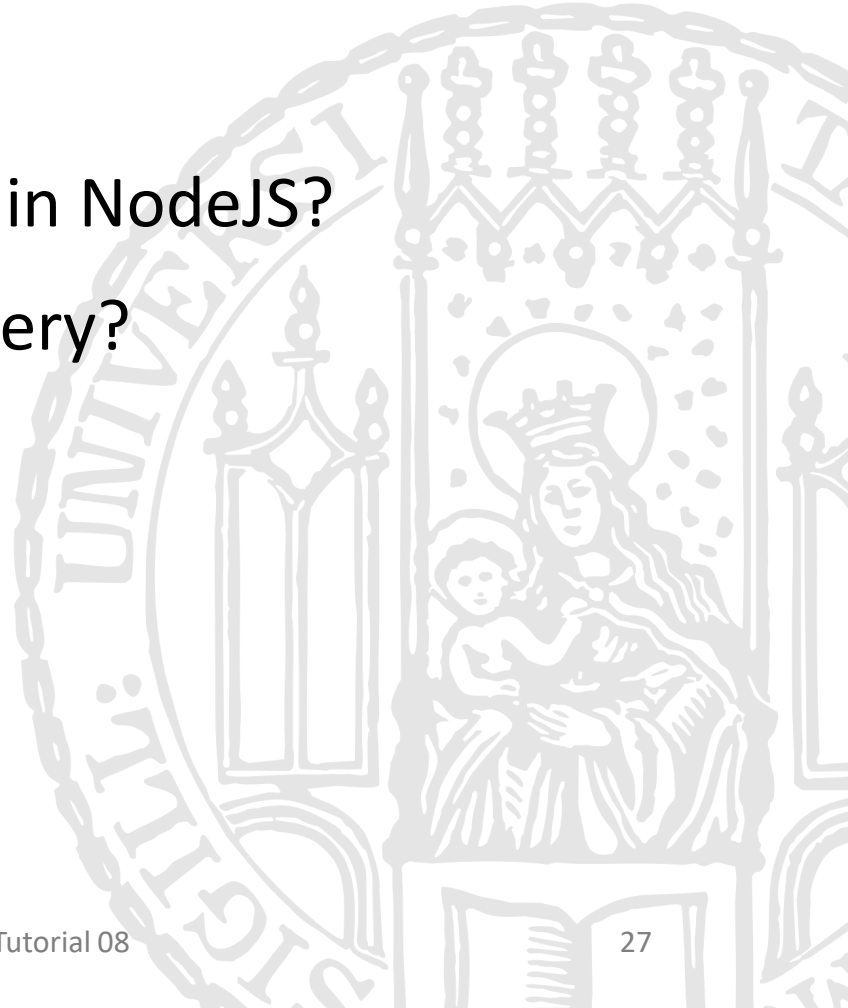
Breakout

User Database

- Setup MongoDB on your computer
- Create a MongoDB collection for users
- In the login route, check for a user in the database instead of comparing against a hardcoded Basic-Auth header

Round-up Quiz

1. Does Authentication usually come before or after Authorization?
2. What is the benefit of JWTs over Sessions?
3. What is the canonical way to add authentication in NodeJS?
4. Which parameters are common for a … Monk query?
    1. Find
    2. Update
5. Is Monk client or server?



Thanks!

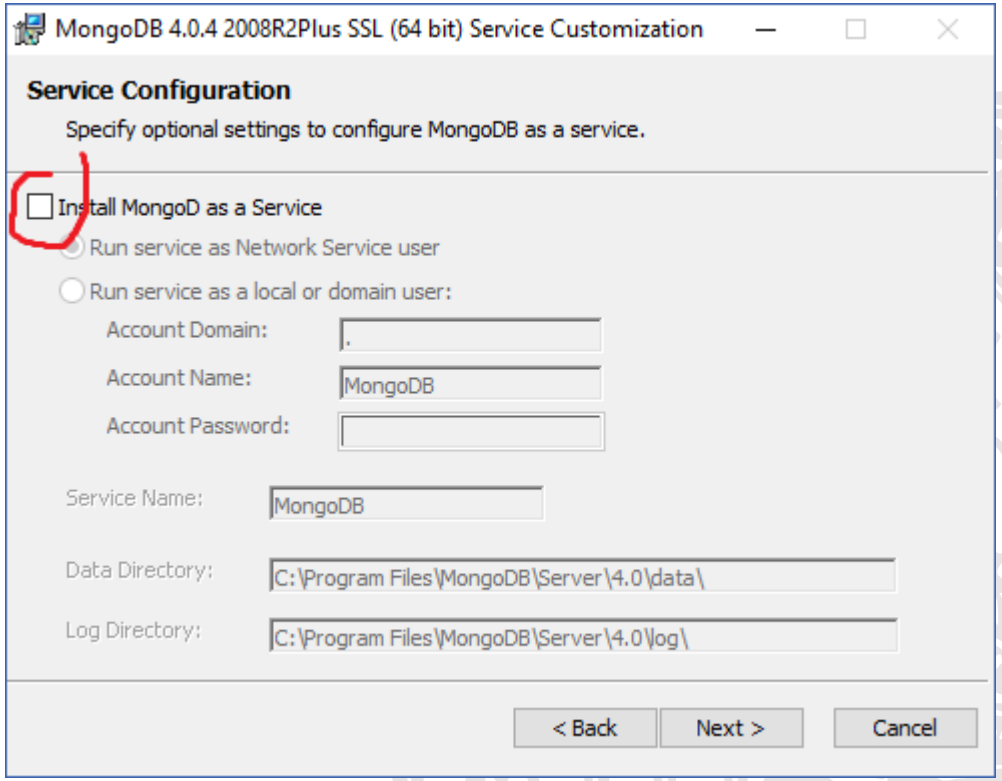
What are your questions?

