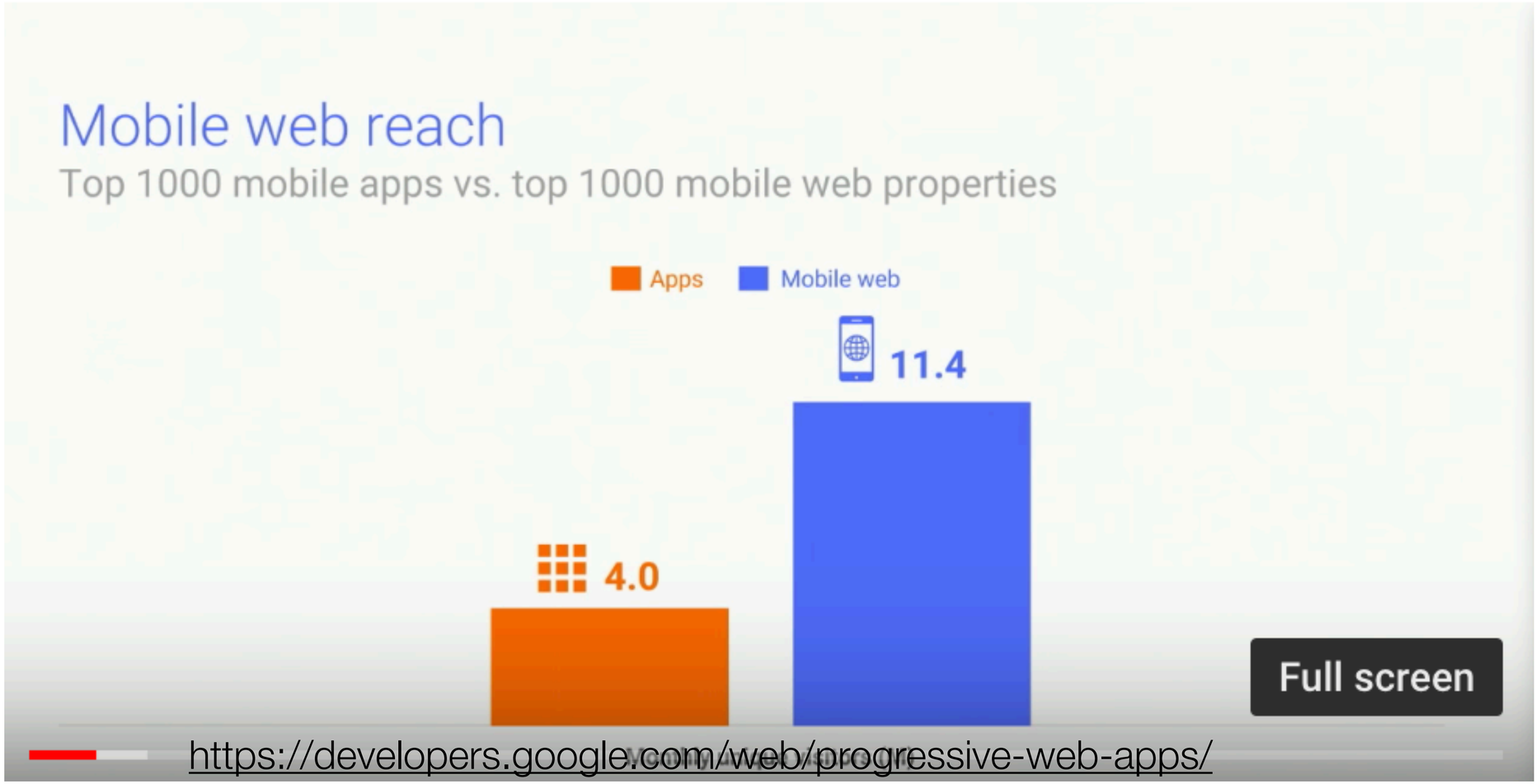
Offline Experience for the Mobile Web: Progressive Web Apps Service Workers

Prof Fabio Ciravegna
Department of Computer Science, The University of Sheffield
f.ciravegna@shef.ac.uk

Apps Vs Web Users

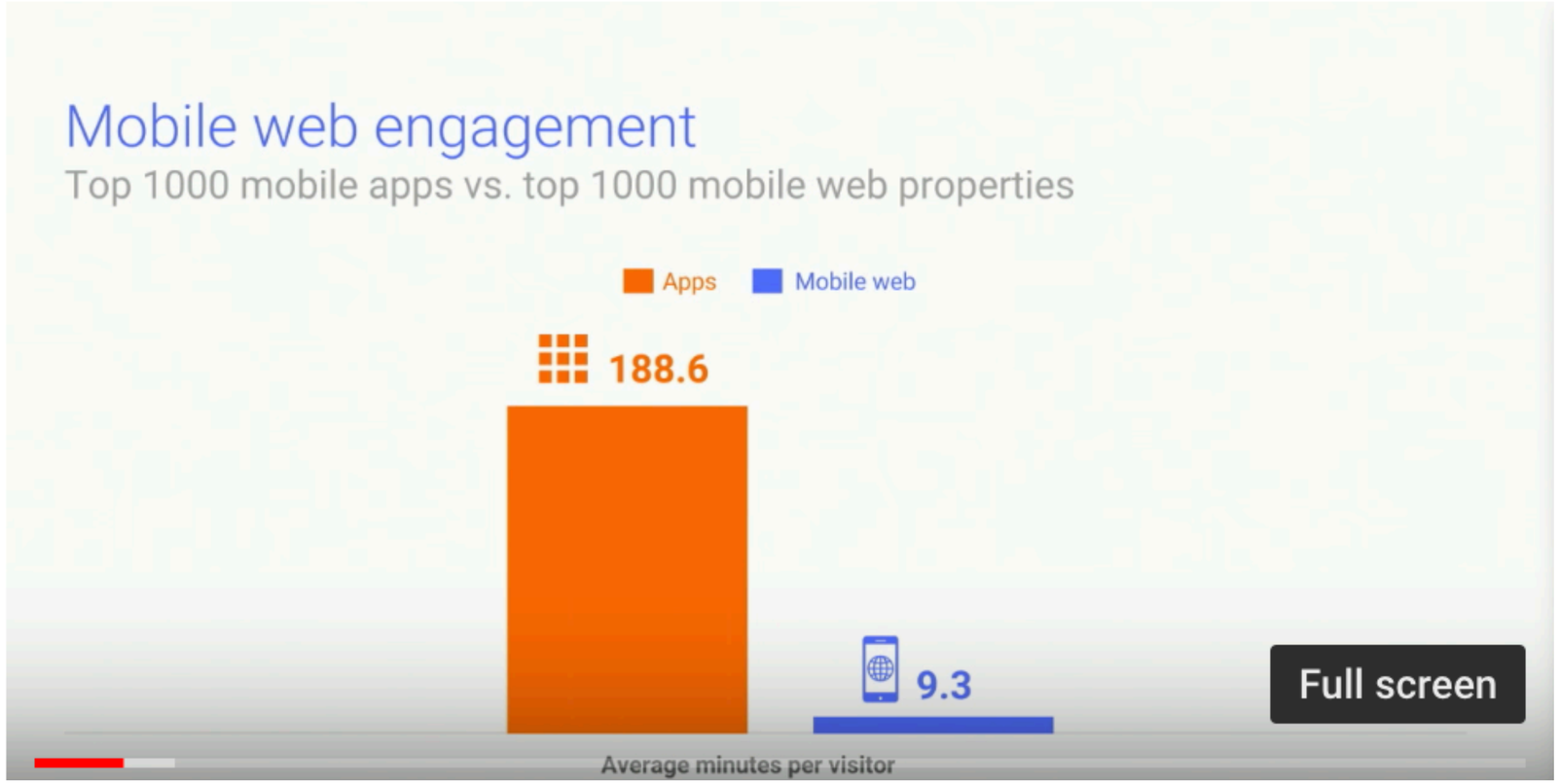
- Users are far more likely to access Web services via the browser than via mobile apps



https://developers.google.com/web/progressive-web-apps/

The Mobile Web

- However, those using apps are more easily engaged and spend more time on your content



The University Of Sheffield.

Issues with the Mobile Web

- Despite bringing traffic to your site, it is unlikely to bring returning customers
  - difficulty in typing URLs, general cumbersome browser interface
- Despite the advances in responsive, mobile-first websites web content may still be unnatural on mobile browsers
- Break in connectivity is the norm rather than the exception in mobile web experience
  - especially if out of town or using underground trains
- Sluggish network are even more the norm
  - anywhere really

The University Of Sheffield.

Towards a different Web Experience

- The real advantage of apps is the ability to
  - design for the specific device
    - e.g. using Android's or iOS's own design style
  - use the phone's sensors
    - HTML can use the GPS and camera only (although things are changing)
  - provide a perfect offline experience
    - Content may not be updated if offline but apps can take opportunistic strategies
      - to download content when online
      - to store content for off line viewing
  - allow push notifications even when the user is using a different app
- The HTTP protocol does not allow any of the above
  - although HTML5 allows storage of content to a limited extent

Progressive Web Apps

Prof Fabio Ciravegna
Department of Computer Science
The University of Sheffield
f.ciravegna@shef.ac.uk

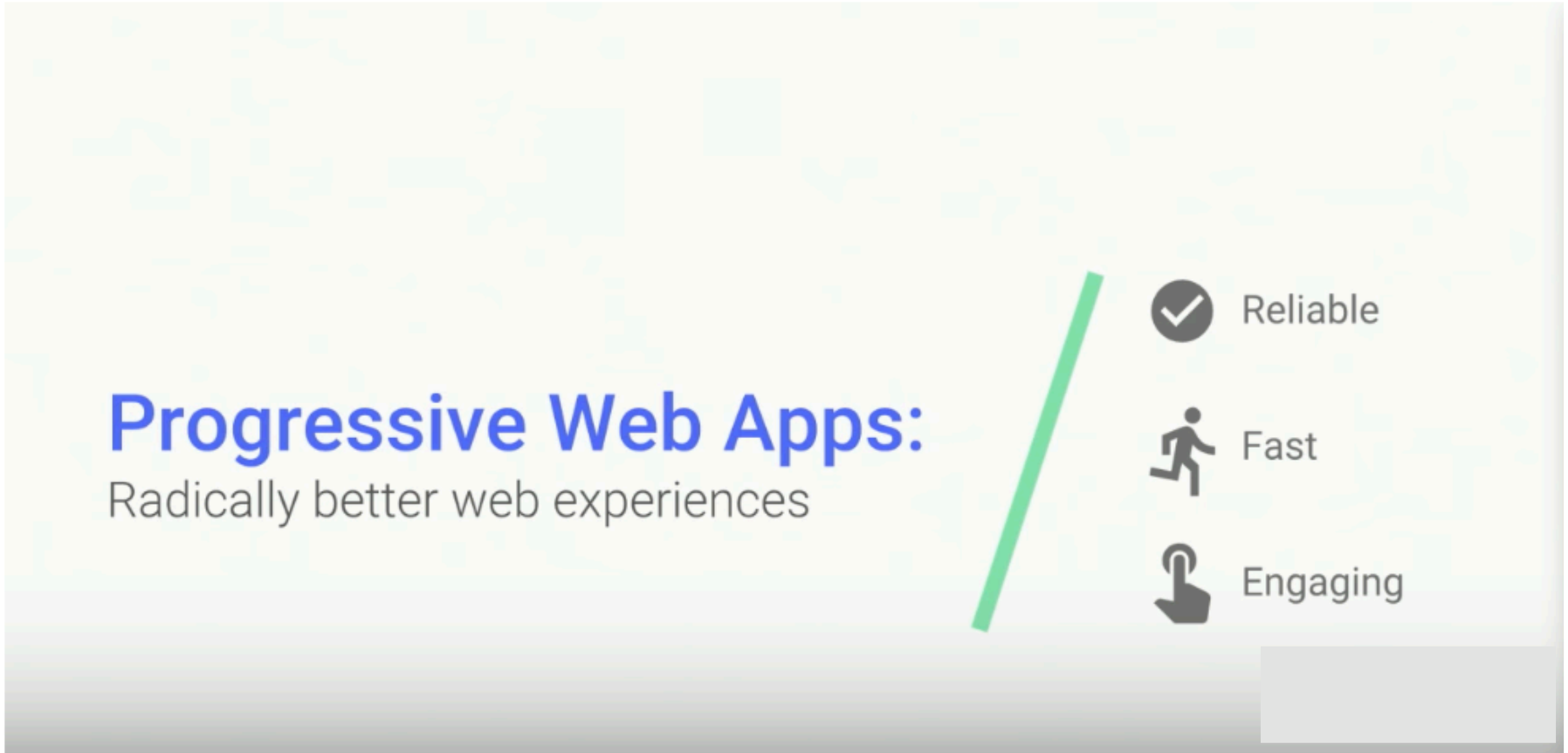
The University Of Sheffield.

