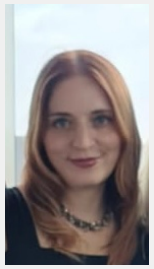
The University Of Sheffield.

COM3504/6504 The Intelligent Web

Lecture 6: MongoDB



Dr Vitaveska Lanfranchi
v.lanfranchi@sheffield.ac.uk
https://www.sheffield.ac.uk/dcs/people/academic/vitaveska-lanfranchi

Mr. Andy Stratton
a.stratton@sheffield.ac.uk
https://www.sheffield.ac.uk/dcs/people/academic/andrew-stratton

Dr Fatima Maikore
F.Maikore@sheffield.ac.uk
https://www.sheffield.ac.uk/dcs/people/research-staff/fatima-maikore

A WORLD TOP 100 UNIVERSITY

The University Of Sheffield.

Learning Objectives

- During this unit you will learn about:
  - Relational vs. NoSQL Databases
  - Types of NoSQL Databases
  - What is MongoDB
  - How do we query it
  - How do we connect to MongoDB
  - How do we model the data

The University Of Sheffield.

Large scale data storage

- Large providers such as social media providers need to scale up their infrastructure
  - both physical (number of servers) and software
- Their infrastructure is composed of hundreds of thousands of physical servers
  - Failure of nodes is expected, rather than exceptional
  - They require to build in backup and failover.
  - The number of nodes in a cluster is not constant

The University Of Sheffield.

Relational databases

- Relational databases (e.g. MySQL) are based on the idea of storing data through a standard data modeling and query language
  - SQL
- Since the rise of the web, the volume of data stored has increased
- Data is also accessed more frequently, and is processed more intensively
- Relational databases were never designed to cope with the scale and agility challenges

The University Of Sheffield.

How are relational DB used

- We need to define structure and schema of data first and then only we can process the data
- They are designed for the old mainframe world
  - They provides consistency and integrity of data
    - Useful in e.g. a banking system
    - But a significant performance overhead with large distributed data
- They require vertical scaling (i.e. increasing resources to the existing machine)

The University Of Sheffield.

How are relational DB used – cont.

- They are designed for a world where the data is to be mapped into a predefined structure
- Most applications store their data in JSON format
  - which is flexible by design
- Conversion of data has an enormous overhead in applications with high throughput
  - in one of our applications with 1 million users sending location data at high velocity
  - conversion from JSON to relational data was the single bottleneck

The University Of Sheffield.

NoSQL databases

- NoSQL databases are characterised by
  - Dynamic schemas
    - to allow the insertion of data without a predefined schema
      - But you can use one if best for your application
        - We will see how to use one
    - easy to make significant application changes in real- time
    - Relational databases require schemas to be defined before you can add data.
      - Changes require downtime

The University Of Sheffield.

NoSQL databases – cont.

- NoSQL databases are characterised by
  - Auto-sharding, replication and integrated caching
    - Sharding is creating partitions in DBs to allow for fast data processing
      - relational databases usually scale vertically – a single server has to host the entire database to ensure reliability and continuous availability of data
      - NoSQL DBs automatically scale horizontally, by adding servers instead of concentrating more capacity in a single server.
        - Distributed architecture
      - automatic replication, meaning that you get high availability and disaster recovery
      - integrated caching capabilities, keeping frequently-used data in system memory, removing the need for a separate caching layer

The University Of Sheffield.

NoSQL databases – many different types

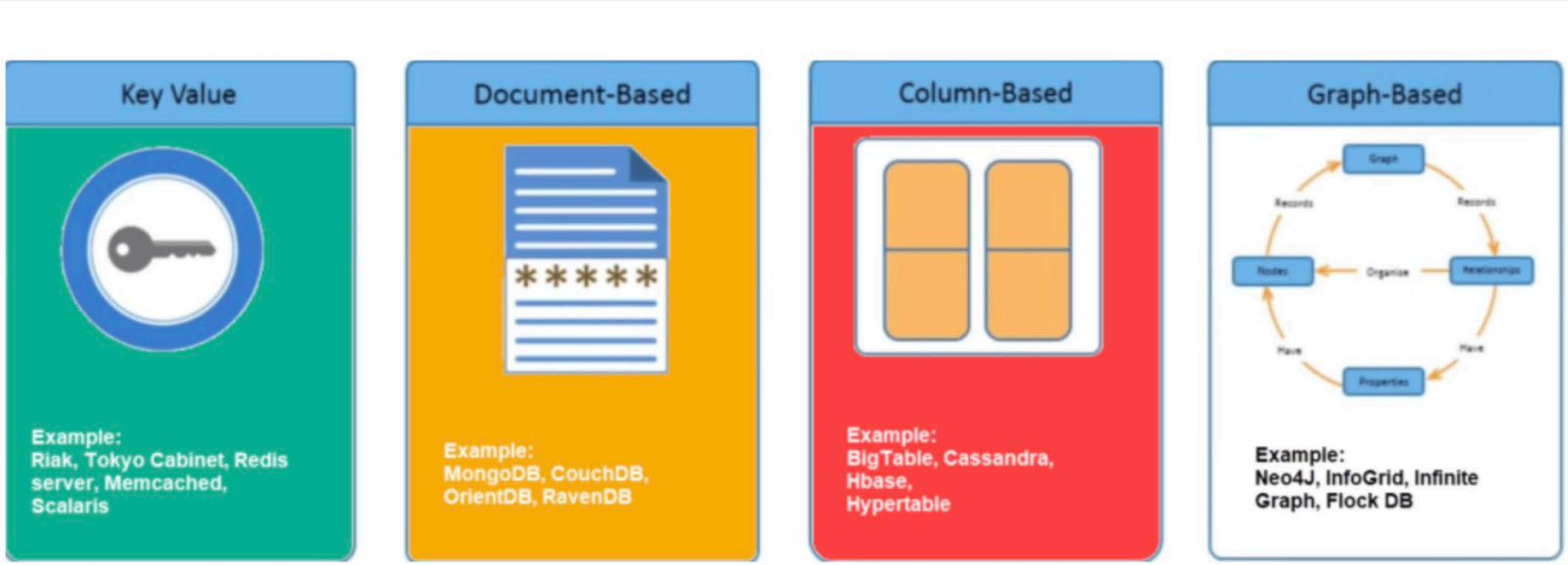


image from guru99

The University Of Sheffield.