Appendix

Install MongoDB (Windows)

- Download MongoDB Community Server:
  - https://www.mongodb.com/download-center/community

- Uncheck *Install MongoD as a Service*



Install MongoDB (Mac OS)

- Easiest: Use *Homebrew*
- Step 1 (optional): Update Homebrew's package database.
  ```
  > brew update
  ```
- Step 2: Install MongoDB
  ```
  > brew install mongodb
  ```

Start MongoDB (Windows)

- In a shell, start the daemon:
  ```
  C:\Program Files\MongoDB\Server\4.0\bin>mongod
  ```
- Launch mongo client:
  ```
  C:\Program Files\MongoDB\Server\4.0\bin>mongo
  ```
- Create a database:
  ```
  > use mmn1819
  ```
- Verify:
  ```
  > show dbs
  ```

Start MongoDB (Mac OS)

- In the terminal, start the daemon (using the default data directory `/data/db`):
  ```
  > mongod
  ```

- Launch mongo client:
  ```
  > mongo
  ```

  Create a database:
  ```
  > use mmn1819
  ```

- Verify by checking the process output for the following line:
  ```
  > [initandlisten] waiting for connections on port 27017
  ```

Terminology

SQL	MongoDB
database	database
table	collection
row	document
column	field
index	index
table joins	embedded documents and linking
primary key	primary key
UNIQUE column	Automatically generated _id field

Creating Collections

- Collections are created implicitly (as are databases)
- Alternative:
  db.createCollection("collectionName")

SQL	MongoDB

CREATE TABLE users (
  id INT NOT NULL
  AUTO_INCREMENT PRIMARY KEY,
  user_id VARCHAR(30),
  age INT,
  status CHAR(1)
)
	db.users.insert({
	  user_id: "abc123",
	  age: 55,
	  status: "A"
	})

Inserting Data

- Inserts are JSON Objects
- Multiple objects can be wrapped into an array and then inserted

SQL	MongoDB

`INSERT INTO users`
`  (user_id, age, status)`
`VALUES`
`  ("bcd001", 45, "A")`
	`db.users.insert({`
	`  user_id: "bcd001",`
	`  age: 45,`
	`  status: "A",`
	`})`

Multiple types in the same collection

- ```
  db.collection.insert({ foo: 'bar' });
  db.collection.insert({ lorem: 'ipsum' });
  ```