Towards a different Web Experience

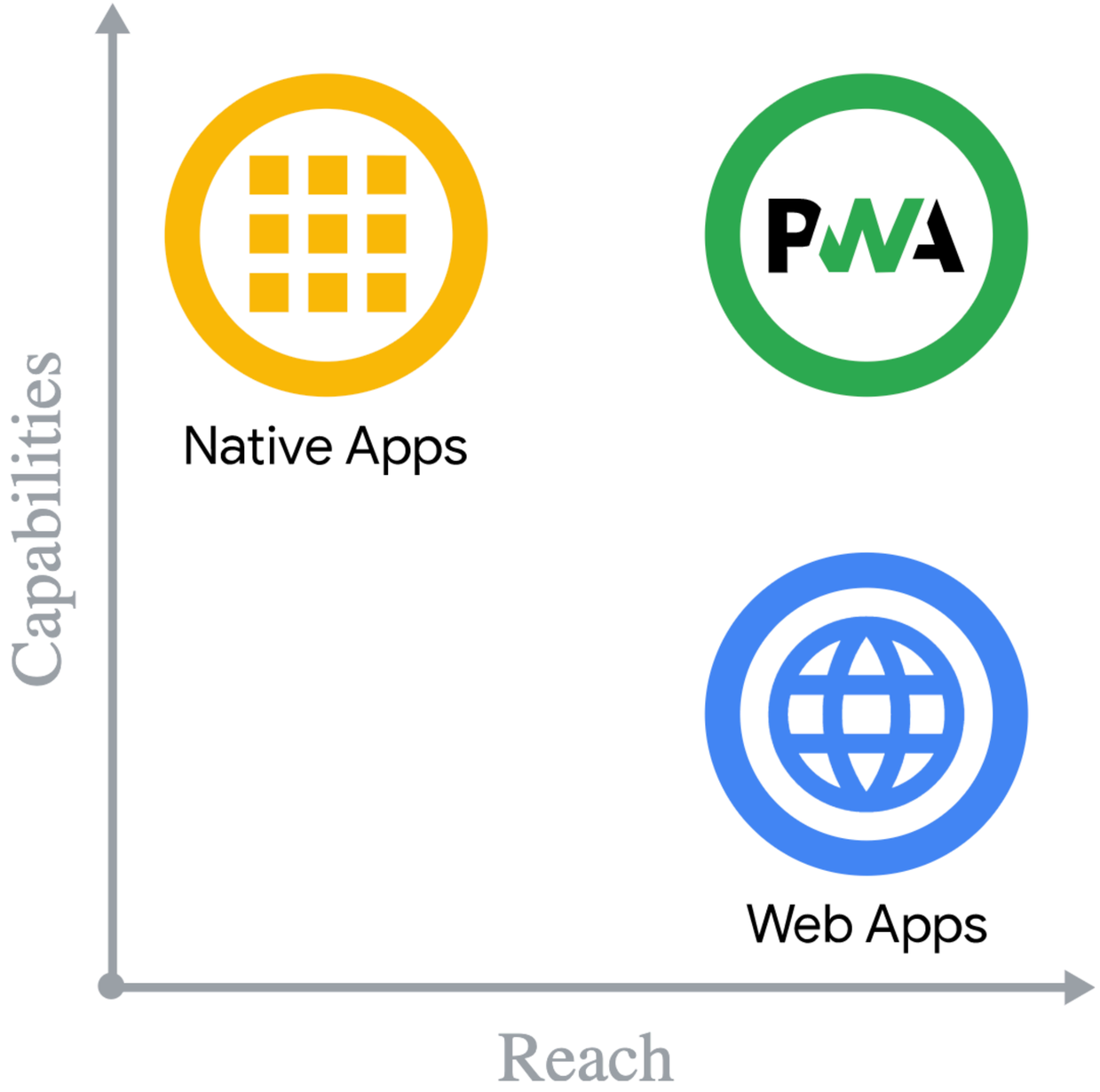
- 53% of users will abandon a site if it takes longer than 3 seconds to load!
  - Once loaded, users expect them to be fast—no janky scrolling or slow-to-respond interfaces
- The real advantage of apps is the ability to
  - provide a perfect offline experience
    - Content may not be updated if offline but apps can take opportunistic strategies
      - to download content when online
      - to store content for off line viewing
  - allow push notifications even when the user is using a different app

Progressive Web Apps

- Are websites that provide a native app experience without downloading an app
  - they use web service workers



The University Of Sheffield.



Capabilities vs. reach of native apps, web app, and progressive web apps.

https://web.dev/what-are-pwas/

The University Of Sheffield.

PWAs

https://en.wikipedia.org/wiki/Progressive_web_app

- Progressive - Work for every user, regardless of browser choice because they provide a better experience when the browser is better equipped (i.e. there is a fallback position for the old browsers)
- Responsive - Fit any form factor: desktop, mobile, tablet, or forms yet to emerge.
- Connectivity independent - Service workers allow work offline, or on low quality networks.
- App-like - Feel like an app to the user with app-style interactions and navigation.
- Fresh - Always up-to-date thanks to the service worker update process.
- Safe - Served via HTTPS to prevent snooping and ensure content hasn't been tampered with.
- Discoverable - Are identifiable as "applications" thanks to W3C manifests[6] and service worker registration scope allowing search engines to find them.
- Re-engageable - Make re-engagement easy through features like push notifications.
- Installable - Allow users to "keep" apps they find most useful on their home screen without the hassle of an app store.
- Linkable - Easily shared via a URL and do not require complex installation

The University Of Sheffield.

Application shell architecture

- Some progressive web apps use an architectural approach called the App Shell Model
- For rapid loading, service workers store the Basic User Interface or "shell" of the responsive web design web application.
- This shell provides an initial static frame, a layout or architecture into which content can be loaded progressively as well as dynamically,
  - allowing users to engage with the app despite varying degrees of web connectivity.
  - Technically, the shell is a code bundle stored locally in the browser cache of the mobile device

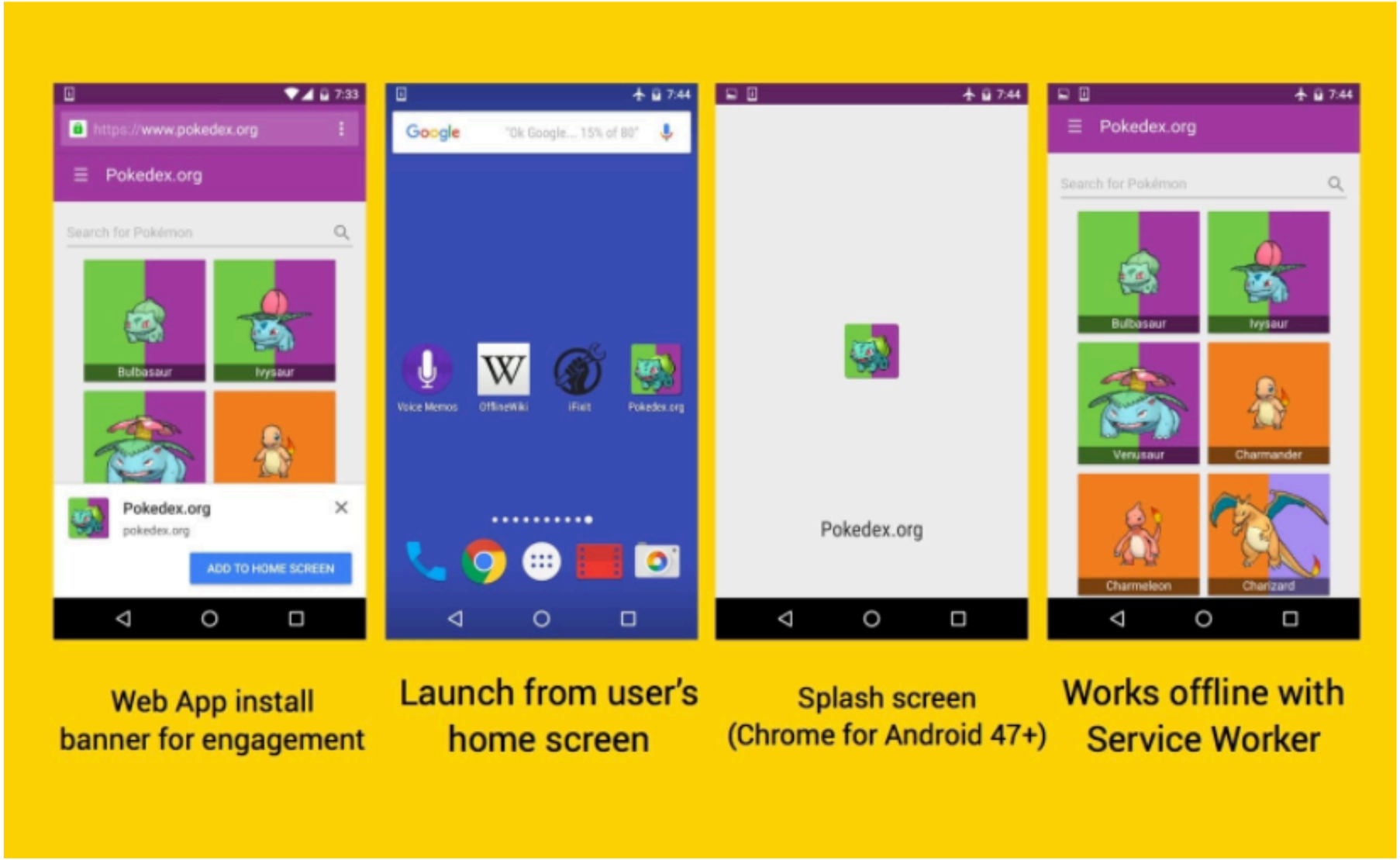
The University Of Sheffield.

Advantages

- Cost saving in terms of development and maintenance
  - Average customer acquisition costs may be up to ten times smaller compared to those of native applications
  - (this claim is disputable but it is good to understand the potential impact)
- Always up to date
  - one of the major issues of apps is that the average user does not update native apps
    - in my experience 30% to 50% of users never do!!
    - this is a serious issue in terms of effectiveness and ability to fix issues

The University Of Sheffield.

PWAs



https://developers.google.com/web/updates/2015/12/getting-started-pwa

PWAs

Progressive Web Apps are user experiences that have the reach of the web, and are:

- **Reliable** - Load instantly and never show the downasaur, even in uncertain network conditions.
- **Fast** - Respond quickly to user interactions with silky smooth animations and no janky scrolling.
- **Engaging** - Feel like a natural app on the device, with an immersive user experience.

This new level of quality allows Progressive Web Apps to earn a place on the user's home screen.

https://developers.google.com/web/progressive-web-apps/

The University Of Sheffield.

Based on Web Service Workers

Reliable

When launched from the user's home screen, service workers enable a Progressive Web App to load instantly, regardless of the network state.

A service worker, written in JavaScript, is like a client-side proxy and puts you in control of the cache and how to respond to resource requests. By pre-caching key resources you can eliminate the dependence on the network, ensuring an instant and reliable experience for your users.

LEARN MORE

The University Of Sheffield.