Key-value stores

Key	Value
"Belfast"	{"University of Ulster, Belfast campus, York Street, Belfast, BT15 1ED"}
"Coleraine"	{"University of Ulster, Coleraine campus, Cromore Road, Co. Londonderry, BT52 1SA"}

- Key-value stores
  - e.g. Riak and Voldemort.

The University Of Sheffield.

Document stores

- Document databases pair each key with a complex data structure known as a document.
  - MONGODB
  - Documents can contain many different key-value pairs, or key- array pairs, or even nested documents.

The University Of Sheffield.

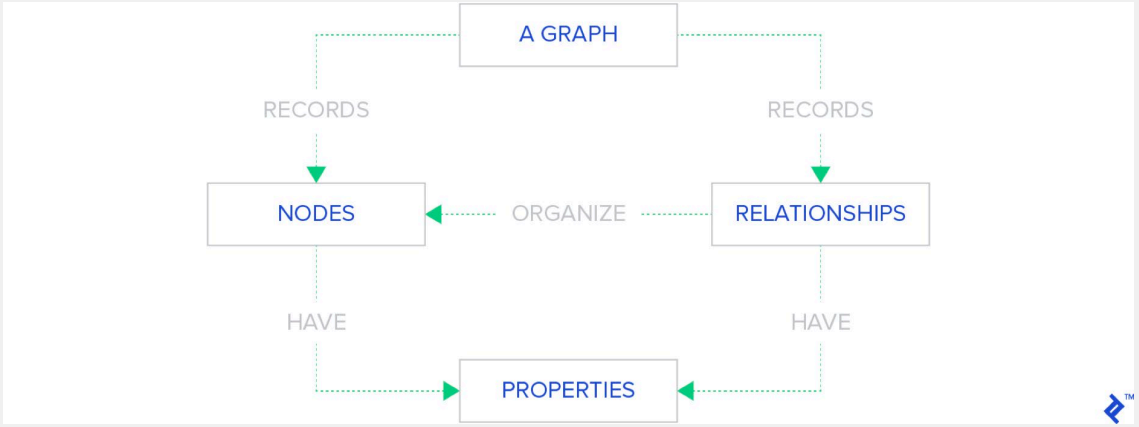
Wide Column stores

- Data is stored in columns, as opposed to rows in SQL DBs
- Fast read/write access to the data stored
- Enable effective compression as columns are typically largely uniform
- Focus on querying on one aspect of the data (e.g. price over time) rather than the entire record (price of company x between two periods)
- e.g. Cassandra and HBase
  - optimized for queries over large datasets
- BigQuery
  - created by Google for their core search engine storage system

The University Of Sheffield.

Graph stores

- A directed graph structure is used to represent the data
- Graphs are composed of edges and nodes
- Typically used in social networking applications.
- Graph databases allow developers to focus more on relations than on objects
  - e.g. Neo4J and HyperGraphDB.



The University Of Sheffield.

An overview

	Storage Type	Query Method	Interface	Programming Language	Open Source	Replication
Cassandra	Column Store	Thrift API	Thrift	Java	Yes	Async
MongoDB	Document Store	Mongo Query	TCP/IP	C++	Yes	Async
HyperTable	Column Store	HQL	Thrift	Java	Yes	Async
CouchDB	Document Store	MapReduce	REST	Erlang	Yes	Async
BigTable	Column Store	MapReduce	TCP/IP	C++	No	Async
HBase	Column Store	MapReduce	REST	Java	Yes	Async

The University Of Sheffield.

Issues with NoSQL DB

- No standardisation for data into relations
  - Freedom but also a damning feature if overused
- Limited query capabilities
- No automatic consistency checking
  - e.g. when multiple transactions are performed simultaneously
- Eventual consistency is not appropriate for every application

The University Of Sheffield.

MONGODB

The University Of Sheffield.

