Web Programming

Lecture 8 – Server-side Programming

Shamsa Abid

So far

- Client side development
  - The code we wrote so far is executed on the browser
  - Hosted on Github server
  - We request our page from the server and our browser processes and renders the page

Moving on to

- Server side development

Introduction

- MEAN
  - a free and open-source JavaScript software stack for building dynamic web sites and web applications.

What do you need to learn to be a MEAN developer?

JS

JavaScript

I am already using it on Front End, What will I use on Backend?

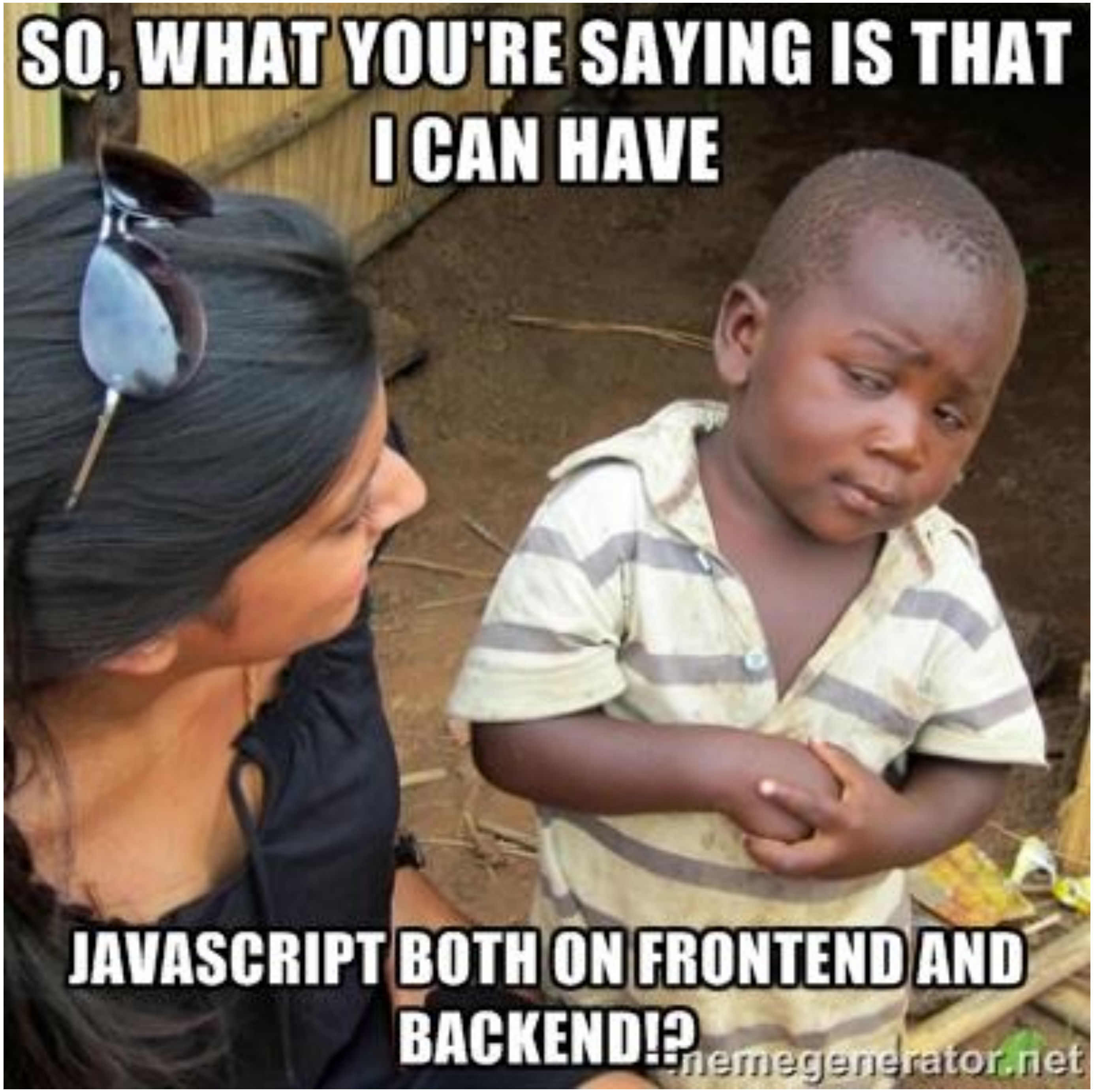
JS

JavaScript

OK. And Which language I’ll use to QUERY my database?

JS

JavaScript


SO, WHAT YOU'RE SAYING IS THAT I CAN HAVE
JAVASCRIPT BOTH ON FRONTEND AND BACKEND!?
memegenerator.net

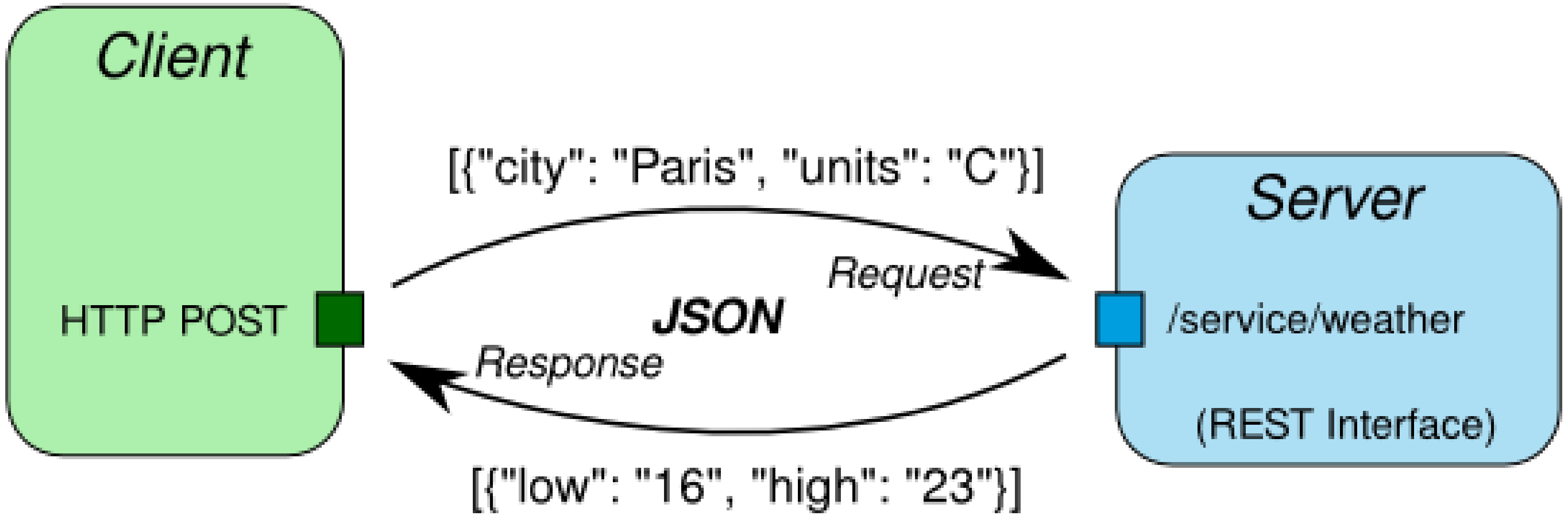
YES!

What is JSON?

(JavaScript Serialized Object Notation)

```
{
  "arguments" : { "number" : 10 },
  "url" : "http://localhost:8080/restty-tester/collection",
  "method" : "POST",
  "header" : {
    "Content-Type" : "application/json"
  },
  "body" : [
    {
      "id" : 0,
      "name" : "name 0",
      "description" : "description 0"
    },
    {
      "id" : 1,
      "name" : "name 1",
      "description" : "description 1"
    }
  ],
  "output" : "json"
}
```

JSON / REST / HTTP



MEAN Stands for?