No app store

Engaging

Progressive Web Apps are installable and live on the user's **home screen**, without the need for an app store. They offer an **immersive full screen** experience with help from a web app manifest file and can even re-engage users with web **push notifications**.

The Web App Manifest allows you to control how your app appears and how it's launched. You can specify home screen icons, the page to load when the app is launched, screen orientation, and even whether or not to show the browser chrome.

WEB APP MANIFEST WEB PUSH NOTIFICATIONS

One of the nice aspects of the "progressive" nature to this model is that features can be gradually unlocked as browser vendors ship better support for them.

The University Of Sheffield.

Why build a Progressive Web App?

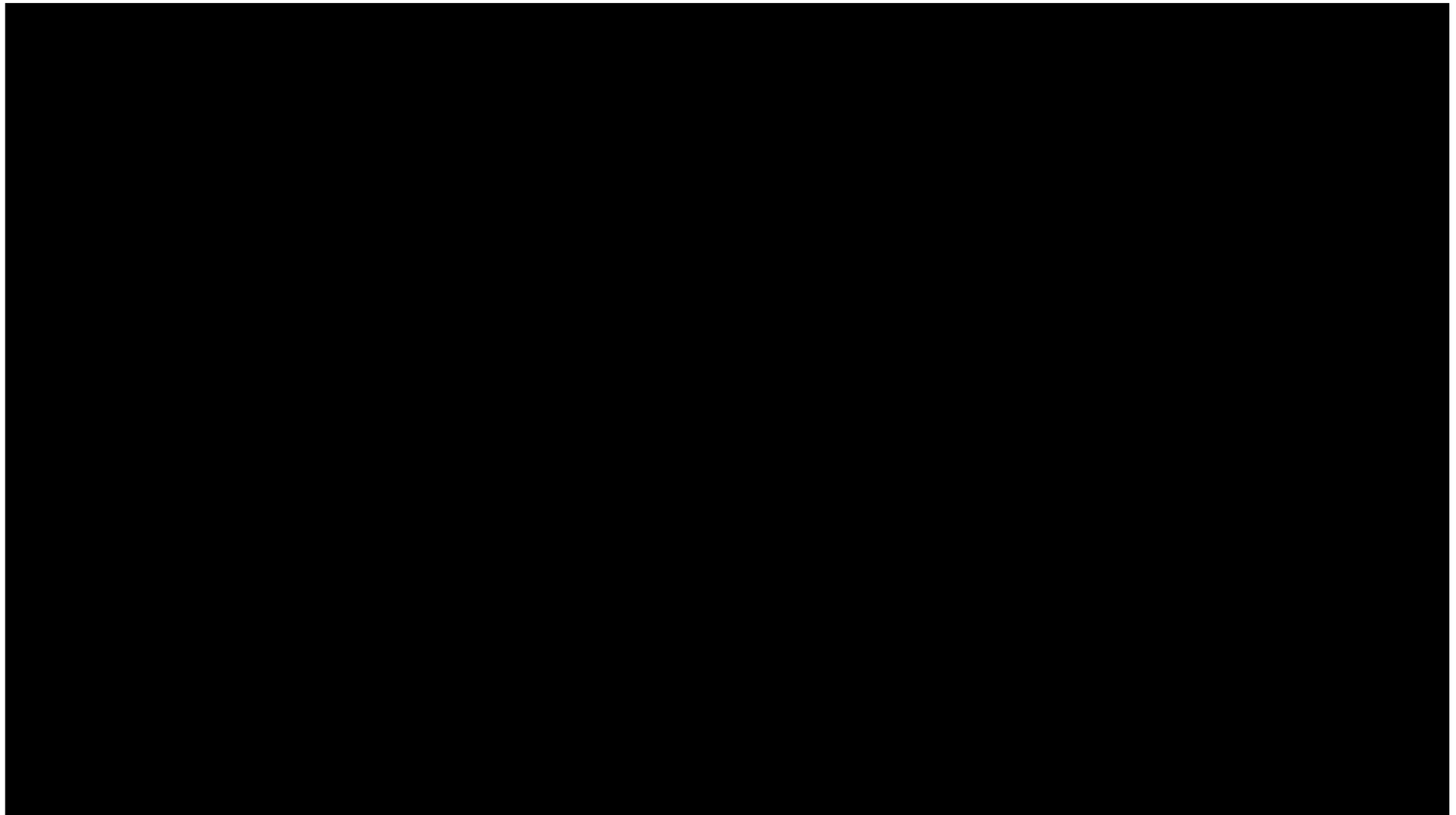
Building a high-quality Progressive Web App has incredible benefits, making it easy to delight your users, grow engagement and increase conversions.

- ✓ **Worthy of being on the home screen**
  When the Progressive Web App criteria are met, Chrome prompts users to add the Progressive Web App to their home screen.
- ✓ **Work reliably, no matter the network conditions**
  Service workers enabled Konga to send 63% less data for initial page loads, and 84% less data to complete the first transaction!
- ✓ **Increased engagement**
  Web push notifications helped eXtra Electronics increase engagement by 4X. And those users spend twice as much time on the site.
- ✓ **Improved conversions**
  The ability to deliver an amazing user experience helped AliExpress improve conversions for new users across all browsers by 104% and on iOS by 82%.

The University Of Sheffield.

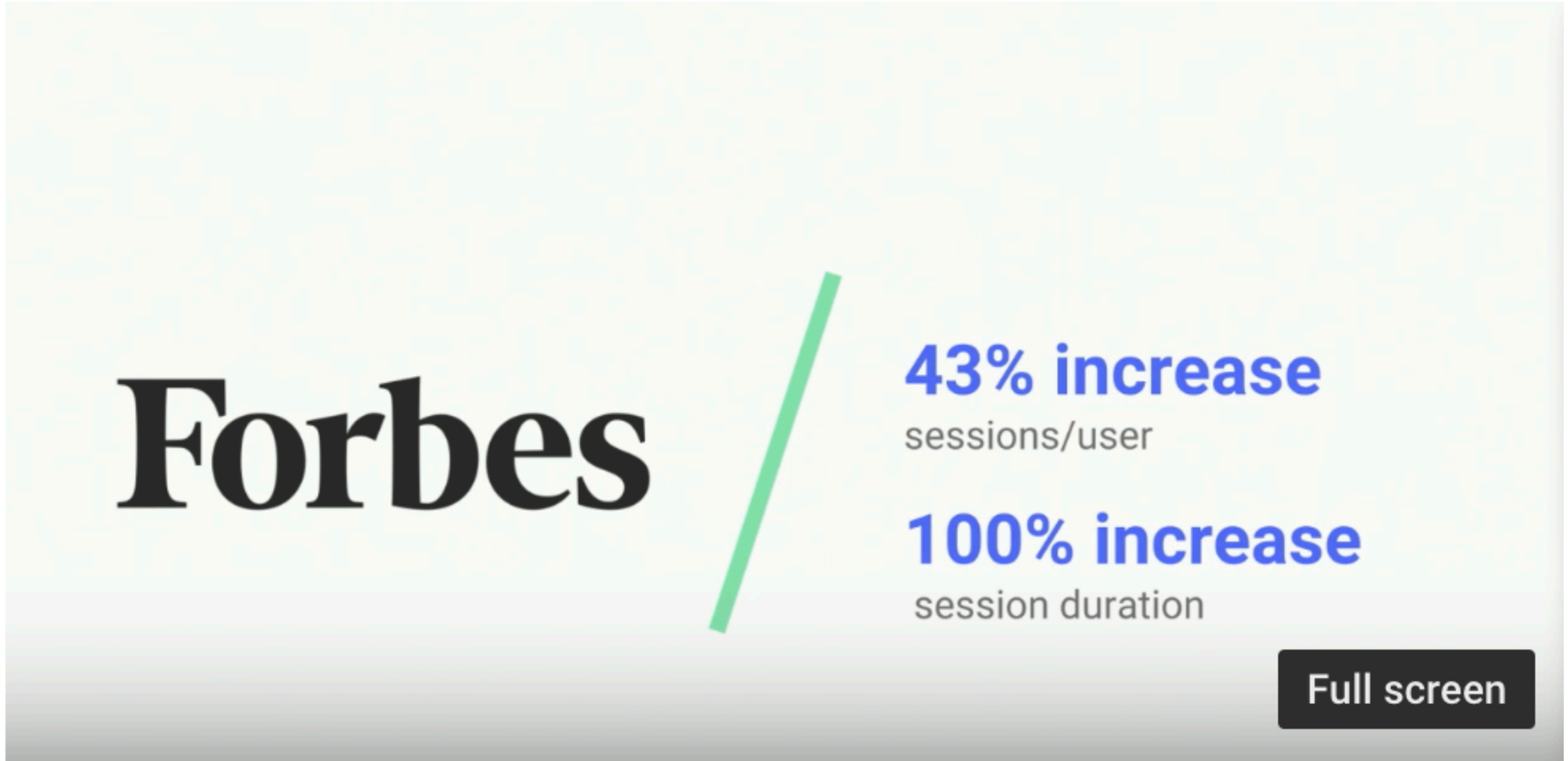
Not just a web site

- A fully fledged app with notifications and offline usage



Why?

- The point is to make sure people come and access your service (rather than website)
  - that is the whole point of having apps. With a progressive app:



Manifest file

https://developer.mozilla.org/en-US/docs/Web/Manifest

- A manifest file is a file containing metadata for a group of accompanying files that are part of a set or coherent unit
  - For example, the files of a computer program may have a manifest describing the name, version number, license and the constituting files of the program
- This is a generic concept that is specialised by the Web Manifest
  - The web app manifest provides information about an application (such as name, author, icon, and description) in a JSON text file.
  - The purpose of the manifest is to install web applications to the home screen of a device, providing users with quicker access and a richer

The University Of Sheffield.

Minimum Requirements

- The name of the web application
- Links to the web app icons or image objects
- The preferred URL to launch or open the web app
- The web app configuration data for a number of characteristics
- Declaration for default orientation of the web app (e.g. portrait)
- Enables to set the display mode e.g. full screen

Example

```
{  "name": "HackerWeb",
  "short_name": "HackerWeb",
  "start_url": ".",
  "display": "standalone",
  "background_color": "#fff",
  "description": "A simply readable Hacker News app.",
  "icons": [{
  "src": “images/touch/homescreen48.png", "sizes": “48x48","type": "image/png"
  }, {
  "src": "images/touch/homescreen72.png","sizes": "72x72","type": "image/png"
  }, {
  "src": "images/touch/homescreen96.png","sizes": "96x96","type": "image/png"
  }, {
  "src": "images/touch/homescreen144.png","sizes": "144x144","type": "image/png"
  }, {
  "src": "images/touch/homescreen168.png","sizes": "168x168","type": "image/png"
  }, {
  "src": "images/touch/homescreen192.png","sizes": "192x192","type": "image/png"
  }],
  "related_applications": [{
    "platform": "play",
    "url": "https://play.google.com/store/apps/details?id=cheeaun.hackerweb"
  }]
}
```

The University Of Sheffield.