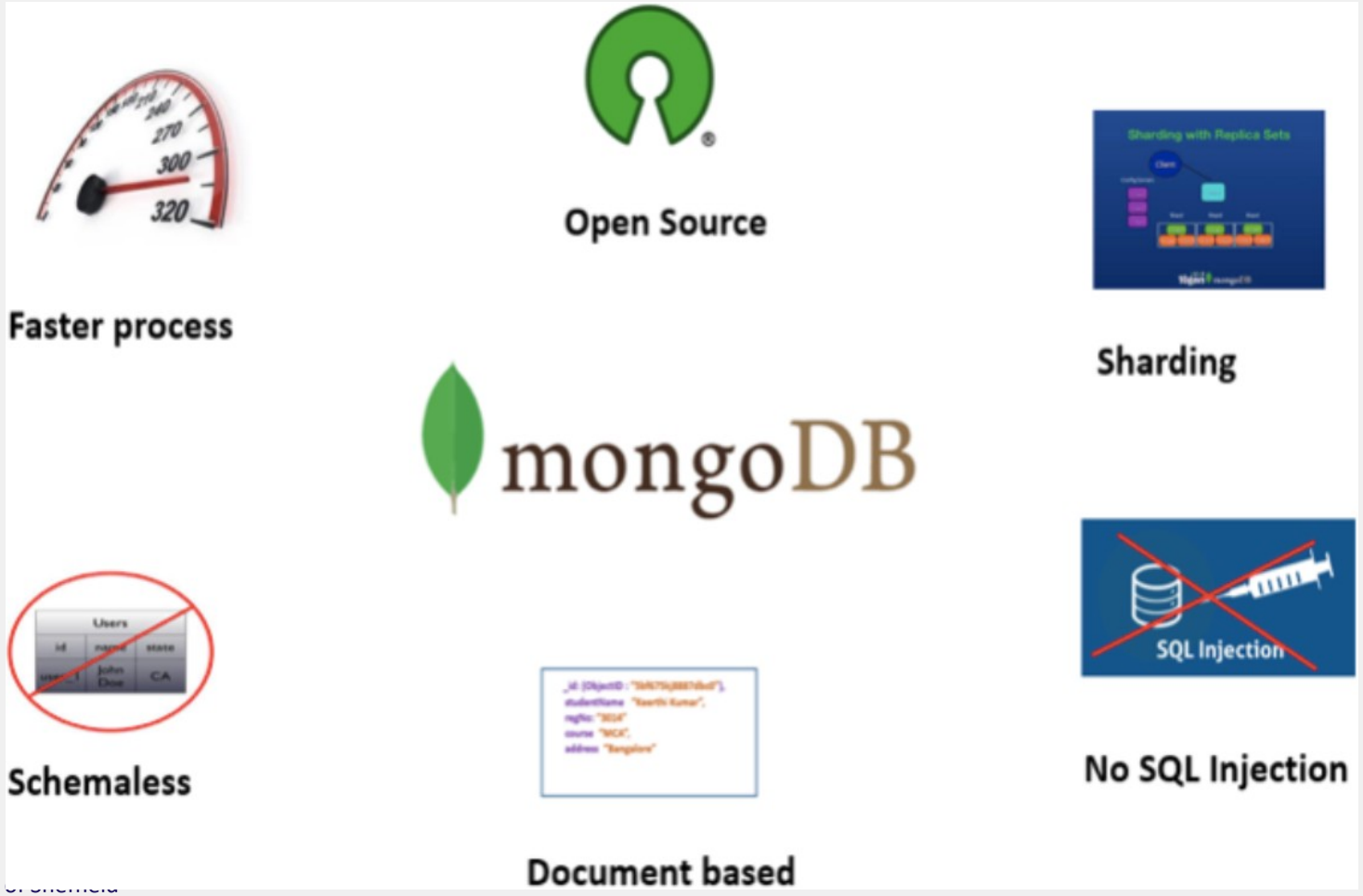
https://docs.mongodb.com/getting-started/shell/introduction/

What is MongoDB

- MongoDB is an open-source document database that provides
  - high performance
  - high availability
  - automatic scaling
- It is a NoSQL database
  - Document database
  - Suitable for high volume data storage
- MongoDB is written in C++

The University Of Sheffield.

Advantages



The University Of Sheffield.

Structure

- Each database contains collections
  - which in turn contains documents.
- Each document can be different with a varying number of fields.
  - The size and content of each document can be different from each other.
- The document structure is in line with how developers construct their classes and objects
- The data model allows you to represent hierarchical relationships, to store arrays etc.

The University Of Sheffield.

Documents

- A record in MongoDB is a document, which is a data structure composed of field and value pairs.
- MongoDB documents are similar to JSON objects.
- The values of fields may include other documents, arrays, and arrays of documents

SQL Records -> Mongos' Documents

The University Of Sheffield.

Collections

- MongoDB stores documents in collections. Collections are analogous to tables in relational databases.
  - Unlike a table, however, a collection does not require its documents to have the same schema
- In MongoDB, documents stored in a collection must have a **unique _id** field that acts as
- a **primary key**

SQL Relations -> Mongos' Collections

The University Of Sheffield.

Example

- Example of a MongoDB document describing a restaurant


```
{
  "_id" : ObjectId("54c955492b7c8eb21818bd09"),
  "address" : {
    "street" : "2 Avenue",
    "zipcode" : "10075",
    "building" : "1480",
    "coord" : [ -73.9557413, 40.7720266 ]
  },
  "borough" : "Manhattan",
  "cuisine" : "Italian",
  "grades" : [
    {
      "date" : ISODate("2014-10-01T00:00:00Z"),
      "grade" : "A",
      "score" : 11
    },
    {
      "date" : ISODate("2014-01-16T00:00:00Z"),
      "grade" : "B",
      "score" : 17
    }
  ],
  "name" : "Vella",
  "restaurant_id" : "41704620"
}
```

The University Of Sheffield.

BSON

- Mongo's documents are internally represented as binary JSON

The University Of Sheffield.

BASIC MONGODB COMMANDS

The University Of Sheffield.

Create a database

- To create a database in MongoDB,
  - start by creating a MongoClient object,
  - then specify a connection URL with
    - the correct ip address and
    - the name of the database you want to create.
- MongoDB will create the database if it does not exist, and make a connection to it.

The University Of Sheffield.

Create a database – cont.

- Remember In MongoDB, a database is not created until it gets content!
  - waits until you have created a collection with at least one document before it actually creates it

The University Of Sheffield.

Example

Example

Create a database called "mydb":

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  console.log("Database created!");
  db.close();
});
```

The University Of Sheffield.

Create a collection

- To create a collection in MongoDB: `createCollection()` method

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
      dbo.createCollection("cats", function(err, res) { if (err) throw err;
          console.log("Collection created!"); db.close();
  });
});
```

The University Of Sheffield.

Inserting a document

- Tocreate a document
  - insertOne()method.
- It takes as parameters
  - an object containing the name(s) and value(s) of each field in the document
  - A callback function where you can work with any errors, or the result of the insertion

The University Of Sheffield.

Inserting a document - example

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myobj = { name: "Neve", breed: "Angora" };
dbo.collection("cats").insertOne(myobj, function(err, res) {
        if (err) throw err;
            console.log("1 document inserted"); db.close();
  });
});
```

The University Of Sheffield.

Finding a document

- To find a document
  - findOne() method.
- It takes as parameters
  - an object containing the query
    - If empty finds all
  - A callback function where you can work with any errors, or the result of the function

The University Of Sheffield.

Finding a document - example

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
dbo.collection("cats").findOne({}, function(err, result) {
        if (err) throw err; console.log(result.name); console.log(result.breed);
        db.close();
  });
});
```

The University Of Sheffield.