- **MongoDB** is a NoSQL Database. It’s designed for storing non-relational data.
- **Node** is a runtime environment for server-side applications.
- **Express** is a lightweight Node.js web application framework.
- **AngularJS** is the front end, UI and client side logic of the stack

nodeJS

NodeJS

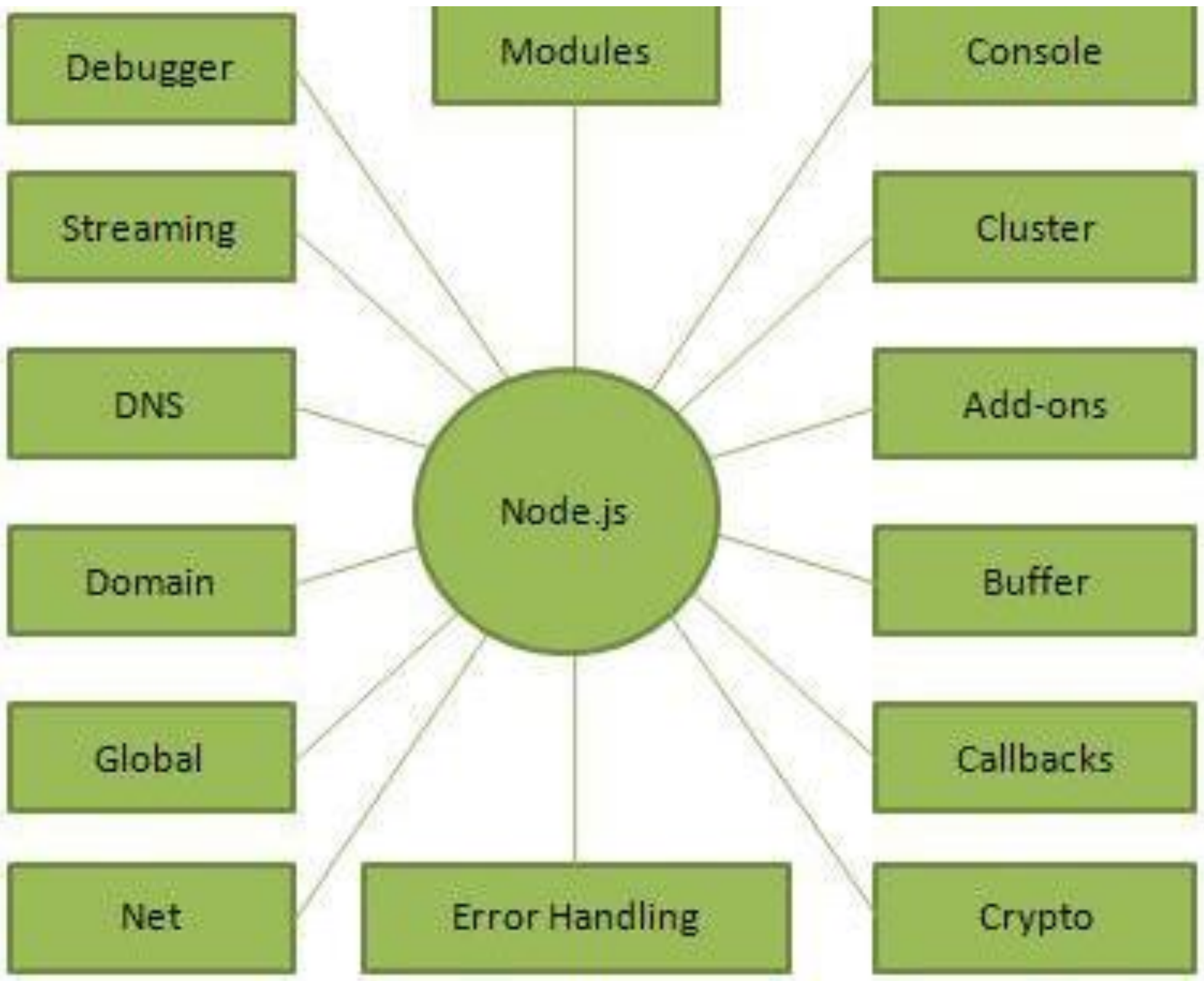
- Node.js is a platform built on Chrome's V8 JavaScript Engine for easily building fast and scalable network applications.

- Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

- It was created by Ryan Dahl starting in 2009, and its growth is sponsored by Joyent, his employer.

What can you do with Node ?

- Node is a platform for writing JavaScript applications outside web browsers.
- There is no DOM built into Node, nor any other browser capability.
- Node can’t run on GUI, but run on terminal


Debugger
Modules
Console
Streaming
Cluster
DNS
Add-ons
Node.js
Domain
Buffer
Global
Callbacks
Net
Error Handling
Crypto

Callbacks

- A callback function is called at the completion of a given task.
- Node makes heavy use of callbacks.
- All the APIs of Node are written in such a way that they support callbacks.

Blocking Code Example

Create a text file named **input.txt** with the following content

```
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

Create a js file named **main.js** with the following code –

```
var fs = require("fs");

var data = fs.readFileSync('input.txt');

console.log(data.toString());
console.log("Program Ended");
```

Now run the main.js to see the result –

```
$ node main.js
```

Verify the Output.

```
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
Program Ended
```

Non-Blocking Code Example

Update main.js to have the following code –

```
var fs = require("fs");

fs.readFile('input.txt', function (err, data) {
   if (err) return console.error(err);
   console.log(data.toString());
});

console.log("Program Ended");
```

Now run the main.js to see the result –

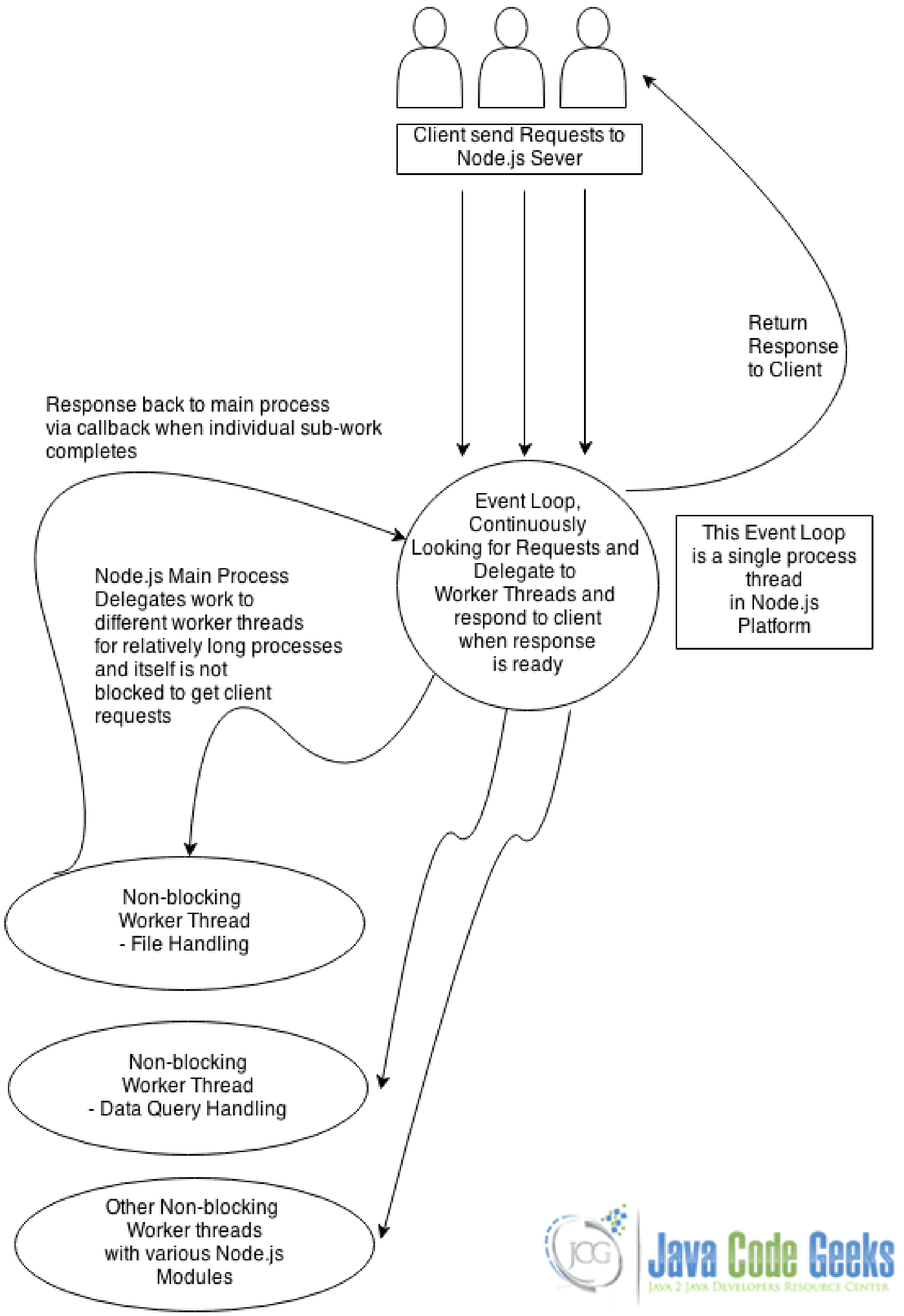
```
$ node main.js
```

Verify the Output.