- Developers have to take care about what objects to put in a collection

- Yet, Mongo is really flexible!


  ```

Querying

SQL	MongoDB
`SELECT * FROM users`	`db.users.find()`
`SELECT id, user_id, status FROM users`	`db.users.find({ }, { user_id: 1, status: 1 })`
`SELECT * FROM users WHERE status = "A"`	`db.users.find({ }, { user_id: 1, status: 1, _id: 0 })`
`SELECT user_id, status FROM users WHERE status = "A"`	`db.users.find({ status: "A" }, { user_id: 1, status: 1, _id: 0 })`
`SELECT * FROM users WHERE status != "A"`	`db.users.find({ status: { $ne: "A" }})`

http://docs.mongodb.org/manual/reference/sql-comparison/

Update & Delete

SQL	MongoDB
`UPDATE users SET status = 'C' WHERE age > 25`	`db.users.update({ age: { $gt: 25 }}, { $set: { status: "C" } }, { multi: true })`
`UPDATE users SET age = age + 3 WHERE status = 'A'`	`db.users.update({ status: "A" }, { $inc: { age: 3 } }, { multi: true })`

SQL	MongoDB
`DELETE FROM users WHERE status = "D"`	`db.users.remove( { status: "D" } )`
`DELETE FROM users`	`db.users.remove({})`

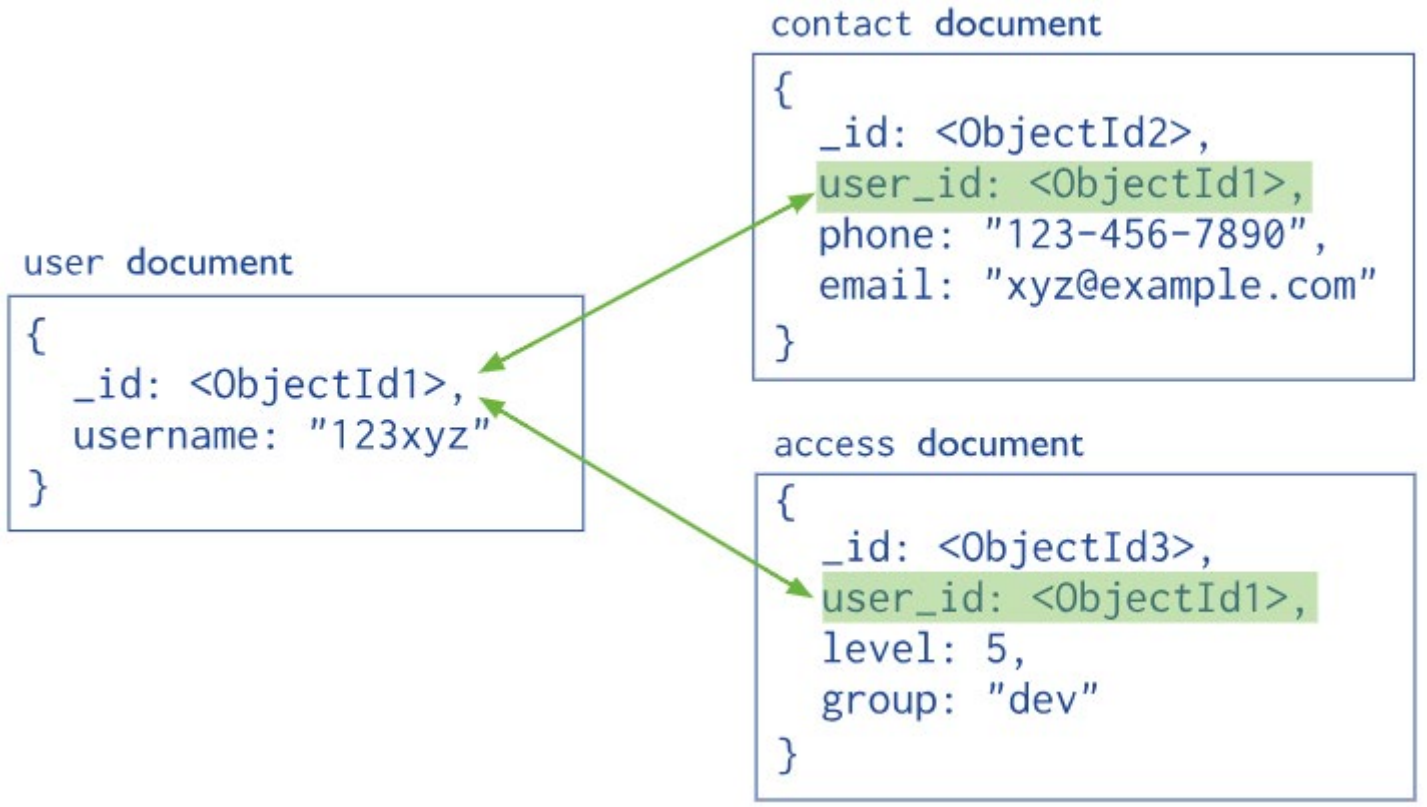
Operators

- Operators are special keys inside queries in MongoDB
- You can't write 'someKey' != 'someValue'.
- Most common operators:
  - $ne, $gt, $lt, $gte, $lte
  - $and, $or
  - $elemMatch

SQL	MongoDB
`SELECT * FROM users WHERE status = "A" OR age = 50`	`db.users.find({ $or: [ { status: "A" } , { age: 50 } ]})`

http://docs.mongodb.org/manual/reference/operator/

SQL-Like Modelling in Mongo - References

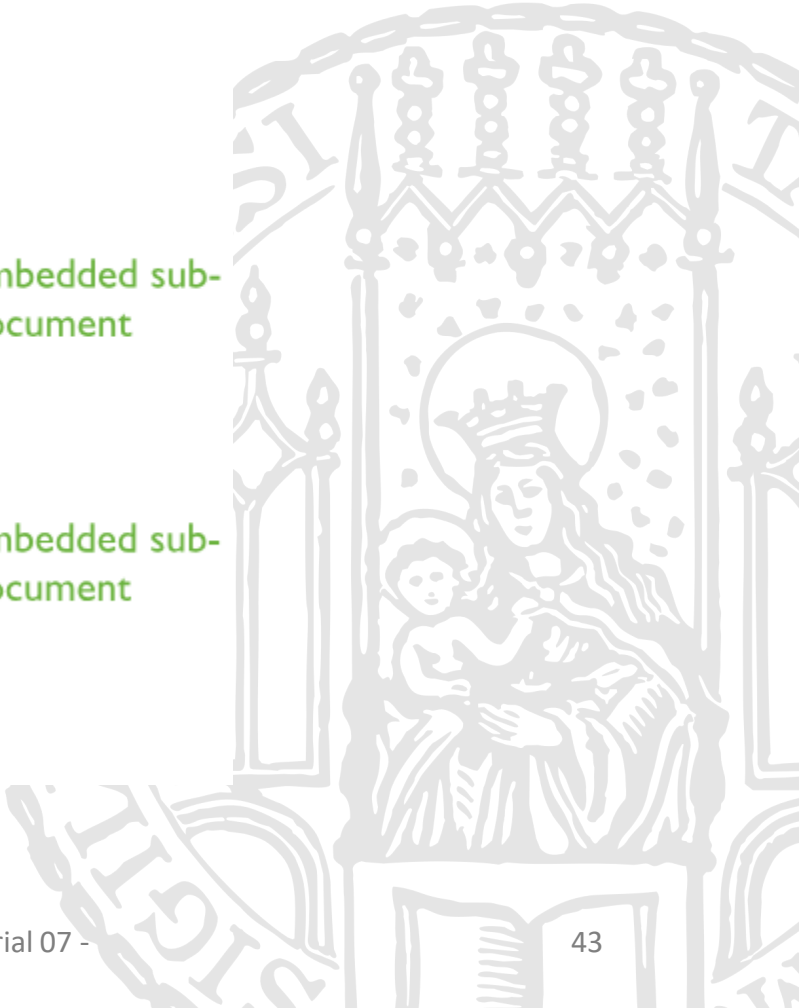


The Mongo Way – Embedded Documents

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
             phone: "123-456-7890",
             email: "xyz@example.com"
           },
  access: {
            level: 5,
            group: "dev"
          }
}
```