Querying a document

- To query a document
  - find() method.
- It takes as parameters
  - an object containing the query
    - used to limit the search.
    - Can use regular expressions
  - A callback function where you can work with any errors, or the result of the function

The University Of Sheffield.

Querying a document - example

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var query = { breed: "Angora" };
dbo.collection("cats").find(query).toArray(function(err, result) {
        if (err) throw err; console.log(result); db.close();
  });
});
```

The University Of Sheffield.

Other commands

- sort()
  - sort the result in ascending or descending order
  - sort object takes as value
    - 1 - ascending
    - -1 descending

```
var mysort = { name: 1 };
dbo.collection("customers").find().sort(mysort).toArray(function(e  rr, result)
```

- deleteOne()
  - defining which document to delete.

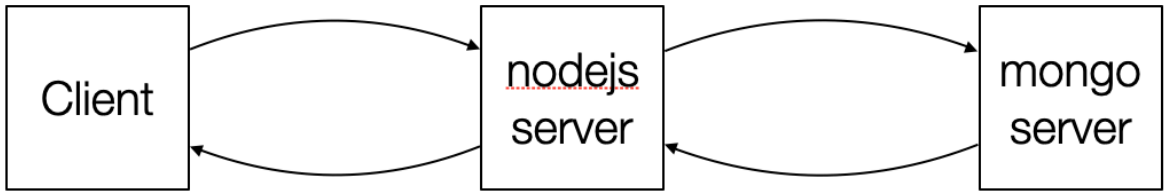
The University Of Sheffield.

Other commands – cont.

- drop()
  - Deletes a collection
- updateOne()
  - Updates a document.

```
MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    var myquery = { name: "Neve" };
    var newvalues = { $set: {name: "Neve",
    breed: "Turkish Van" } };
    dbo.collection("cats").updateOne(myquery,
    newvalues, function(err, res) {
        if (err) throw err;
        console.log("1
        document
        updated");
        db.close();
    });
});
```

The University Of Sheffield.



A separate process

- Mongo, as any db system, must run in a separate process from your main nodejs server
  - remember: nodeJS is fast and scalable but no long-running process is to be run on its single thread
- Create a separate process for that and connect using the appropriate library
  - Mongoose in our case

The University Of Sheffield.

How to connect MongoDB to Node.JS

- First of all you must download MongoDB
  - https://www.mongodb.com/download-center/ community
- To access Mongo from within a Node application, a driver is required.
  - There are number of Mongo drivers available
  - Mongoose and MongoDB are among the most popular.
- You can install it from command line or IntelliJ
  - We will do this in the live lesson

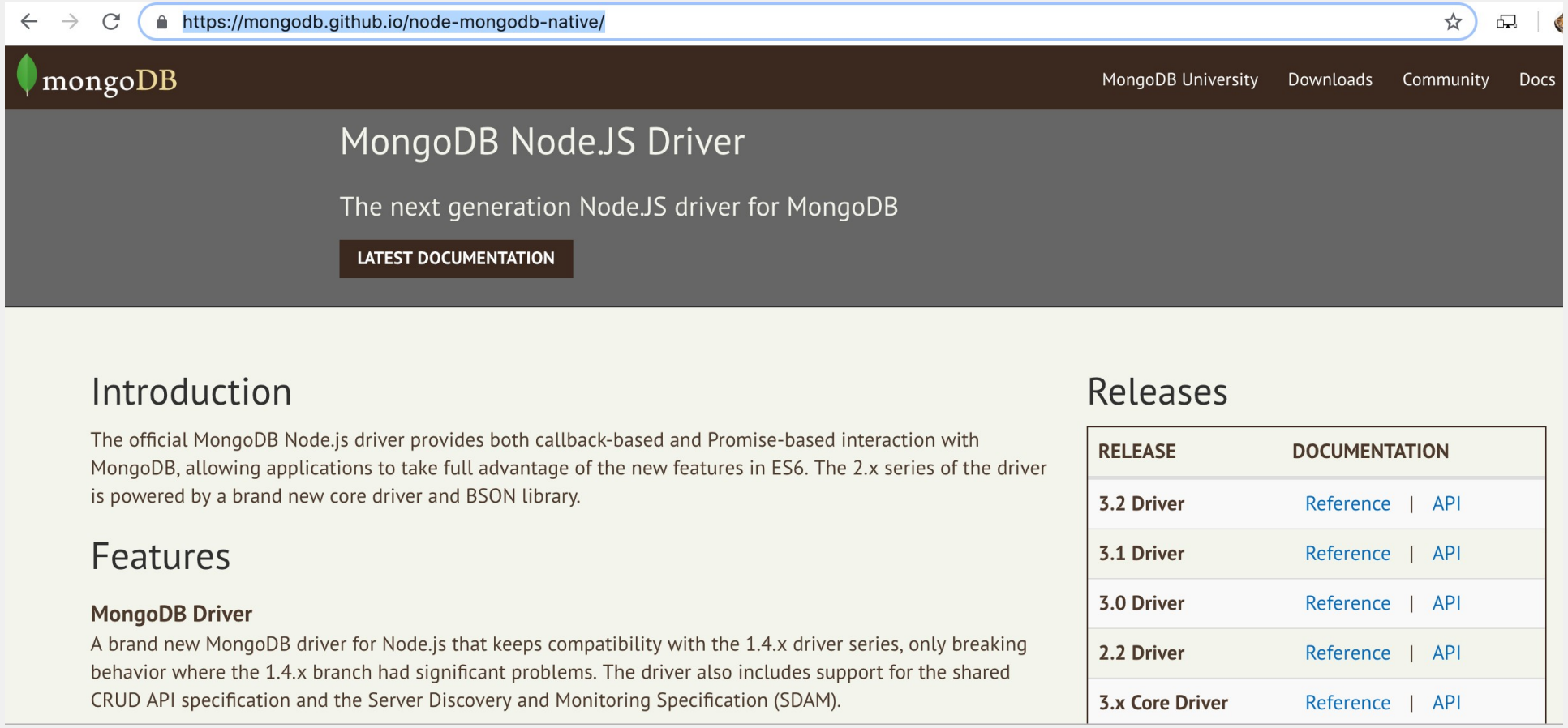
The University Of Sheffield.

Mongoose and MongoDB driver

- MongoDB driver is the **native driver** for interacting with a mongodb instance
- Mongoose is an **Object modeling tool** for MongoDB
  - built upon the MongoDB driver to provide programmers with a way to model their data.
  - Abstraction layer over MongoDB
  - We will use both!

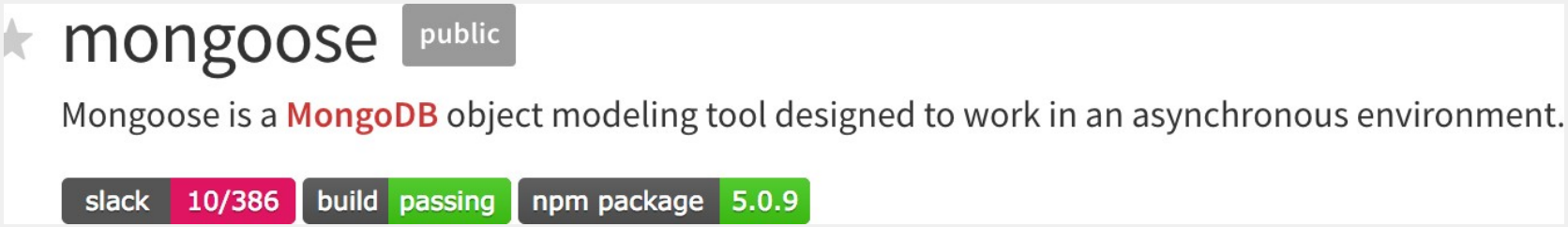
The University Of Sheffield.

MongoDB driver



The University Of Sheffield.

Mongoose