```
Program Ended

Tutorials Point is giving self learning content

to teach the world in simple and easy way!!!!!
```


Client send Requests to Node.js Sever
Return Response to Client
Response back to main process via callback when individual sub-work completes
Event Loop, Continuously Looking for Requests and Delegate to Worker Threads and respond to client when response is ready
This Event Loop is a single process thread in Node.js Platform
Node.js Main Process Delegates work to different worker threads for relatively long processes and itself is not blocked to get client requests
Non-blocking Worker Thread - File Handling
Non-blocking Worker Thread - Data Query Handling
Other Non-blocking Worker threads with various Node.js Modules

Who is using NodeJS?

- Some Companies
  - Netflix
  - New York Times
  - PayPal
  - LinkedIn
  - Uber
- Complete List
  - https://www.quora.com/What-companies-are-using-Node-js-in-production

Where to Use Node.js?

- I/O bound Applications
- Data Streaming Applications
- Data Intensive Real-time Applications (DIRT)
- JSON APIs based Applications
- Single Page Applications

INTRODUCING FULL-STACK DEVELOPMENT

Full stack development

- Developing all parts of a website or application
- Full stack starts with the database and web server in the backend, contains application logic and control in the middle, and goes all the way through to the user interface at the front end
- MongoDB- the database
- Express- the web framework
- AngularJS- the front-end framework
- Node.js- the web server

Node.js

- A platform that allows you to create your own web server and build web applications on top of it
- Its not a web server or a language
- It contains a built-in HTTP server library
  - You don’t need to run a separate server program like Apache or IIS

Fast, efficient and scalable

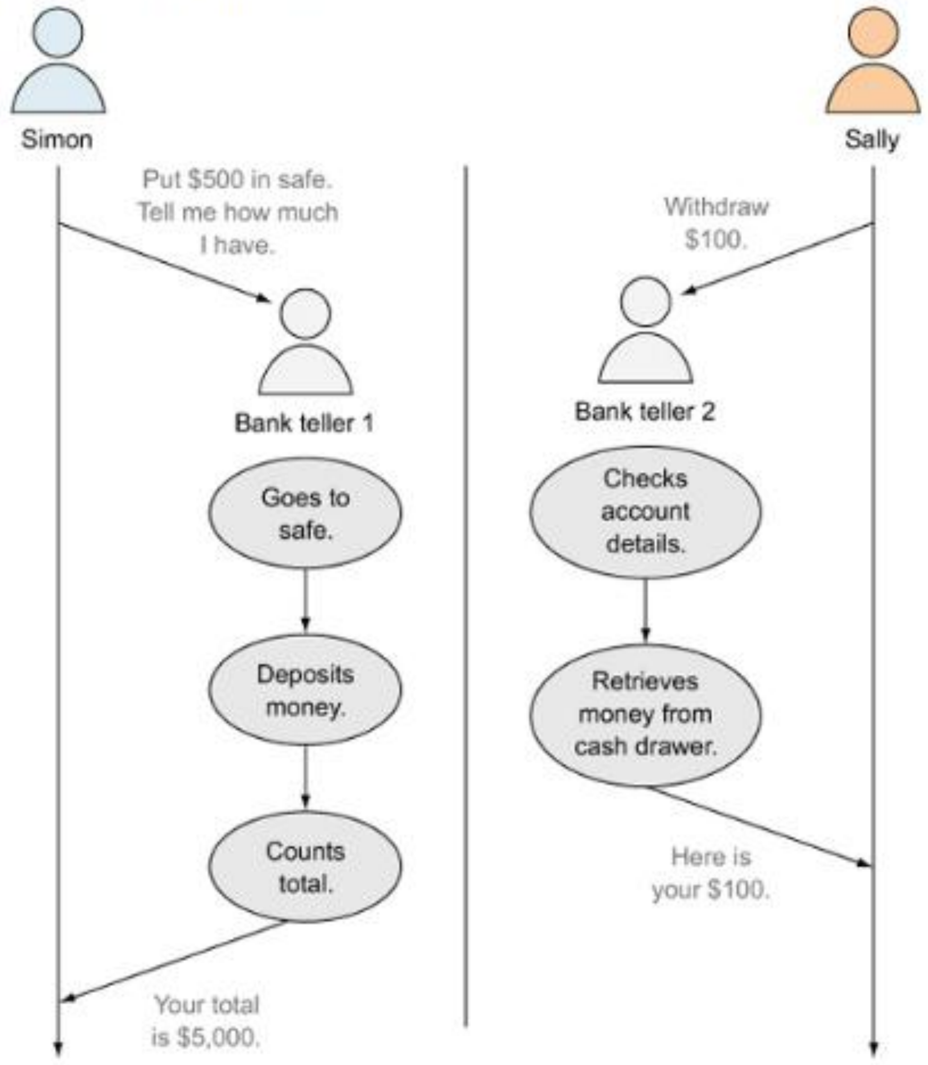
- Node.js is light on system resources because its single treaded
- It can serve more users on fewer server resources than most mainstream server technologies

Traditional multithreaded web server

- Every new visitor is given a separate thread and associated amount of RAM, often around 8MB
- One of the reasons why web servers are often overpowered and have so much RAM, even though they don’t need it 90% of the time
- The hardware is set up to be prepared for a huge spike in traffic
- Is there a better way to be more scalable?

Traditional multithreaded web server

Figure 1.3. Example of a multithreaded approach: visitors use separate resources. Visitors and their dedicated resources have no awareness of or contact with other visitors and their resources.

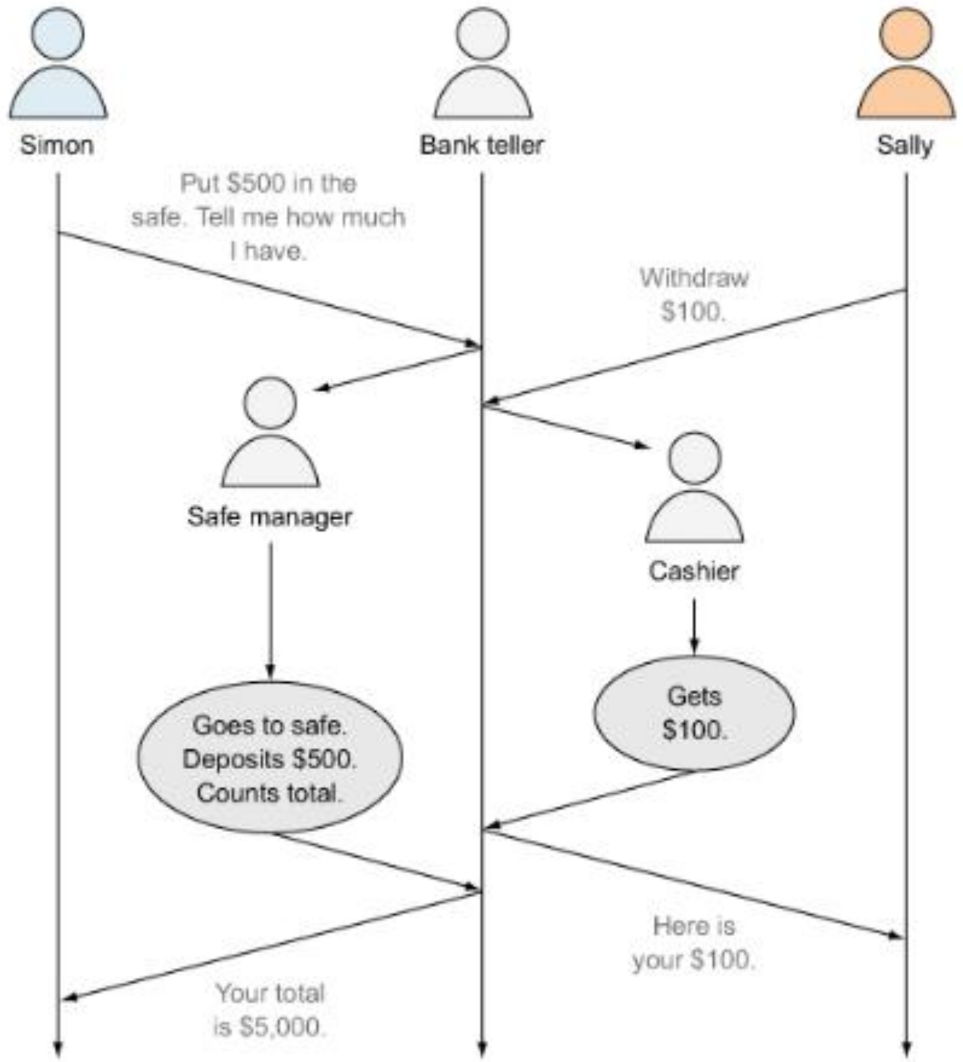


A single-threaded web server

- Node.js server is single-threaded
  - Rather than giving each visitor a unique thread and a separate set of resources, every visitor joins the same thread