App Shell

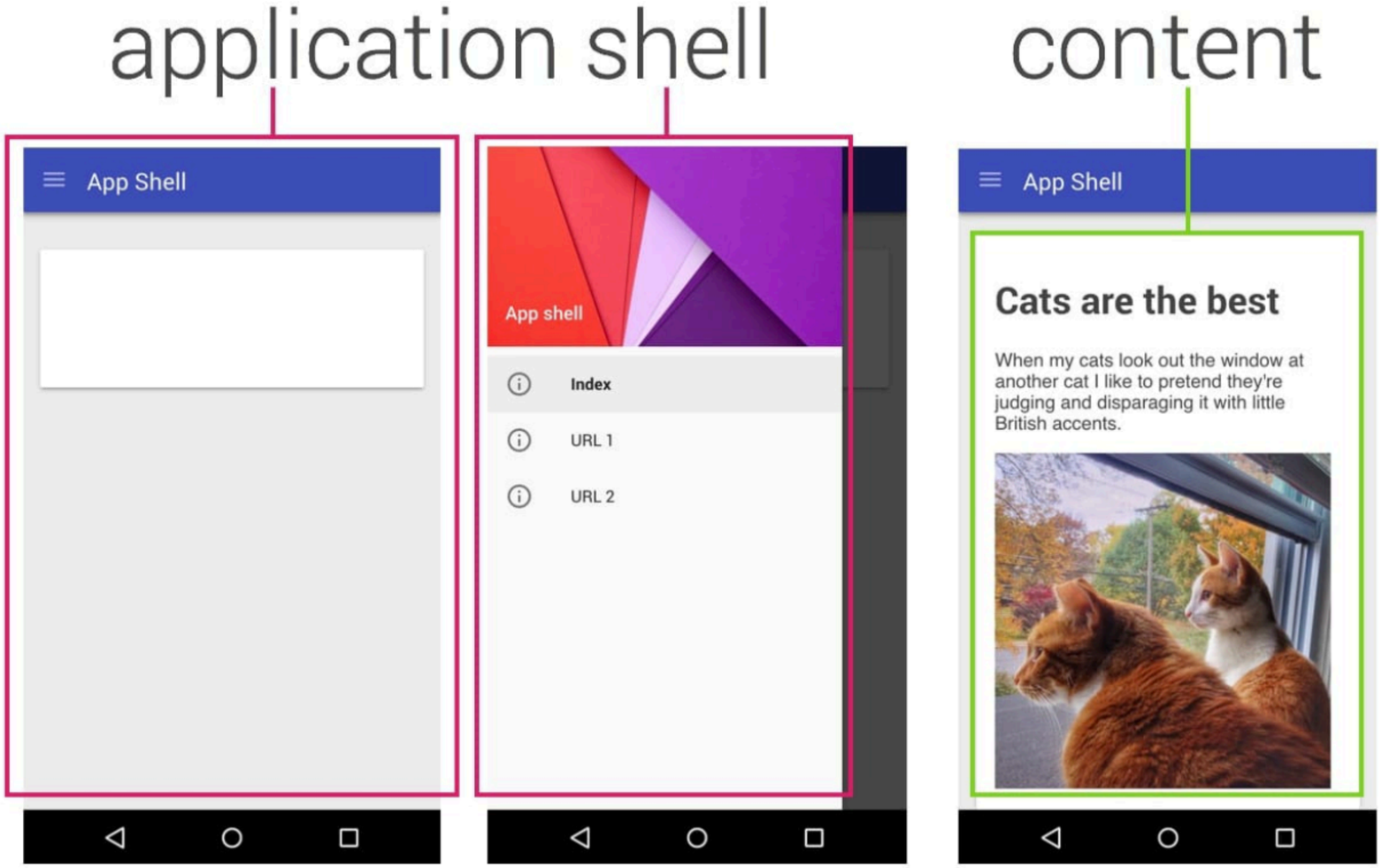
- The app's shell is the minimal HTML, CSS, and JavaScript that is required to power the user interface of PWa
  - the components that ensures reliably good performance also in limited connectivity conditions
  - Its first load should be extremely quick and immediately cached
    - "Cached" means that the shell files are loaded once over the network and then saved to the local device.
  - Every subsequent time that the user opens the app, the shell files are loaded from the local device's cache, which results in blazing-fast startup times
  - **(you will have guessed that that is done using a Service Worker!!)**

https://developers.google.com/web/fundamentals/codelabs/your-first-pwapp/

The University Of Sheffield.

- App shell architecture separates the core application infrastructure and UI from the data
- All of the UI and infrastructure is cached locally using a service worker
  - so that on subsequent loads, the Progressive Web App only needs to retrieve the necessary data
- For example a Facebook like app will have
    - as shell the interface containing the news feeds and the different pages (profile, etc.)
    - as cashed data the news you have already downloaded, your profile information etc.
    - as downloaded data, the updates of news, to profile, etc,

Example



Cached shell loads **instantly** on repeat visits.

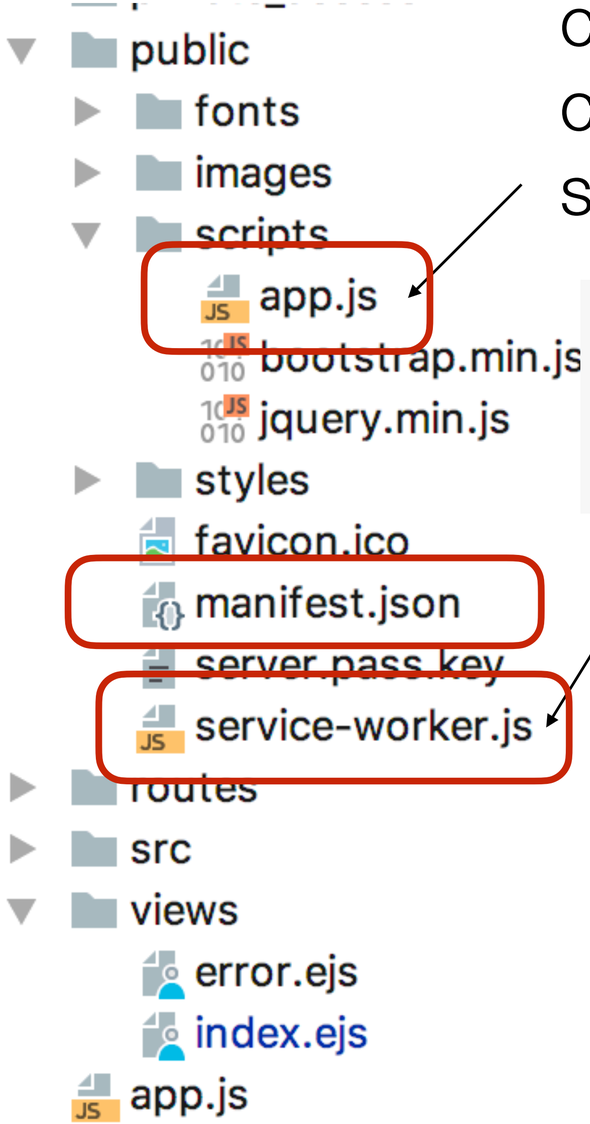
Dynamic content then populates the view

Designing the shell

- The first step is to break the design down into its core components.
- Ask yourself:
  - What needs to be on screen immediately?
  - What other UI components are key to our app?
  - What supporting resources are needed for the app shell?
    - For example images, JavaScript, styles, etc.

With Service Worker

App organisation



code loaded by index.ejs, it will contain the declaration of the service worker

```
if ('serviceWorker' in navigator) {
   navigator.serviceWorker
            .register('./service-worker.js')
            .then(function() {
                  console.log('Service Worker Registered'); });
 }
```

The service worker: it contains the list of shell files and the install&fetch events capturing declarations

```
var cacheName = 'weatherPWA-step-6-1';
var filesToCache = [...];

self.addEventListener('install', function(e) {
  console.log('[ServiceWorker] Install');
  e.waitUntil(
    caches.open(cacheName).then(function(cache) {
      console.log('[ServiceWorker] Caching app shell');
      return cache.addAll(filesToCache);
    })
```


```
var dataCacheName = 'weatherData-v1';
var cacheName = 'weatherPWA-step-8-1';
var filesToCache = [
    '/',     '/scripts/app.js','/styles/inline.css','/styles/bootstrap.min.css',
    '/scripts/bootstrap.min.js','/scripts/jquery.min.js','/scripts/database.js',
    '/fonts/glyphicons-halflings-regular.woff2','/fonts/glyphicons-halflings-regu
    '/fonts/glyphicons-halflings-regular.ttf'];
/**
 * installation event: it adds all the files to be cached
 */
self.addEventListener('install', function (e) {
    console.log('[ServiceWorker] Install');
    e.waitUntil(
        caches.open(cacheName).then(function (cache) {
            console.log('[ServiceWorker] Caching app shell');
            return cache.addAll(filesToCache);}) );});
/**
 * activation of service worker: it removes all cashed files if necessary
 */
self.addEventListener('activate', function (e) {
    console.log('[ServiceWorker] Activate');
    e.waitUntil(
        caches.keys().then(function (keyList) {
            return Promise.all(keyList.map(function (key) {
                if (key !== cacheName && key !== dataCacheName) {
                    console.log('[ServiceWorker] Removing old cache', key);
                    return caches.delete(key); }}));}));
```


```
self.addEventListener('fetch', function(e) {
  console.log('[Service Worker] Fetch', e.request.url);
    /*
    * The app is asking for app shell files. In this scenario the app uses the
    * "Cache, falling back to the network" offline strategy:
    * https://jakearchibald.com/2014/offline-cookbook/#cache-falling-back-to-network
    */
   e.respondWith(
     caches.match(e.request).then(function(response) {
       return response
           || fetch(e.request)
               .then(function(response) {
               // note if network error happens, fetch does not return
              // an error. it just returns response not ok
             // https://www.tjvantoll.com/2015/09/13/fetch-and-errors/
                   if (!response.ok) {
                       console.log("error: " + err);
                   }
               }) // here we capture HTTP errors such as 404 file not found
               .catch(function (e) {
                   console.log("error: " + err);
               })
     })
   );
```

The University Of Sheffield.

Summary

- You should understand the motivations behind the use of PWA
- You should remember how they are organised in WebStorm
  - This will come useful when programming the PWA in the lab
- You should be able to manage the manifest file
  - This will come useful when programming the PWA in the lab
- You should be able to run a PWA with an appropriate caching strategy for the Service Worker

NoSQL databases for the Web

Prof. Fabio Ciravegna
Department of Computer Science
The University of Sheffield
f.ciravegna@shef.ac.uk

The University Of Sheffield.

Scaling up to Web Size

- 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
- We will see the details in the lecture on Search Engines

The University Of Sheffield.

Limits of SQL databases

- 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.

ctd

- 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 my applications with 1 million users sending location data at high velocity
      - conversion from JSON to relational data was the single bottleneck that caused severe pain and required a large and highly parallel architecture
- Join operations are the deathbed of efficiency

The University Of Sheffield.

No SQL databases

- Do not enforce a strict schema
- Provide
  - Native sharding for horizontal scaling
    - i.e. distribution of data on multiple machines along cloud computing paradigm of flexibility
      - If you need more resources, just add more nodes (as opposed to upgrade the server size)
      - if you need less resources, remove some nodes
  - Auto replication of data which in case of failure will return to the last consistent state
    - a requirement in large computing centres where failure is an expected situation

The University Of Sheffield.

ctd

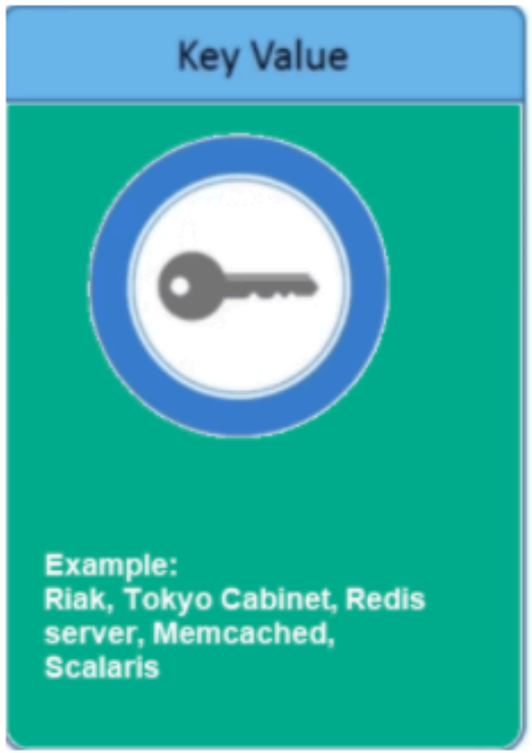
- Eventual Consistency
  - copies of data are stored on multiple machines to get high availability and scalability
  - Changes made to any data item on one machine has to be propagated to other replicas (and will eventually be propagated)
- BASE: Basically Available, Soft state, Eventual consistency
  - Basically, available means DB is available all the time
  - Soft state means even without an input; the system state may change
  - Eventual consistency means that the system will become consistent over time
- No single point of failure

The University Of Sheffield.