```
npm install mongoose
```

```
// Using Node.js `require()`
const mongoose = require('mongoose');
```

The University Of Sheffield.

USING MONGOOSE AND DATA MODELS

Why?

- Using an Object Data Model ("ODM") / Object Relational Model (“ORM")
  - Allows to represent our data as JavaScript objects
  - which are then mapped to the underlying database.
- Advantages
  - It is consistent with the language you are using
  - You can continue to think in terms of JavaScript objects rather than database ones
- Mongoose acts as a front end to MongoDB

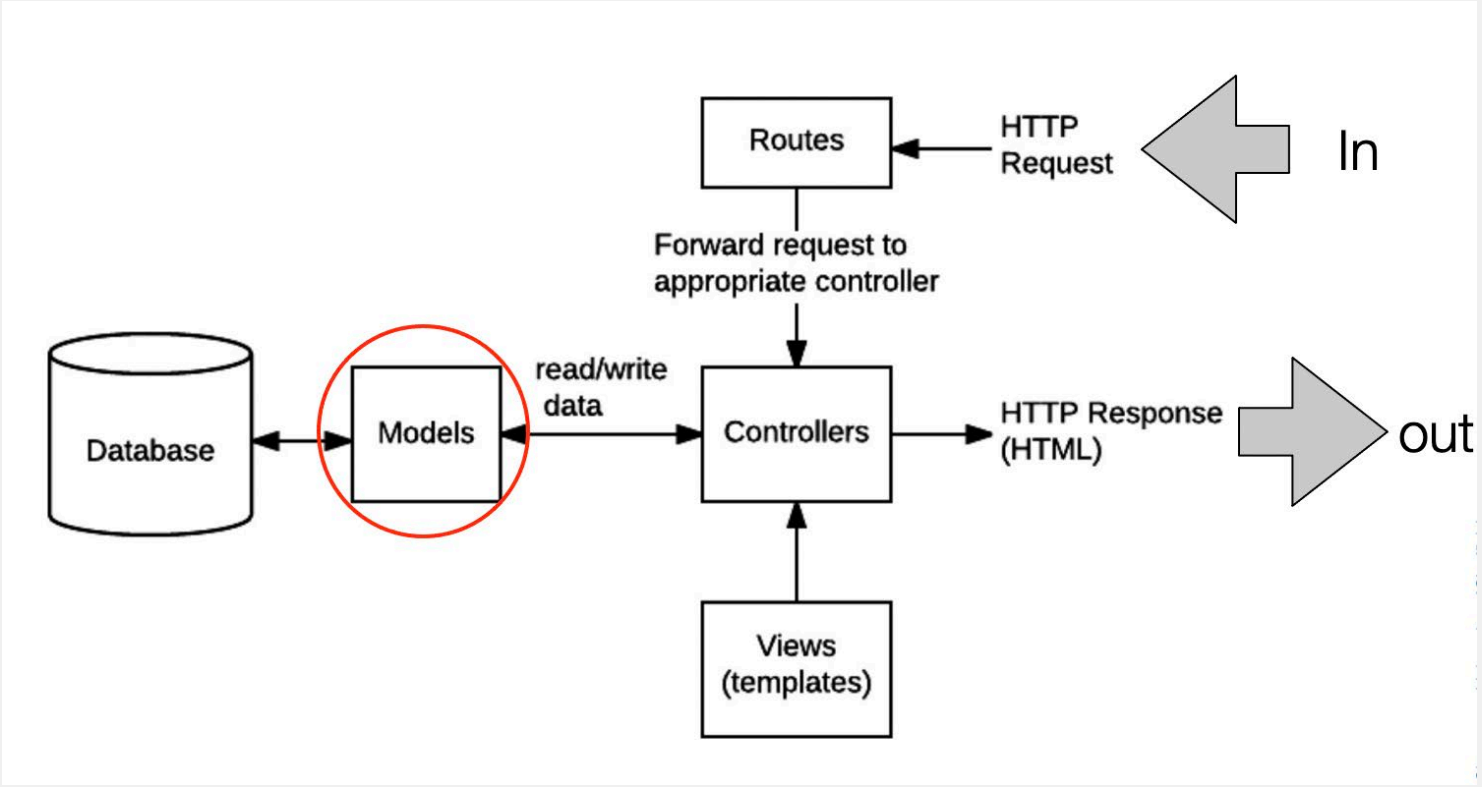
The University Of Sheffield.

Why? – cont.

- Creating a data model allows you to reason on your data
- It helps to
  - validate data integrity of documents
  - Ensure legibility/maintenance
- When designing your models you should have separate models for every "object" (a group of related information)
  - e.g. a model for cats, dogs, instance of cats, instances of dogs
- You can think in terms of UML diagrams if it helps

The University Of Sheffield.

Do you remember this?



The University Of Sheffield.

How to create models

- Models are *defined* using the `Schema` interface:
  - the fields stored in each document
    - their validation requirements and default values.
  - static and instance helper methods
  - virtual properties
    - Are not stored in the database but can be used

The University Of Sheffield.

How to create models – cont.

- Each model maps to a *collection* of *documents* in the MongoDB database.
- Schemas are then "compiled" into models using the `mongoose.model()` method.
  - you can use it to find, create, update, and delete

The University Of Sheffield.

Like Java

- A schema is the equivalent of a java interface
  - it must be implemented in a model before being used
- Unlike java interfaces however, it defines types and restrictions
- It is not an instance that can be used

The University Of Sheffield.

Example