  - A visitor and thread interact only when needed, when the visitor is requesting something or the thread is responding to a request

Single threaded approach

Figure 1.4. Example of a single-threaded approach: visitors use the same central resource. The central resource must be well disciplined to prevent one visitor from affecting others.



Blocking vs Nonblocking code

- Since all users use the same central process in a single-threaded model
  - Make sure nothing in your code causes delay, blocking another operation
- Blocking tasks
  - Reading static files(CSS, JavaScript, images)
  - Interacting with the database
- Make any blocking operations run asynchronously
  - Asynchronous capability of JavaScript: Callbacks!

Using pre-built packages via npm

- Npm is a package manager
  - Gets installed when u install Node.js
  - Can download Node.js modules or packages
  - 46000 plus packages available
    - Helper libraries like *Underscore*
    - Testing frameworks like *Mocha*
    - Utilities like *Colors*

Introducing Express: The framework

- Its hard to get a basic website up and running with simply Node.js
  - Express abstracts away this difficulty
    - Sets up a web server to listen to incoming requests and return responses
    - Defines a directory structure
    - One folder is set up to serve static files in a non-blocking way
- Common tasks that need doing every time
- Express does these tasks in a well-tested repeatable way

Introducing Express: The framework

- Routing URLs to responses
  - Simple interface for directing an incoming URL to a certain piece of code
    - Serve static HTML
    - Read from database
    - Write to database etc.

Introducing Express: The framework

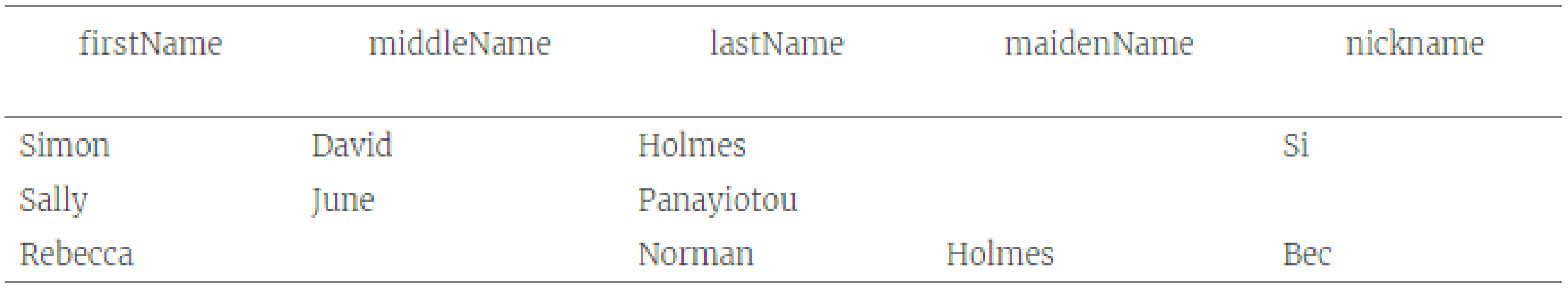
- Remembering visitors with session support
  - Node.js being single threaded, doesn’t remember a visitor from one request to the next
  - It just sees a series of HTTP requests
  - Express comes with the ability to use sessions so you can identify individual visitors through multiple requests and pages

Introducing MongoDB: The database

- MongoDB is a document database.
- The concept of rows still exists but columns are removed from the picture.
- Each row is a document, and this document both defines and holds the data itself.
- MongoDB stores data as BSON, which is binary JSON

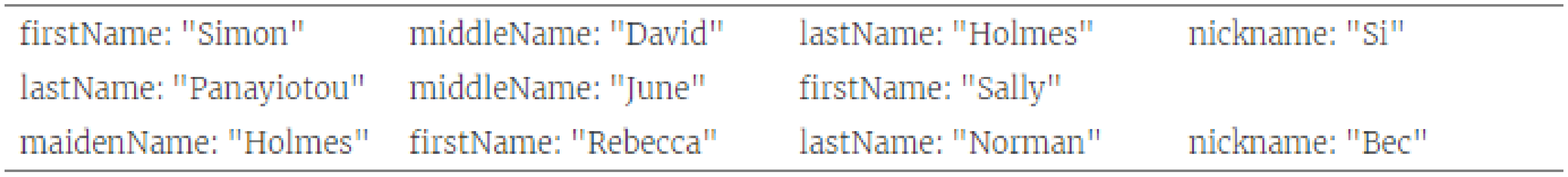
Relational versus document databases

Table 1.1. How rows and columns can look in a relational database table



firstName	middleName	lastName	maidenName	nickname
Simon	David	Holmes		Si
Sally	June	Panayiotou		
Rebecca		Norman	Holmes	Bec

Table 1.2. Each document in a document database defines and holds the data, in no particular order.



firstName: "Simon"	middleName: "David"	lastName: "Holmes"	nickname: "Si"
lastName: "Panayiotou"	middleName: "June"	firstName: "Sally"	
maidenName: "Holmes"	firstName: "Rebecca"	lastName: "Norman"	nickname: "Bec"

Simple MongoDB document