Issues

- 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

Types



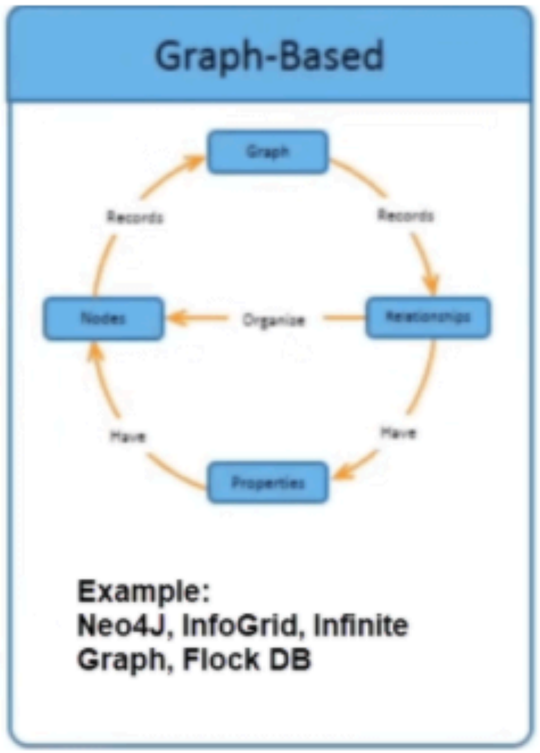


image from guru99

Types of NoSQL databases

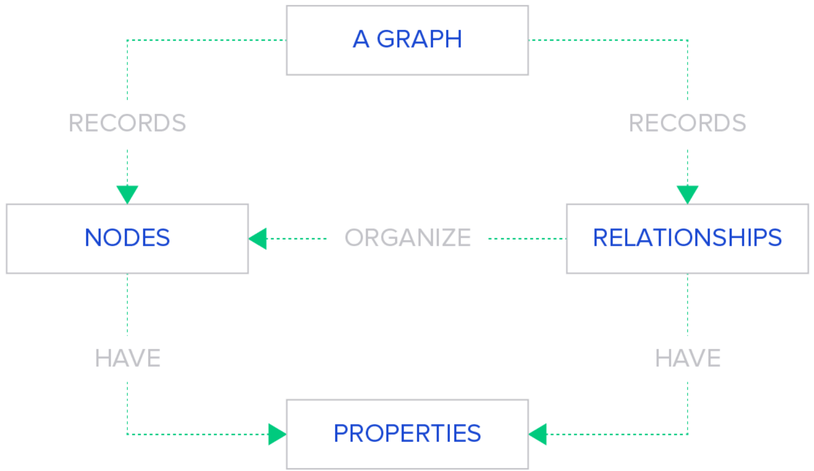
- Key value stores
  - Like hash tables

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"}

- Document Oriented
  - as in key values but data is stored as a key value pair
    - but the value is a document (e.g. composed of multiple fields)
    - stored in JSON or XML formats
    - this has a number o advantages in terms of querying

Types (cdt)

- Column Databases
  - 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)
  - created by Google for their core search engine storage system
- Graph Databases
  - a directed graph structure is used to represent the data
  - graphs are composed of edges and nodes
  - typically used in social networking applications. G
    - focus more on relations than on objects



	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.

What you should know

- Understand the limitations of SQL databases
- Understand the large scale requirements for the web
- Understand the motivation behind NoSQL databases
- Understand the types of noSQL databases

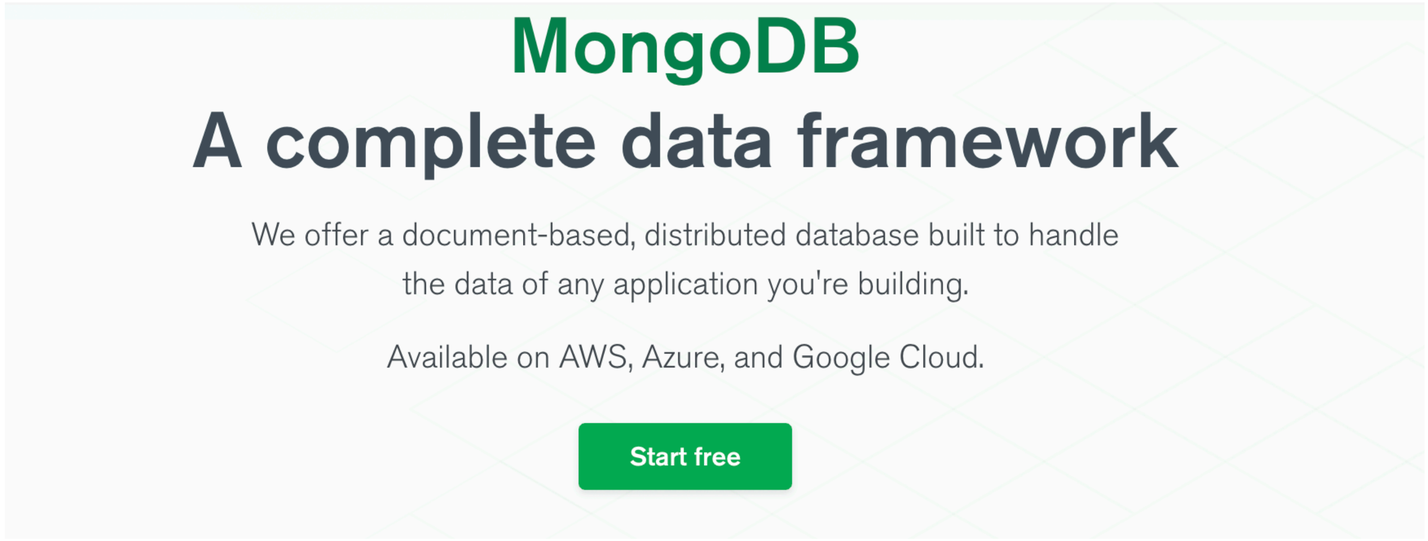
Questions?

Persisting Data with Mongo DB

Prof. Fabio Ciravegna
Department of Computer Science
The University of Sheffield
f.ciravegna@shef.ac.uk

MongoDB

- MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling



invision Adobe Google EA

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

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.

A restaurant **document** example


```
{
  "_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"
}
```

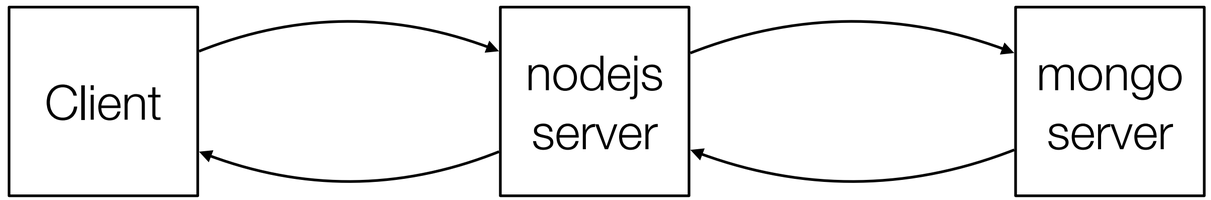
BSON

- Mongo's documents are internally represented as binary JSON
  - this is why in express you are likely to have to include the npm module called bson

https://www.slideshare.net/mdirolf/introduction-to-mongodb

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.

Mongoose

https://www.npmjs.com/package/mongoose

```
to install use: npm i mongoose
```

npm Search packages Search Si

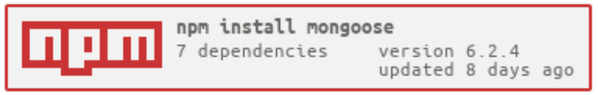
mongoose TS

6.2.4 • Public • Published 8 days ago

Readme | Explore BETA | 7 Dependencies | 11,784 Dependents | 710 Versions

Mongoose

Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment. Mongoose supports both promises and callbacks.



Documentation

The official documentation website is mongoosejs.com.

Mongoose 6.0.0 was released on August 24, 2021. You can find more details on backwards breaking changes in 6.0.0 on our docs site.

Install

```
npm i mongoose
```

Repository

github.com/Automattic/mongoose

Homepage

mongoosejs.com

♥Fund this package

Weekly Downloads

1,626,241

Version License

Creating a Database

- Install MongoDB and start the process
  - see lab class
- In your Express programme import Mongoose

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

- Create a database in MongoDB:
  - start by creating a MongoClient object,
  - then specify a MongoDB 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.

https://www.w3schools.com/nodejs/nodejs_mongodb_create_db.asp

The University Of Sheffield.

```
var MongoClient = require('mongodb').MongoClient;

// the client is running on a server and has an individual URL
// the URL works on the mongodb protocol.
// The server runs on a specific port, typically
// on 27017 or 27019
// in my examples I can use either - make sure to use
// the port you run your installation on
// In this case the database name is mydb

var url = "mongodb://localhost:27017/mydb";

// connect to the client. If the db does not exist,
// Mongoose will create it
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  console.log("Database created!");
  db.close();
});
```

The University Of Sheffield.

Connecting using await

```
configDB.url= mongodb://localhost:27017/database_name
```

```
await mongoose.connect(configDB, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  useFindAndModify: false,
  useCreateIndex: true
});
```

Mongoose buffers all the commands until it's connected to the database. This means that you don't have to wait until it connects to MongoDB in order to define models, run queries, etc.
but I suggest always to wait otherwise you will get a huge number of consecutive errors if the connection fails

Schemas

- Although mongo is schema-less, it helps to define schemas in order to
  - Validate data integrity of documents
  - Legibility/maintenance
    - pretty much like in Javascript where although the same variable can contain different data structures, you end up specialising the variables to only one type
- Mongoose imposes a schema for document models

The University Of Sheffield.

- A schema is the equivalent of a java interface
  - It is not a class that can be used to create instances
  - it must be implemented in a MODEL before being used to create instances

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

Aside from defining the structure of your documents and the types of data you're storing, a Schema handles the definition of:

- Validators (async and sync)
- Defaults
- Getters
- Setters
- Indexes
- Middleware
- Methods definition
- Statics definition
- Plugins
- pseudo-JOINs
-

The University Of Sheffield.

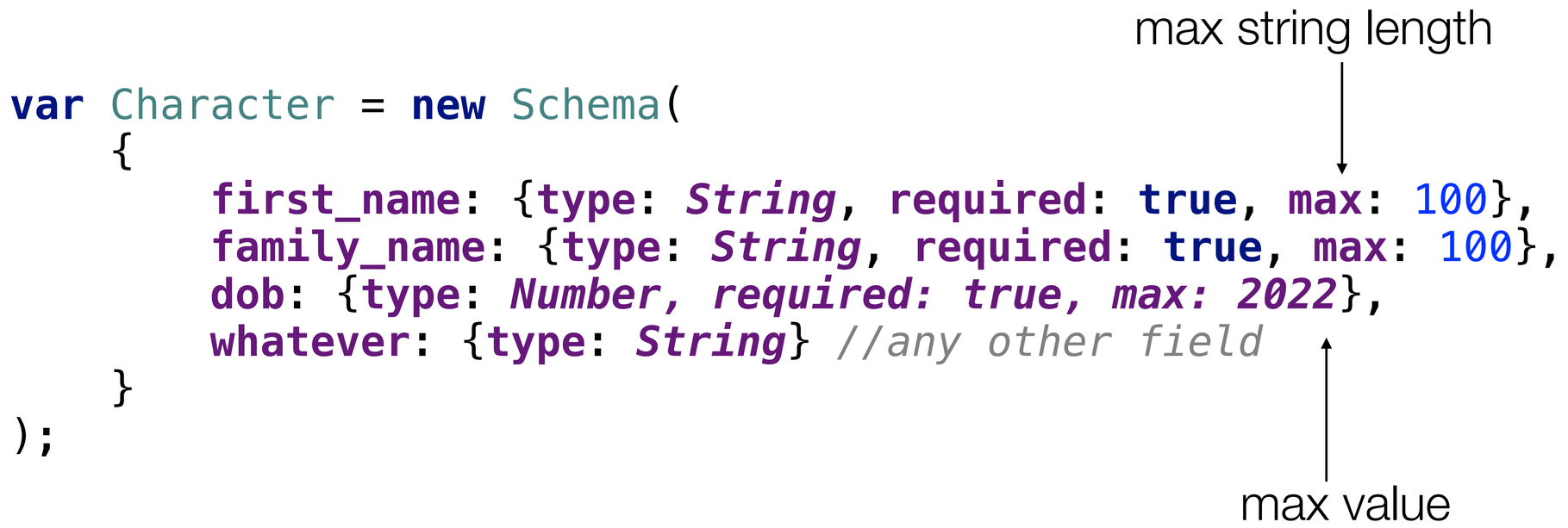
Validation

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.