```
// grab the things we need
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

// create a schema
var catSchema = new Schema({
    name: String,
    breed: String,
    location: String,
    age: Number,
    created_at: Date,
    updated_at: Date
});

catSchema.methods.catOfTheMonth = function() {
    // add some stuff to the users name
    this.name = this.name + '- Cat of the Month';

    return this.name;
};


// the schema is useless so far
// we need to create a model using it
var Cat = mongoose.model('Cat', catSchema);

// make this available to our users in our Node applications
module.exports = Cat;
```

Uses mongoose

Creates a new Schema

Creates a schema of a cat with some variables

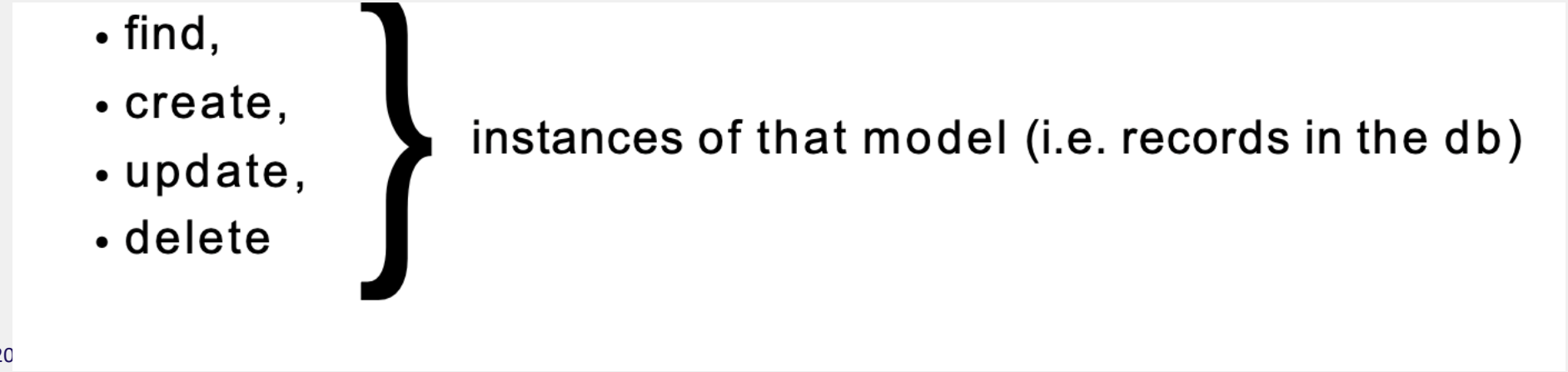
Creates a method that does something

Creates a model of my cat

The University Of Sheffield.

Compiling a schema into a model

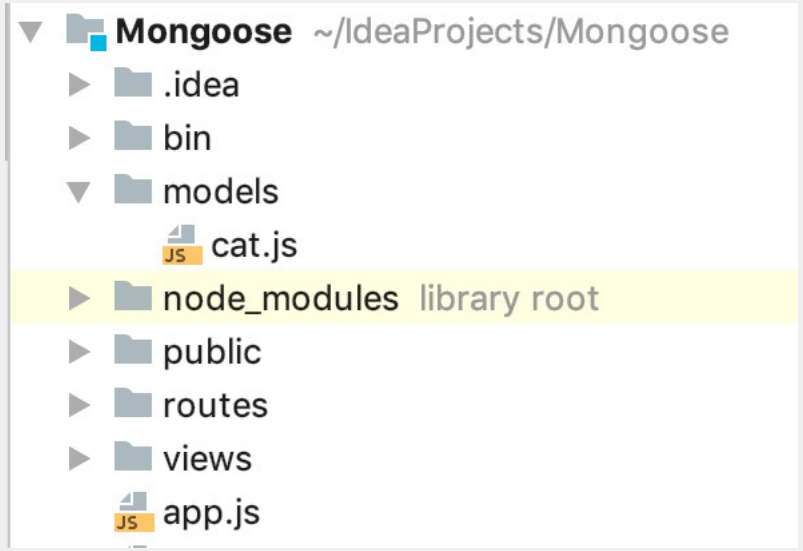
- The Schema allows you to define the fields stored in each document along with their validation requirements and default values.
- Schemas are then "compiled" into models using the **mongoose.model()** method.
- Once you have a model you can use it to



The University Of Sheffield.

How to save models

- Your models will be saved as a .js file
- Keep them in a specific folder to keep your code organised
  - E.g. models



The University Of Sheffield.

Validation

- Mongoose provides built-in and custom validators, and synchronous and asynchronous validators. It allows you to specify both the acceptable range or values and the error message for validation failure in all cases.
- Examples of built-in validators
  - Numbers have min and max validators.
  - Strings have:
    - enum: specifies the set of allowed values for the field.
    - match: specifies a regular expression that the string must match.
    - maxlength and minlength for the string.

The University Of Sheffield.

Validation - Example