```
{
  "firstName" : "Simon",
  "lastName" : "Holmes",
  _id : ObjectId("52279effc62ca8b0c1000007")
}
```

- A document holds name and value pairs
- _id entity is a unique identifier that MongoDB assigns to any new document when it is created

MongoDB

- Support for secondary indexing and rich queries.
  - This means that you can create indexes on more than just the unique identifier field, and querying indexed fields is much faster.
  - You can also create some fairly complex queries against a MongoDB database—not to the level of huge SQL commands with joins all over the place, but powerful enough for most use cases.

MongoDB

- MongoDB is a NoSQL Database.
- It’s designed for storing non-relational data.
- It’s included in the stack because it’s queried using JSON and can persist objects as serialized JSON, making it ideal for Javascript clients.
- It’s fast, cheap and flexible, suited for simpler applications.

Mongoose for data modelling

- Data modeling, in the context of Mongoose and MongoDB, is defining what data *can* be in a document, and what data *must* be in a document.
  - When storing user information you might want to be able to save the first name, last name, email address, and phone number.
  - But you only *need* the first name and email address, and the email address must be unique.
  - This information is defined in a **schema**, which is used as the basis for the data model.

Mongoose for data modeling

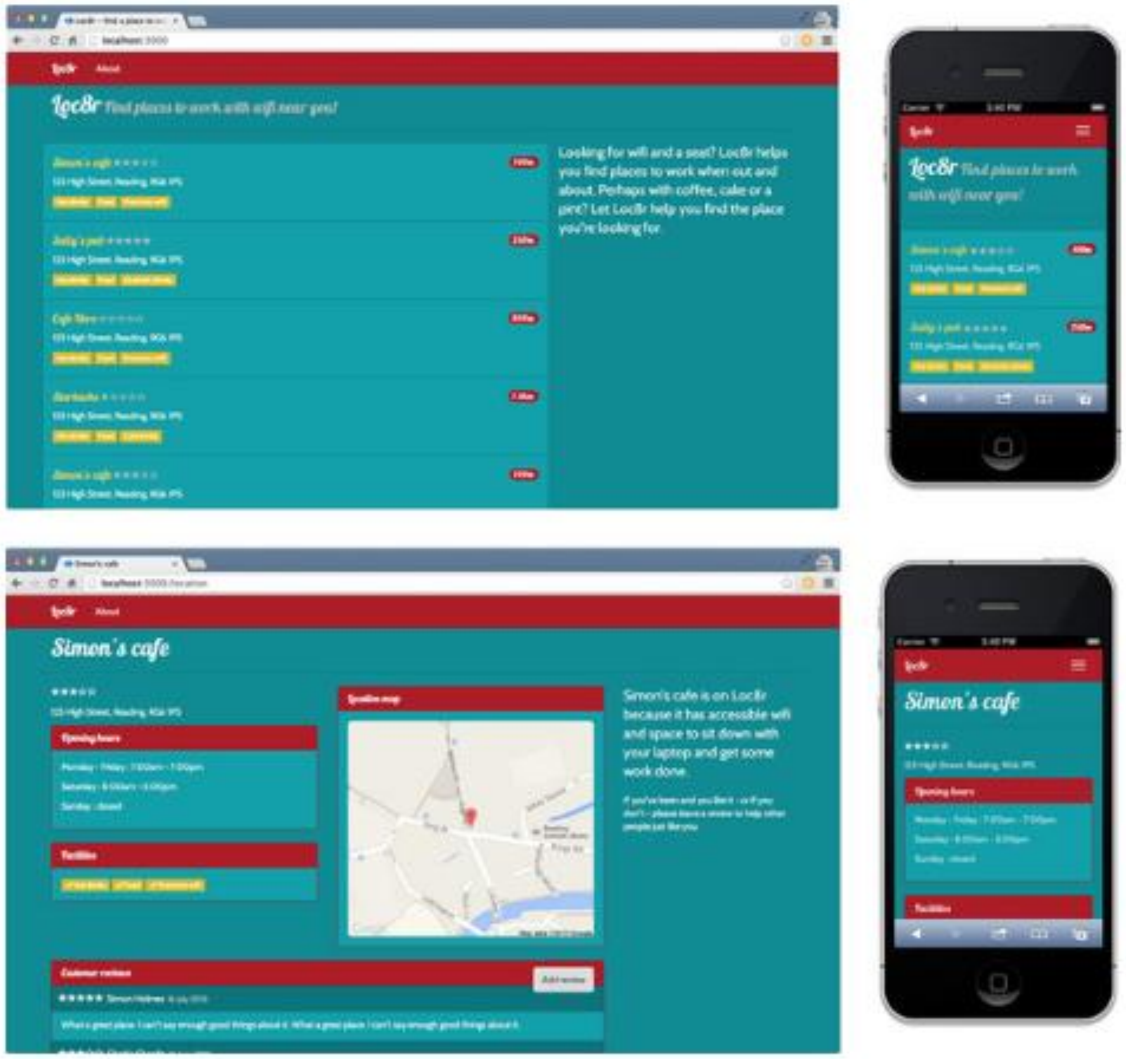
- The company behind MongoDB created Mongoose.
- In their own words, Mongoose provides “elegant MongoDB object modeling for Node.js”
- Mongoose makes it easier to manage the connections to your MongoDB database, as well as to save data and read data.
- Mongoose enables you to add data validation at the schema level, making sure that you only allow valid data to be saved in the database.