The built-in validators include:

- All SchemaTypes have the built-in required validator. This is used to specify whether the field must be supplied in order to save a document.
- 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.

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: 2022},
        whatever: {type: String} //any other field
    }
);
```

see details at http://mongoosejs.com/docs/validation.html

The University Of Sheffield.

Validation example

```
const breakfastSchema = new Schema({
  eggs: {
    type: Number,
    min: [6, 'Too few eggs'],
    max: 12
  },
  bacon: {
    type: Number,
    required: [true, 'Why no bacon?']
  },
  drink: {
    type: String,
    enum: ['Coffee', 'Tea'],
    required: function() {
      return this.bacon > 3;
    }
  }
});
```

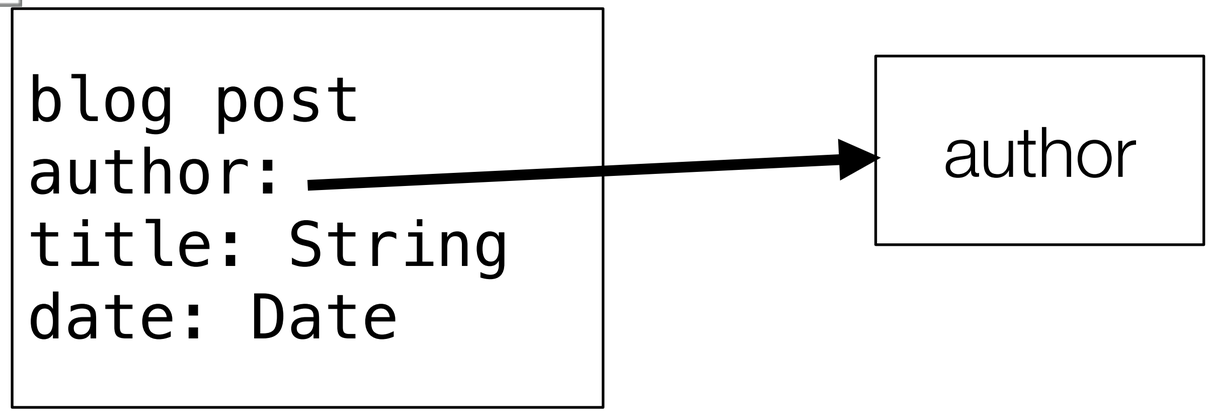
Virtual properties

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose

- Document properties that you can get and set but that do not get persisted to MongoDB
- The getters are useful for formatting or combining fields,
  - It is easier and cleaner and uses less disk space
  - it allows for dynamic properties
- Examples:
  - 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;
});
```

Objects in fields




```
const Schema = mongoose.Schema;
const ObjectId = Schema.ObjectId;

const BlogPost = new Schema({
  author: ObjectId,
  title: String,
  body: String,
  date: Date
});
```

- `ObjectId`: Represents specific instances of a model in the database. For example, a book might use this to represent its author object. This will actually contain the unique ID (`_id`) for the specified object.

Middleware : pre/post operations

- Used to perform operations before saving (for example to notify an administrator)

```
schema.pre('set', function (next, path, val, typel) {
  // `this` is the current Document
  this.emit('set', path, val);

  // Pass control to the next pre
  next();
});
```

- you can mutate the incoming method arguments so that subsequent middleware see different values for those arguments. To do so, just pass the new values to next

```
.pre(method, function firstPre (next, methodArg1, methodArg2) {
  // Mutate methodArg1
  next("altered-" + methodArg1.toString(), methodArg2);
});
```

Models implement Schemas

- 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
  - find,
  - create,
  - update,
  - delete

} instances of that model (i.e. records in the db)

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose

Models

```
const MyModel = mongoose.model('ModelName', mySchema);
```

- This creates a model from the schema
  - The model is the equivalent to a Java Class
    - You can create instances off it

```
const instance = new MyModel();
//do some operations here e.g. by initialising the fields
instance.save(function (err) {
   //
});
```

Instances with field initialisation

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

```
var character = new Character({
    first_name: 'Mickey',
    family_name: 'Mouse',
    dob: 1908
});
```

then save it into the database (with callback)

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

Searching

- You can search for records using query methods, specifying the query conditions as a JSON document
  - e.g. find all athletes that play tennis, returning just the fields for athlete name and age
    - We just specify one matching field (sport), but you can
      - add more criteria,
      - specify regular expression criteria, or
      - remove the conditions altogether to return all athletes.

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose

```
var Athlete = mongoose.model('Athlete', yourSchema);

// find all athletes (MODEL!) who play tennis, selecting the 'name' and 'age' fields

Athlete.find({ 'sport': 'Tennis' }, 'name age', function (err, athletes) {
  if (err) return handleError(err);
  // 'athletes' contains the list of athletes that match the criteria.
})
```

{ 'sport': 'Tennis' } is the condition

- you will notice that posing conditions is visually similar to create an object with those fields

'name age' are the fields to be returned

Always remember to check for errors!!

The University Of Sheffield.

Example

```
var character = new Character({
    first_name: 'Mickey',
    family_name: 'Mouse',
    dob: 1908
});
console.log('dob: ‘+character.dob);

character.save(function (err, results) {
    if (err)  console.log(‘error! ’+ err);
  else console.log(results._id);
});


/** typical nodeJS call – you pass req and res as well as some conditions*/
function findCharacters(req, res, nameToFind){
   Character.find({first_name: some strings, family_name: some strings},
    'first_name family_name dob age’,  // these are the fields to return
     function (err, characters) {
        // you can return to the client directly from here
        if (err)
            res.status(500).send('Invalid data!’);
        else res.json(JSON.stringify(characters)
...
}
```

Mongoose operations are *Then*able

```
Band.findOne({name: "Guns N' Roses"})
  .then(doc => {
      // use doc
    })
  .catch (e => {
  })
```

Technically they are not promises but you can imagine them as such

Callback Parameters

- If you specify a callback the query will execute immediately while the callback will be invoked when the search completes.
- Note: All callbacks in Mongoose use the pattern
  - callback(error, result)
  - If an error occurs executing the query, the error parameter will contain an error document and result will be null.
  - If the query is successful, the error parameter will be null, and the result will be populated with the results of the query.

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose

The University Of Sheffield.

Querying step by step

```
// find all athletes that play tennis
var query = Athlete.find({ 'sport': 'Tennis' });

// selecting the 'name' and 'age' fields
query.select('name age');

// limit our results to 5 items
query.limit(5);

// sort by age
query.sort({ age: -1 });

// execute the query at a later time
query.exec(function (err, athletes) {
  if (err) return handleError(err);
  // athletes contains an ordered list of 5 athletes who play Tennis
})
```

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose

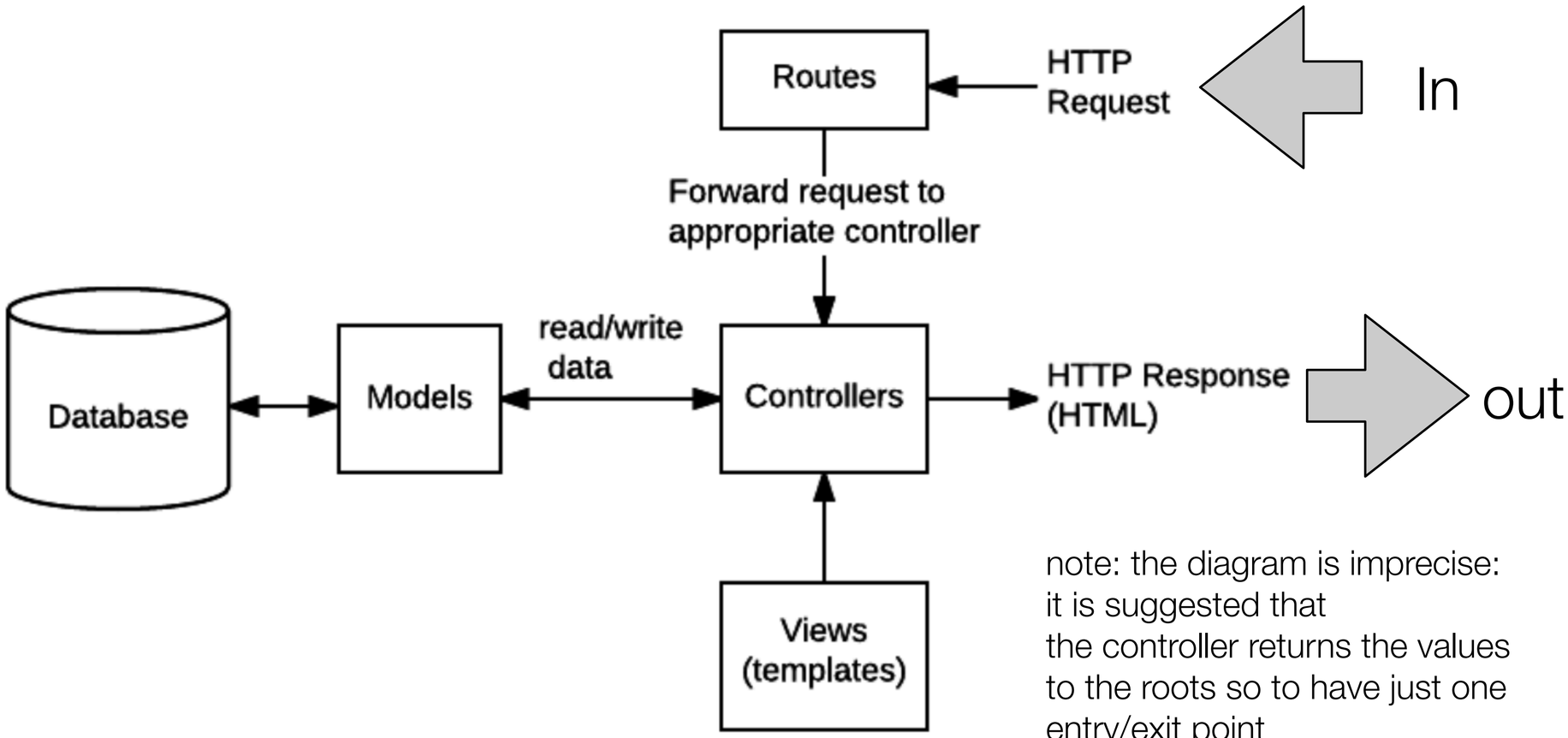
composing with dot

- Better method:
  - using a where() function and chaining all the parts of the query using the dot operator (.)
  - this is clearer than using separate statements or a big query with implicit parameters as in slide 26

```
Athlete.
  find().
  where('sport').equals('Tennis').
  where('age').gt(17).lt(50).  //Additional where query
  limit(5).
  sort({ age: -1 }).
  select('name age').
  exec(callback); // where callback is the name of our callback function.
```

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose

Typical project organisation



note: the diagram is imprecise:
it is suggested that
the controller returns the values
to the roots so to have just one
entry/exit point