Embedded sub-document

Embedded sub-document



References vs. Embedded Data

- Pros Embedded Data:
  - No ORM mapping required. JSON interface fits well to web technologies
  - No tons of empty db fields in case of only sometimes set properties
  - Scalability (e.g. for embedded documents less queries are needed to select)

- Pros References:
  - Structure of a data record is not checked on insert (but you can define and enforce document validation rules)
  - „duplicate content“ in case of many relationships

Monk - Operations

- Monk wraps statements to JS Functions:
  - find()
  - findOne()
  - update()
  - updateById()
- All (!) queries are asynchronous! Callbacks!

```
router.get('/users',function(req,res){
        var db = req.db;
        var users = db.get('users');
        users.find({status : {$ne : 'A'}},function(e,docs){  res.json(docs);
        });
});
```

The Query

- Select documents by criteria
- … see last week‘s tutorial

The Options

What of and how the documents should be returned

– Select fields to return

```
users.find({}, '-name,}).then…
    // everything except the field ‘name‘
});
```

– Sorting

```
users.find({}, {sort:{name:1}}).then…
    // order by name ascending
});
```

Translating terminal queries to Javascript

Usually not that difficult – in most cases both is the same

Terminal:

db.albums.find({artists:{$elemMatch:{name:"Queen"}}})

Javascript:

db.get(**'albums'**).find({**artists**:{**$elemMatch**:{**name**:**"Queen"**}}).then…

Further Tips & Tricks

- MongoDB does not allow you have keys containing dots, eg:
  db.users.insert({'127.0.0.1': 'up'}); *// not allowed*
- Solution: Create a unique hash of the key and store it.

```
String.prototype.hashCode = function() {
        var hash = 0, i, chr, len;
        if (this.length == 0) return hash;
        for (i = 0, len = this.length; i < len; i++) {
                chr         = this.charCodeAt(i);
                hash        = ((hash << 5) - hash) + chr;  hash |= 0; // Convert to 32bit integer
        }
        return hash;
};
```