- MongoDB is a great choice of database for most web applications because it provides a balance between the speed of pure document databases and the power of relational databases.
- That the data is effectively stored in JSON makes it the perfect data store for the MEAN stack.

Using AngularJS

- Something that AngularJS has been specifically designed for is *single-page application* (SPA) functionality.
- In real terms, an SPA runs everything inside the browser and never does a full page reload.
- What this means is that all application logic, data processing, user flow, and template delivery can be managed in the browser.
- Think Gmail. That's an SPA. Different views get shown in the page, along with a whole variety of data sets, but the page itself never fully reloads.
- This approach can really reduce the amount of resources you need on your server, as you're essentially crowd-sourcing the computational power. Each person's browser is doing the hard work, and your server is basically just serving up static files and data on request.
- The user experience can also be better when using this approach. Once the application is loaded there are fewer calls to be made to the server, reducing the potential of latency.

Loc8r: Example Application

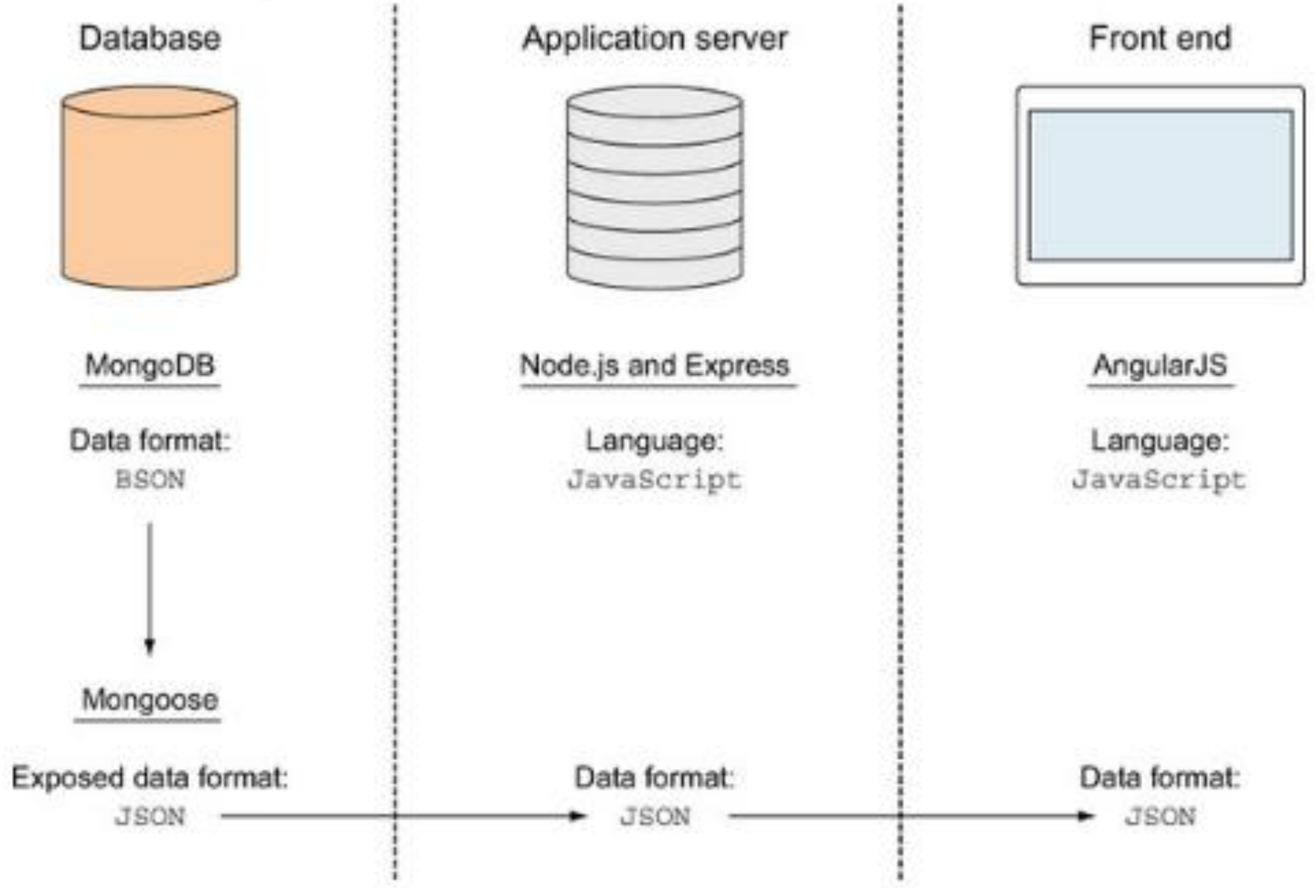
Figure 1.7. Loc8r is the application we're going to build throughout this book. It will display differently on different devices, showing a list of places and details about each place, and will allow visitors to log in and leave reviews.



Loc8r Find places to work with wifi near you!
Looking for wifi and a seat? Loc8r helps you find places to work when out and about. Perhaps with coffee, cake or a pint? Let Loc8r help you find the place you're looking for.
Simon's cafe