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes

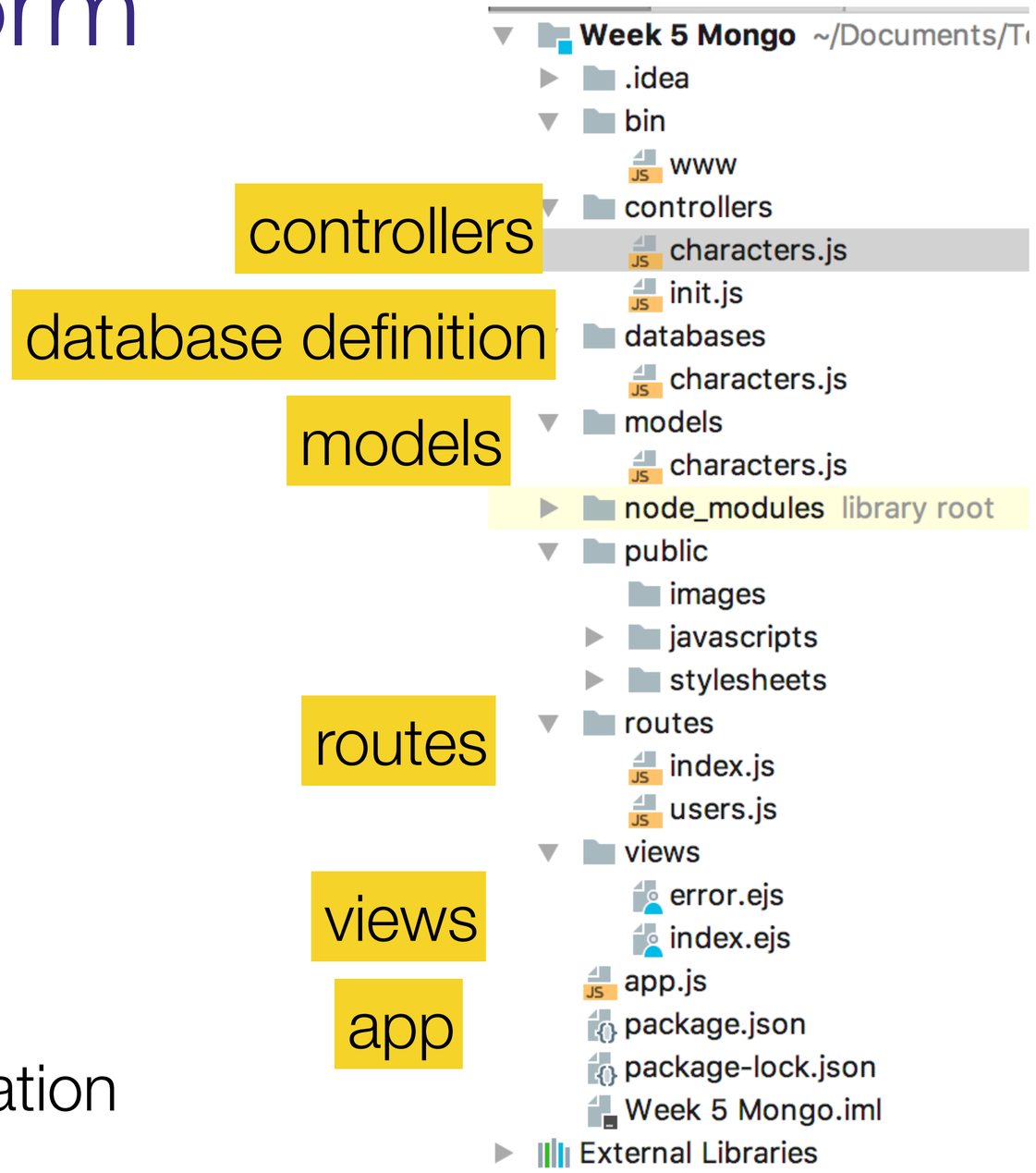
The University Of Sheffield.

…Organisation

- Routes to forward the supported requests (and any information encoded in request URLs) to the appropriate controller functions.
- Controller functions to get the requested data from the models and return it to the user (e.g. via JSON)
- Models: the declaration of the MongoDb documents types
- Views our enhanced-html pages

In WebStorm

- The program is organised in this way
- There is a database called ‘characters’
  - which has a model called ‘Character’ representing name, surname and year of birth of each character



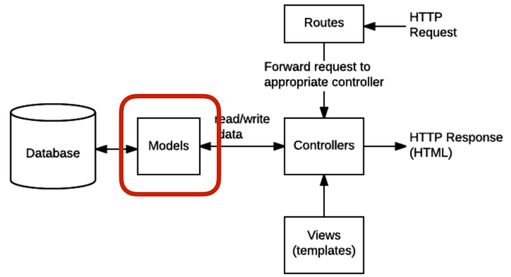
Note: you must learn this organisation and use it in your assignment!!

Routes

- Routes are the entry point of the database via the nodeJS server
- they will communicate with the controller which contains the database access functions

```
// require the controller so to be able to access its functions
var author_controller = require('../controllers/authorController');

// route example: GET a request for list of all Authors.
// author_list is a function defined in the controller file author_controller
// you may ask: if that is a function, why is it not called as author_list() ?
// Because in the route we declare the callback. The event handler will call it.
// see next route with the usual callback definition showing it is a function definition
// rather than a call
router.get('/authors', author_controller.author_list);

// route with callback declaration(a bit messier than the previous one but I prefer this because
// it separates the controller functions from the management of the routes, i.e. returning to the
client )
router.get(‘/authors’, function (req, res){ // you see? this is a definition rather than a call!
      author_list(req, res, function (err, data)
          if (!err){ res.writeHead(200, { "Content-Type": "application/json"});
                    res.end(JSON.stringify(data)); …
•
```

Organising Models

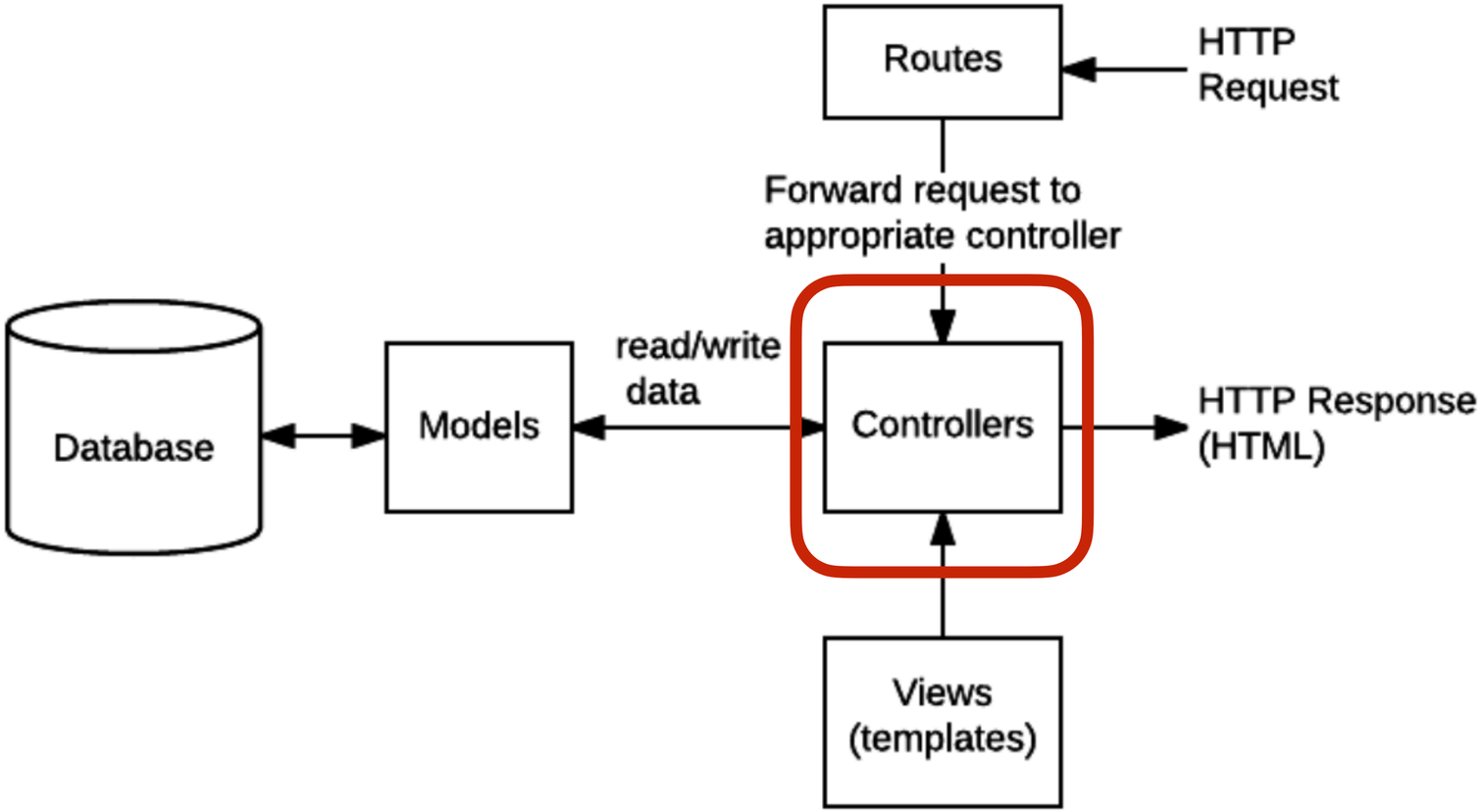


- It is recommended that you define just one schema + model per file
  - and then export the model

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

var Schema = mongoose.Schema;

var Character = new Schema({
        first_name: {type: String, required: true, max: 100},
        family_name: {type: String, required: true, max: 100},
        dob: {type: Number},
        whatever: {type: String} });

var characterModel = mongoose.model('Character', Character );

module.exports = characterModel;
```

we will see how to import the model in a controller in a few slides

Typical project organisation



https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes

Controllers

- Controllers are the route-handler callback functions
  - it is suggest not to insert too much code into the routes file(s)
    - so to keep the code clean and separated

Controllers

- They Import the models and use the model as an object, e.g.:

  - then they define the exported functions to be used in the routes
  - being a module, **we must export** the functions otherwise they will not be visible outside the module

```
// the controller must import the model(s) it works on
var Author = require('../models/author');

// Display list of all Authors. Remember to export the function outside the module
exports.author_list = function(req, res) {
    // here we will make operations on the database and return the data
    // for example here we could have a find operation to retrieve the authors list
     res.send('NOT IMPLEMENTED: Author list');
};
```

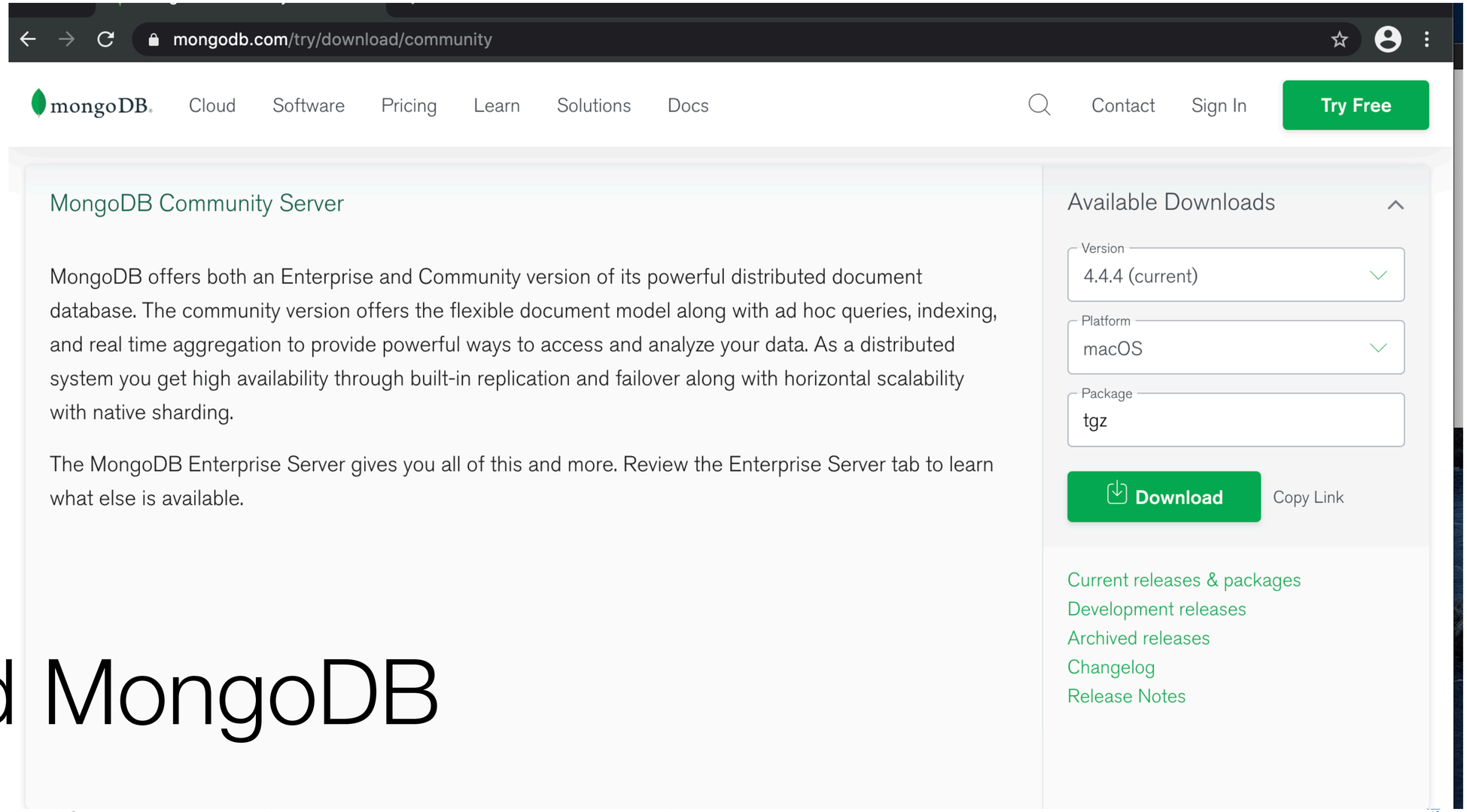
The database declaration

- create a js file called database.js under the folder /databases