```
var Character = new Schema(
    {
        first_name: {type: String, required:true,max:   100},
        family_name:{type: String, required:true,max:   100},
        dob: {type: Number, required: true, max: 2020},
        whatever: {type: String} //any other field
    }
);
```

max string length

max value

A WORLD TOP 100 UNIVERSITY

The University Of Sheffield.

Instances

new <ModelName>({<fields>}) creates an instance of a model

```
var Cat = require('./models/cat')

var cat = new Cat({ name: 'Neve', breed: 'Angora',
location: 'Sheffield', age: '12'
});
```

then you can save it into the database (with callback)

```
cat.save(function (err, results) {
    console.log(results._id);
});
```

The University Of Sheffield.

Virtual properties

- Document properties that you can get and set but that do not get persisted to MongoDB
- useful for formatting or combining fields,
  - It is easier and cleaner and uses less disk space
  - it allows for dynamic properties

The University Of Sheffield.

Virtual properties - example

- a full name virtual property starting from concrete fields called first name and last name
- (Dynamic): the age of a person computed from the current year and date of birth

```
// Virtual for age of a person
Character.virtual('age')
.get(function () {
   const currentDate = new Date().getTime();
   return currentDate-this.date_of_birth;
});
```

The University Of Sheffield.

Searching

- You can search over your objects with the same commands we used before, only calling them on your object
  - i.e. to find a cat

```
var Cat = require('./models/cat')
Cat.find({name: 'Fluffy'}, function(err, results) {
 if (err) throw err;

console.log(results);