Simon's cafe is on Loc8r because it has accessible wifi and space to sit down with your laptop and get some work done.

How the MEAN stack components work together

Figure 1.8. JavaScript is the common language throughout the MEAN stack, and JSON is the common data format.



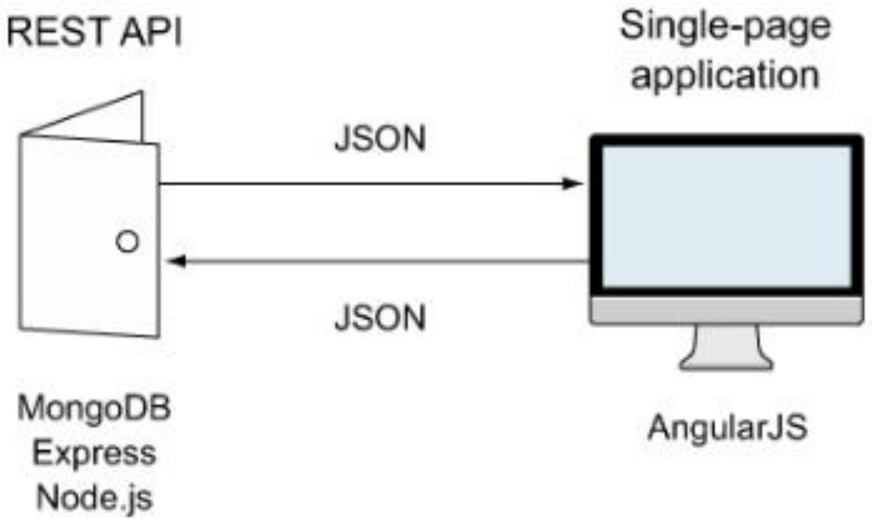
Summary

- The different technologies making up the MEAN stack
- MongoDB as the database layer
- Node.js and Express working together to provide an application server layer
- AngularJS providing an amazing front-end, data-binding layer
- How the MEAN components work together
- A few ways to extend the MEAN stack with additional technologies

DESIGNING A MEAN STACK ARCHITECTURE

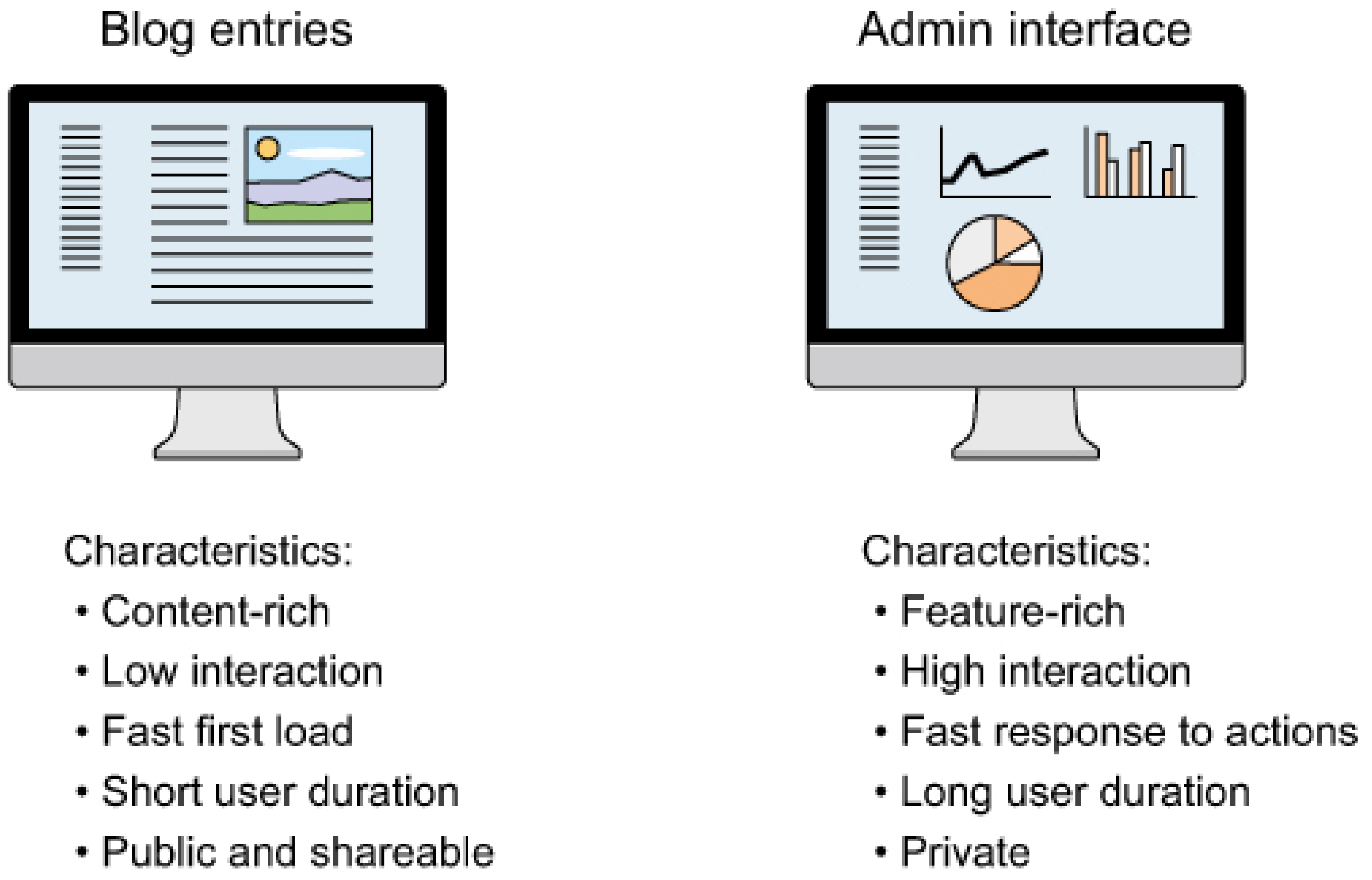
A common MEAN stack architecture

Figure 2.1. A common approach to MEAN stack architecture, using MongoDB, Express, and Node.js to build a REST API that feeds JSON data to an AngularJS SPA run in the browser



Requirements for a blog engine

Figure 2.2. Conflicting characteristics of the two sides of a blog engine, the public-facing blog entries and the private admin interface



Characteristics:

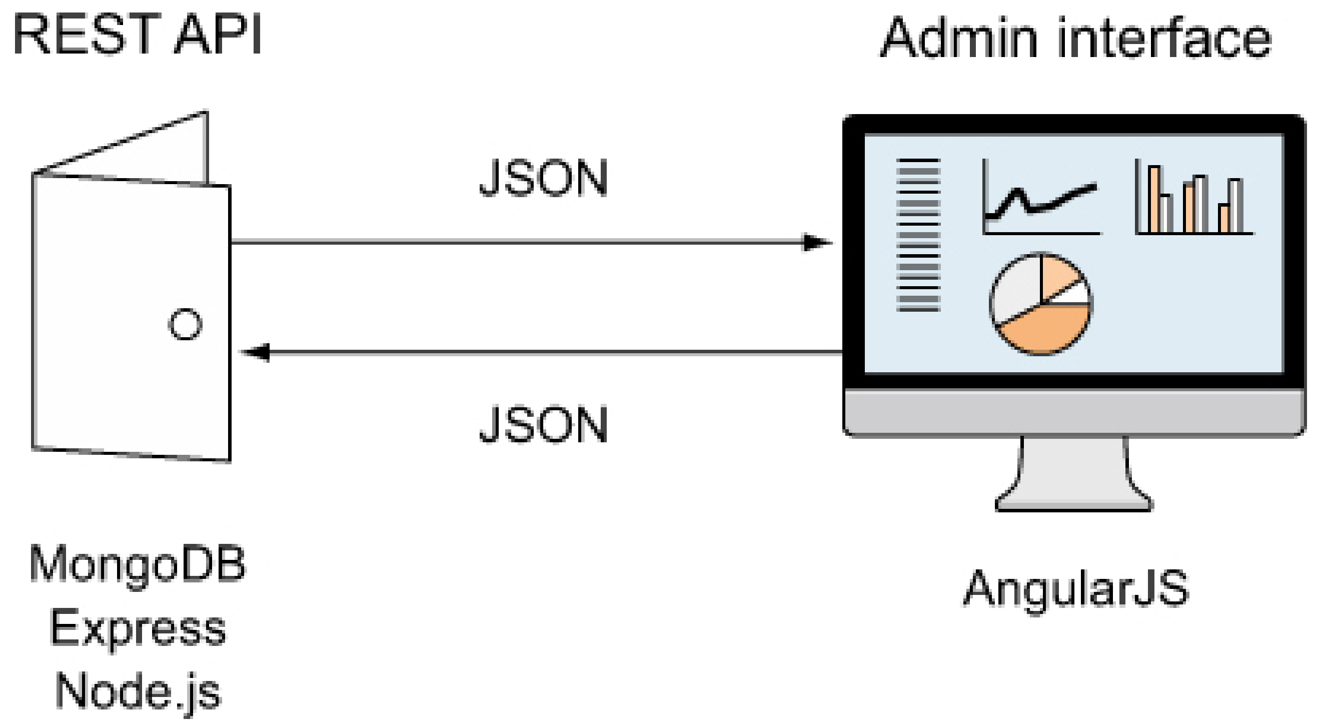
- Content-rich
- Low interaction
- Fast first load
- Short user duration
- Public and shareable

Characteristics:

- Feature-rich
- High interaction
- Fast response to actions
- Long user duration
- Private

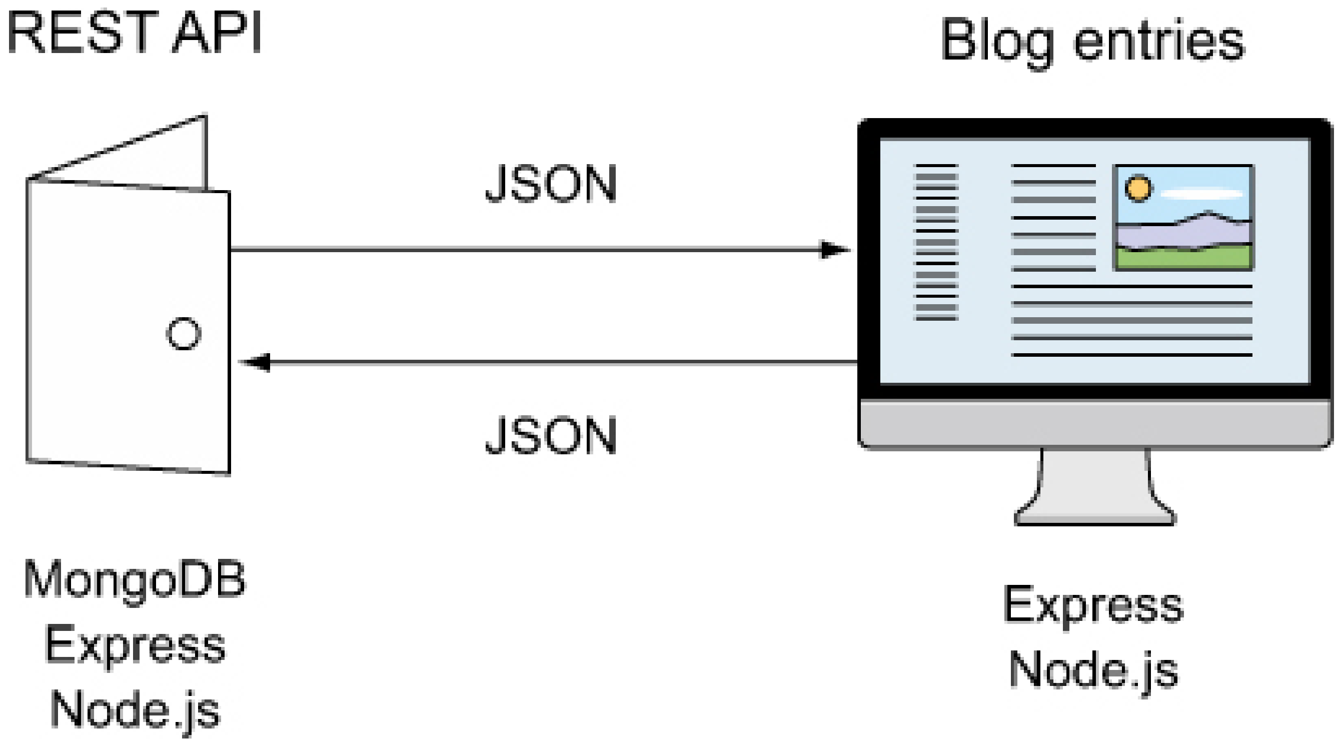
Admin interface: An Angular JS SPA

Figure 2.3. A familiar sight: the admin interface would be an AngularJS SPA making use of a REST API built with MongoDB, Express, and Node.js



Blog entries: Using express to deliver HTML

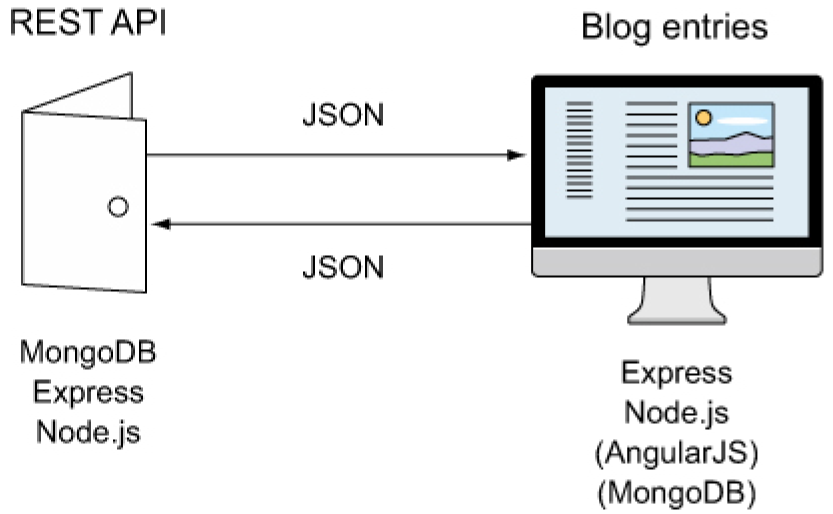
Figure 2.4. An architecture for delivering HTML directly from the server: an Express and Node.js application at the front, interacting with a REST API built in MongoDB, Express, and Node.js