```
let mongoose = require('mongoose');
//The URL which will be queried. Run "mongod.exe" for this to connect
mongoose.Promise = global.Promise;
let mongoDB = 'mongodb://localhost:27017/characters';
mongoose.Promise = global.Promise;
try {
    connection = mongoose.connect(mongoDB, {
        useNewUrlParser: true,
        useUnifiedTopology: true,
        checkServerIdentity: false,
    });
    console.log('connection to mongodb worked!');

} catch (e) {     console.log('error in db connection: ' + e.message);}
```

this will connect to the Mongo server and will create the characters database (if not existent)

The University Of Sheffield.



Download MongoDB

Before the next lab class if you are using your own computer!

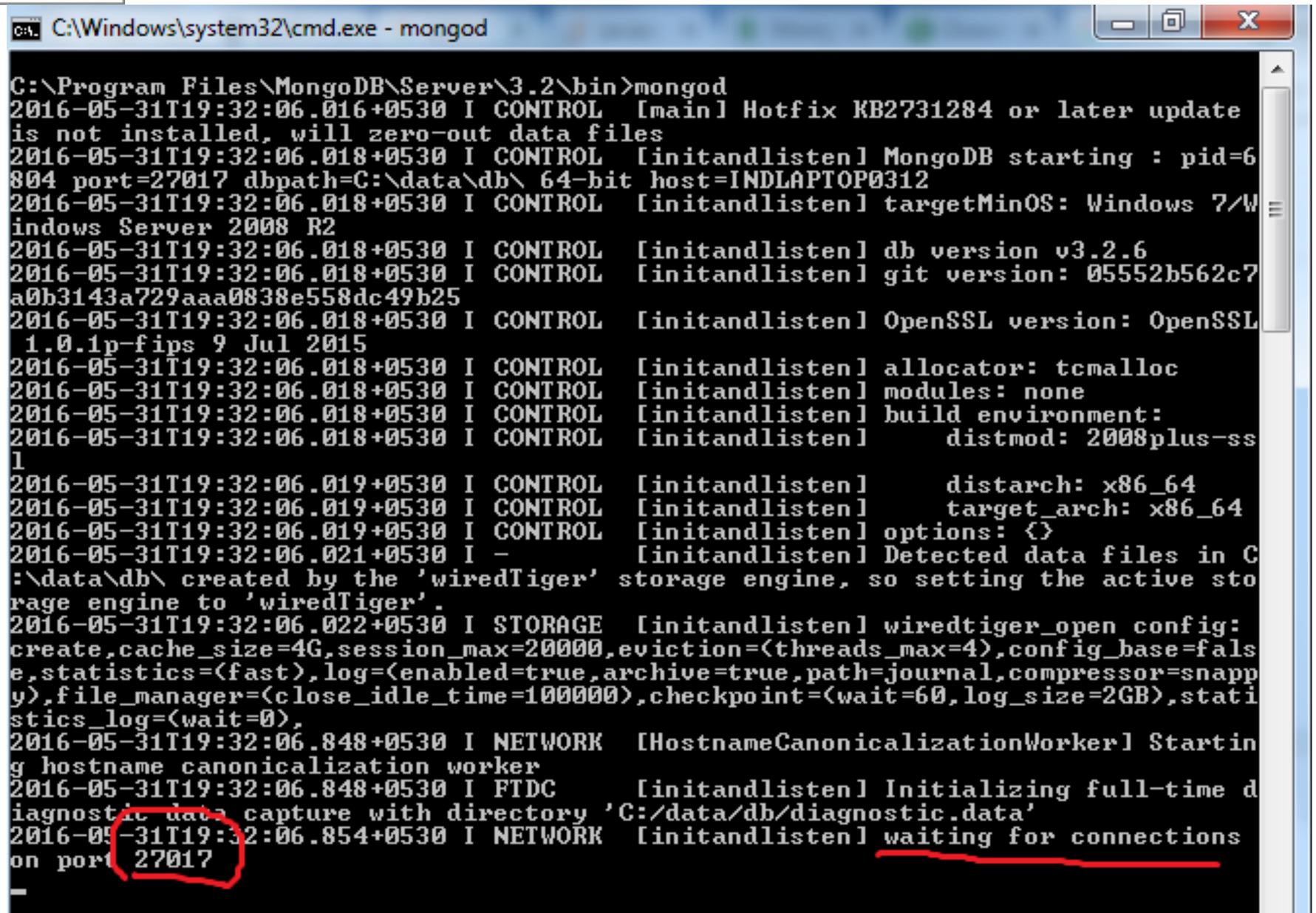
Download the community server
https://www.mongodb.com/try/download/community

MongoDB: Windows

- You should have already downloaded MongoDB
- start mongoldb
  - in Windows: https://www.freecodecamp.org/news/learn-mongodb-a4ce205e7739/
  - In short
    - go to C: -> Program Files -> MongoDB -> Server -> 4.0(version) -> bin
      - your version may be different - look for the bin directory
    - C:\> mkdir data/db
    - C:\> cd data
    - C:\> mkdir db
    - C:> mongod

The University Of Sheffield.

It should run on port 27017



on a Mac

```
administrator@Saturn Downloads % tar -xvzf mongodb-macos-x86_64-4.4.4.tgz
x mongodb-macos-x86_64-4.4.4/LICENSE-Community.txt
x mongodb-macos-x86_64-4.4.4/MPL-2
x mongodb-macos-x86_64-4.4.4/README
x mongodb-macos-x86_64-4.4.4/THIRD-PARTY-NOTICES
x mongodb-macos-x86_64-4.4.4/bin/install_compass
x mongodb-macos-x86_64-4.4.4/bin/mongo
x mongodb-macos-x86_64-4.4.4/bin/mongod
x mongodb-macos-x86_64-4.4.4/bin/mongos
administrator@Saturn Downloads %
```

Move the created folder into a suitable place, e.g. your home directory or somewhere under /lib

on a Mac (2)

- mkdir ~/data/db
- sudo <*path to your mongo instance*>/bin/mongod --dbpath ~/data/db --port 27017

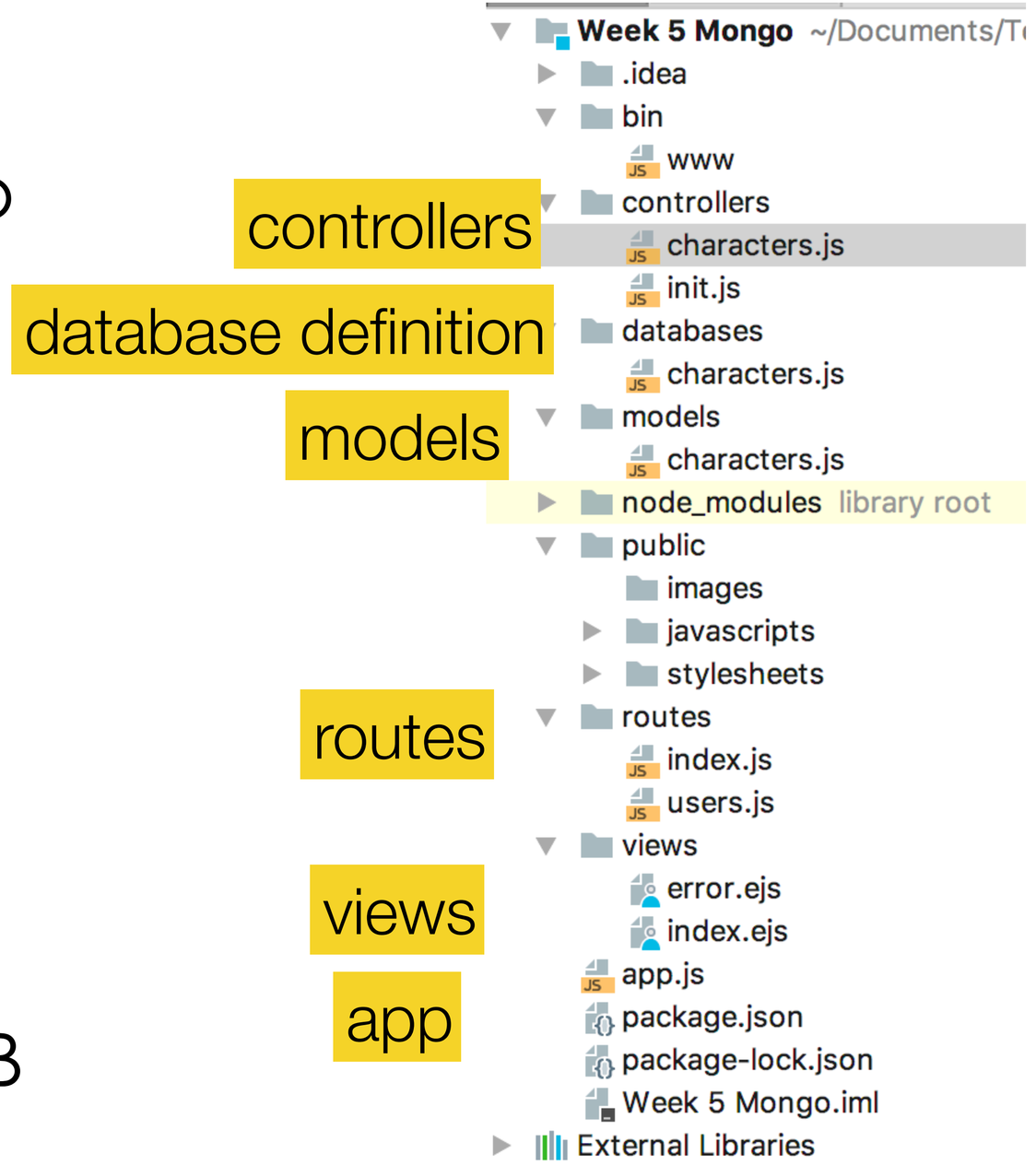
- if you want to keep it running:

- sudo <*path to your mongo instance*>/bin/mongod --dbpath /data/db --port 27017 —fork --logpath mongolog.log

Finally…

Revision

- Understand how MongoDb works
- Being able to create a small nodeJS program querying a networked Mongo DB
- Have a clear understanding of the required organisation of the code for the Mongo DB in a nodes environment



Questions?