});
```

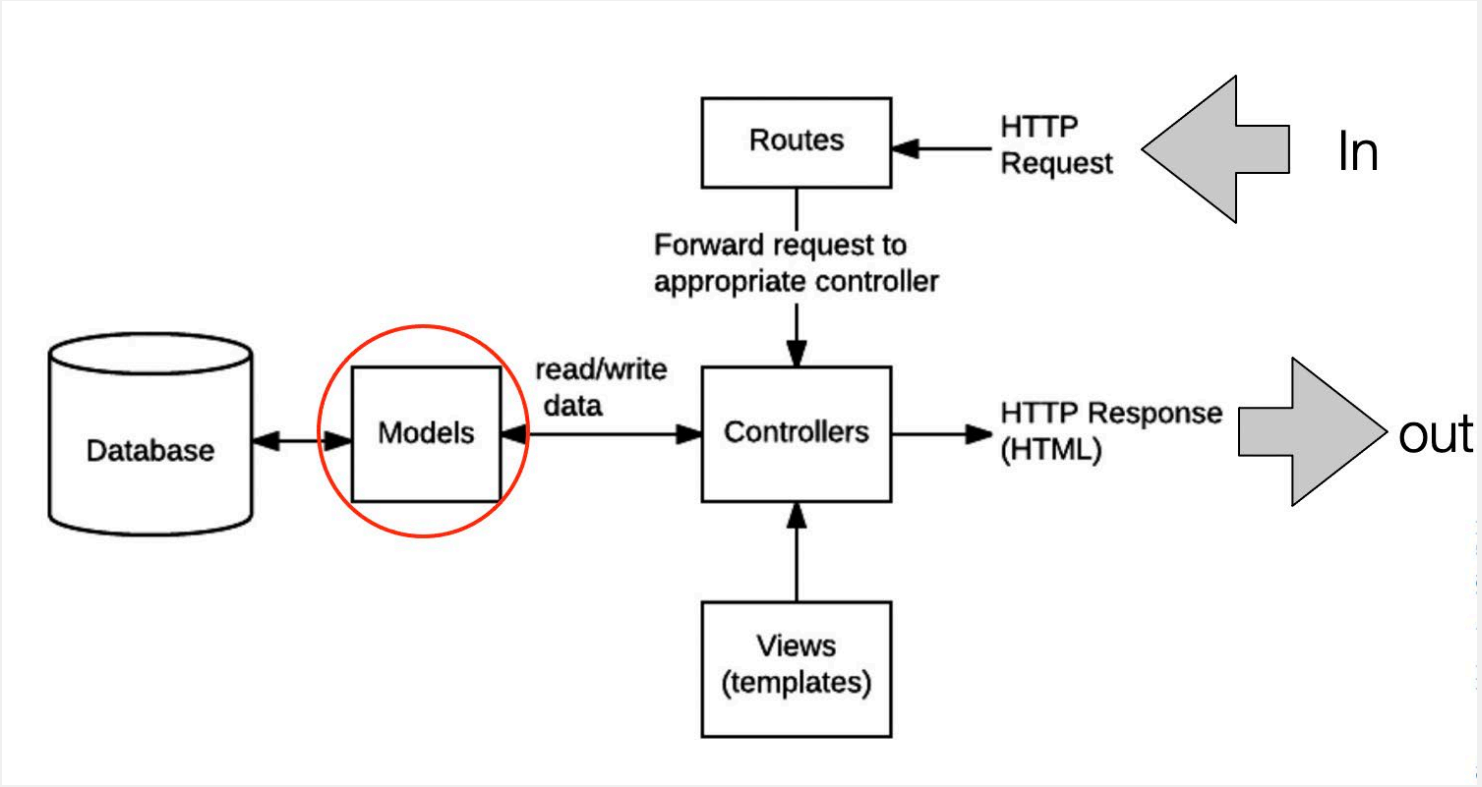
The University Of Sheffield.

HOW SHOULD A PROJECT LOOK LIKE AT THE END?

A WORLD TOP 100 UNIVERSITY

The University Of Sheffield.

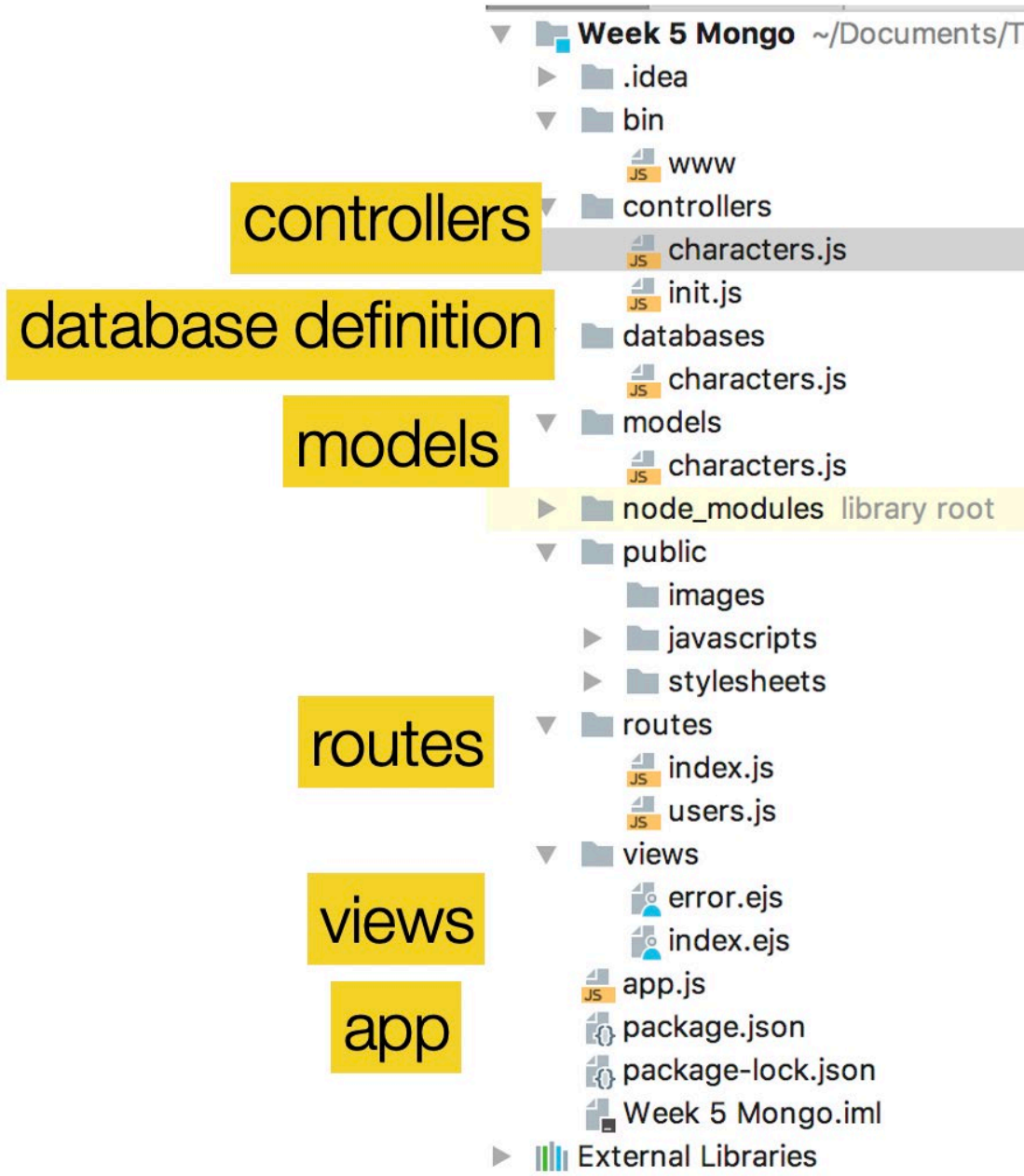
The diagram, again!



The University Of Sheffield.

Project Structure

- Your app will be organised in this way
- There is a database called ‘cats’
  - which has a model called ‘cats’ representing name, breed, location and age of each cat



The University Of Sheffield.

Organising models

- It is recommended that you define just one model per file
  - and then export the model

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

// create a schema
var catSchema = new Schema({
    name: String,  breed: String,
    location: String,  age:
    Number,  created_at: Date,
    updated_at: Date
});


// the schema is useless so far
// we need to create a model using it
var Cat = mongoose.model('Cat', catSchema);

// make this available to our users in our Node applications
module.exports = Cat;
```

The University Of Sheffield.

Controllers

- Controllers are the route-handler callback function
- Controller functions get the requested data from the models, create an HTML page displaying the data, and return it to the user to view

The University Of Sheffield.

How controllers work

- They Import the models and use the model as an object, e.g.:
  - then they define a list of exported functions to be used in the routes

```
var Author = require('../models/author');

// Display list of all Authors.
exports.author_list = function(req, res) {
    res.send('NOT IMPLEMENTED: Author list');
};

// Display detail page for a specific Author.
exports.author_detail = function(req, res) {
```

The University Of Sheffield.

Linking a controller to a route

- Once is exported a controller can be used in the routes

```
// POST request to update Author.
router.post('/author/:id/update', author_controller.author_update_post);

// GET request for one Author.
router.get('/author/:id', author_controller.author_detail);

// GET request for list of all Authors.
router.get('/authors', author_controller.author_list);

/// GENRE ROUTES ///

// GET request for creating a Genre. NOTE This must come before route that d
router.get('/genre/create', genre_controller.genre_create_get);
```

The University Of Sheffield.

Questions

A WORLD TOP 100 UNIVERSITY