Blog entries: Using more of the stack

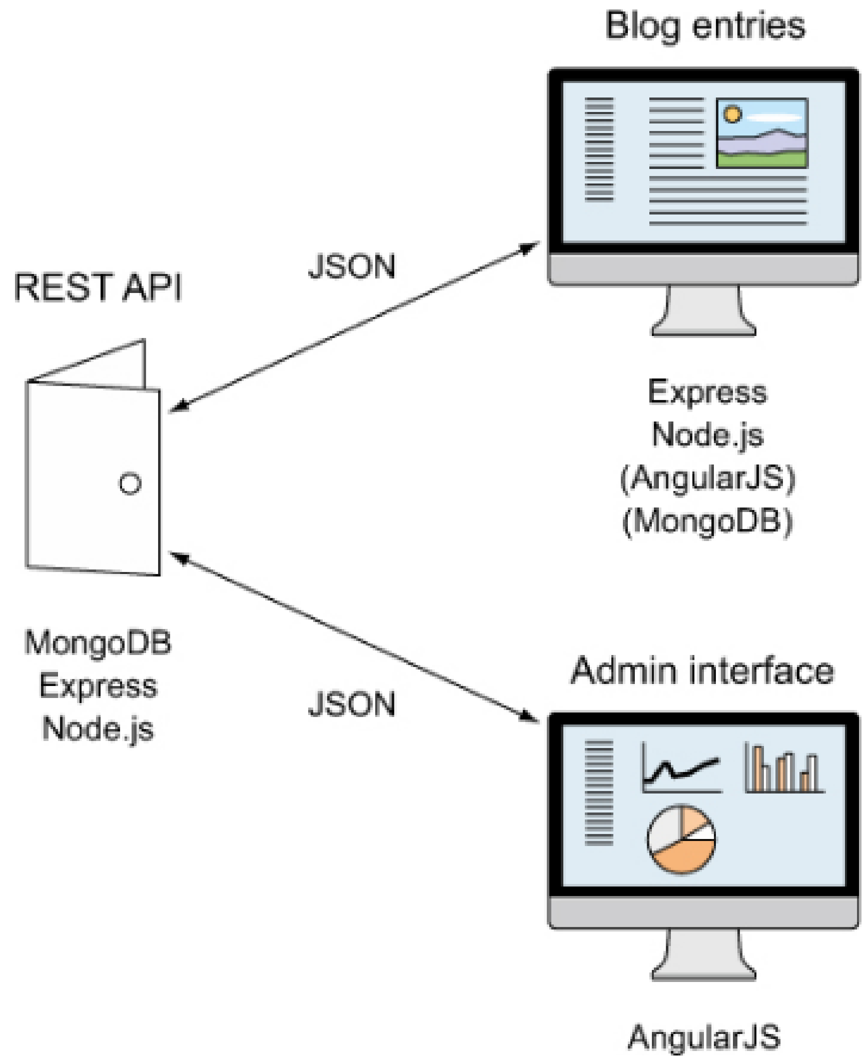
- You want visitors to be able to log in, add comments
- You need to track sessions
  - Use MongoDB with Express
- You want dynamic data such as related posts and search box with type-ahead completions
  - Use AngularJS to implement these

Figure 2.5. Adding the options of using AngularJS and MongoDB as part of the public-facing aspect of the blog engine, serving the blog entries to visitors



Blog engine: A hybrid architecture

Figure 2.6. A hybrid MEAN stack architecture: a single REST API feeding two separate user-facing applications, built using different parts of the MEAN stack to give the most appropriate solution

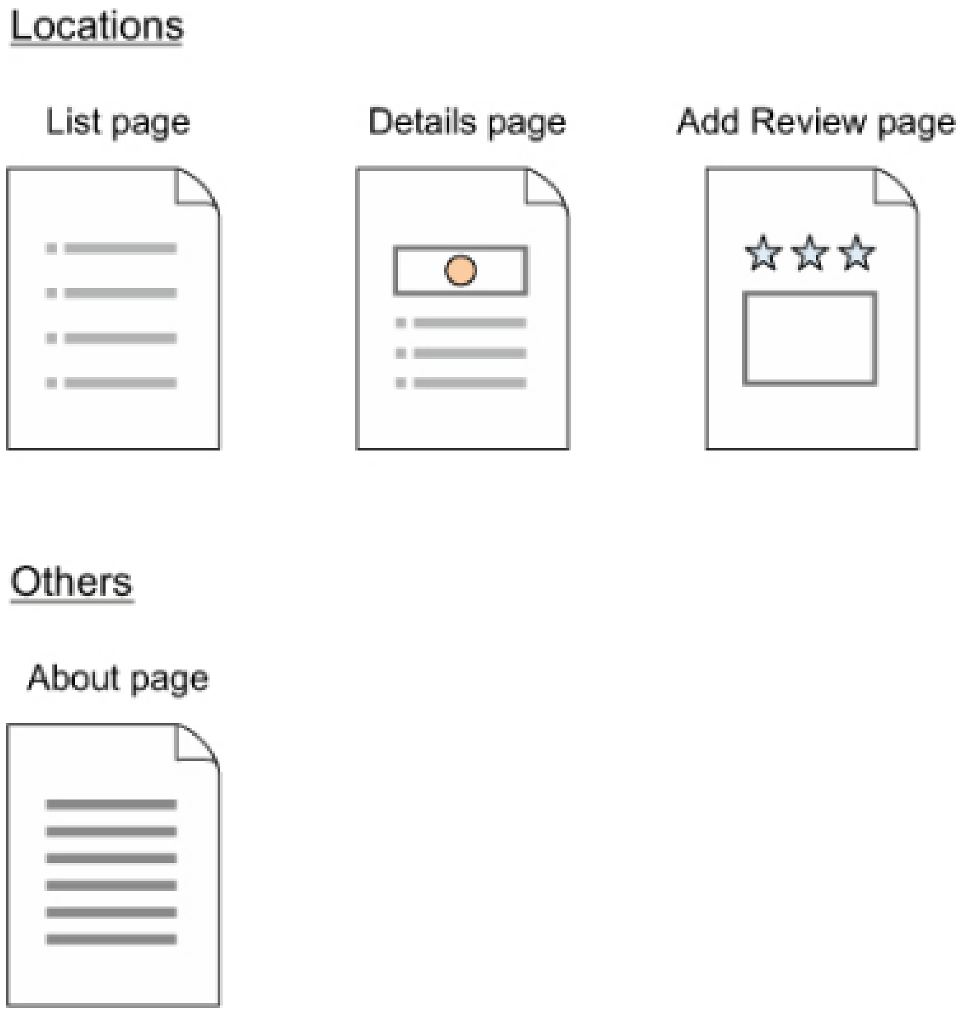


Planning a real application

- Loc8r
- Planning the app at a high level
  - Screens

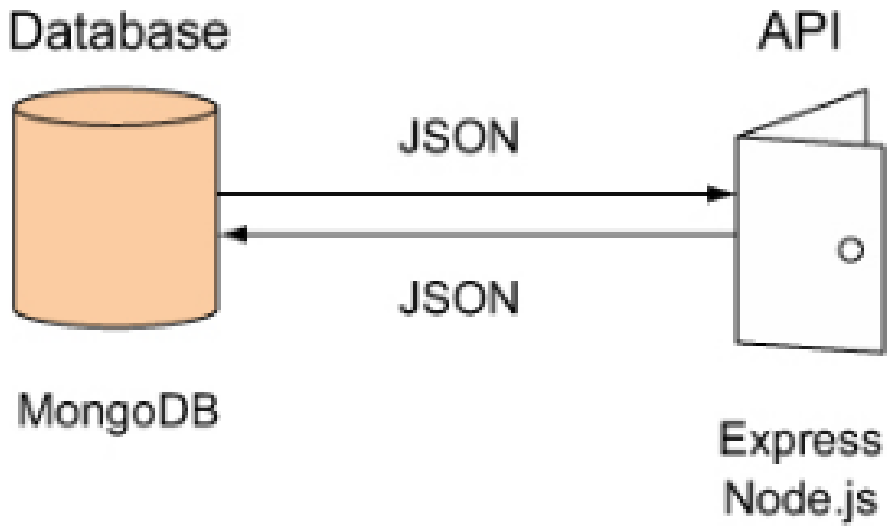
Planning a real application

Figure 2.9. Collate the separate screens for our application into logical collections.



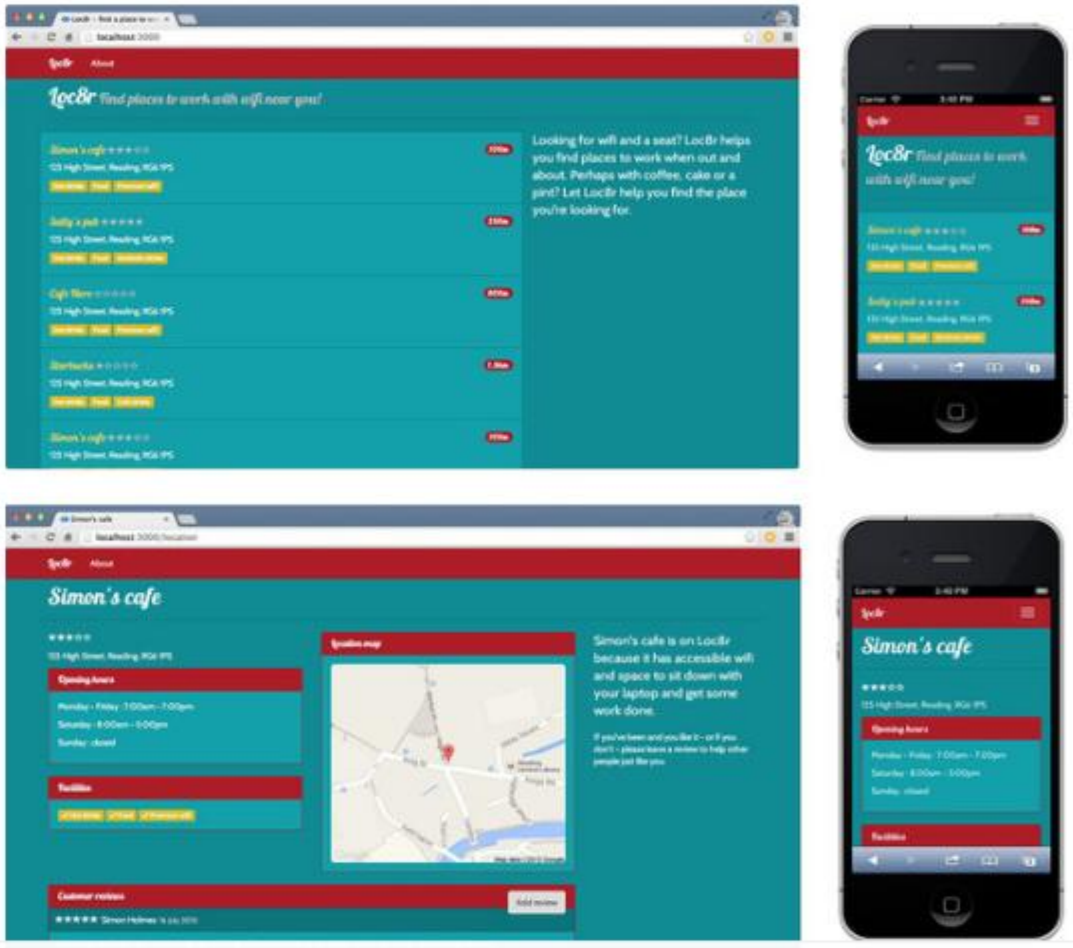
Architecting the application

Figure 2.10. Start with the standard MEAN REST API, using MongoDB, Express, and Node.js.



End product

Figure 2.13. Loc8r is the application we're going to build throughout this book. It will display differently on different devices, showing a list of places and details about each place, and will enable visitors to log in and leave reviews.



Looking for wifi and a seat? Loc8r helps you find places to work when out and about. Perhaps with coffee, cake or a pint? Let Loc8r help you find the place you're looking for.

125 High Street, Reading, RG6 1PS

Monday - Friday : 7:00am - 7:00pm
Saturday : 8:00am - 5:00pm
Sunday : closed

Simon's cafe is on Loc8r because it has accessible wifi and space to sit down with your laptop and get some work done.

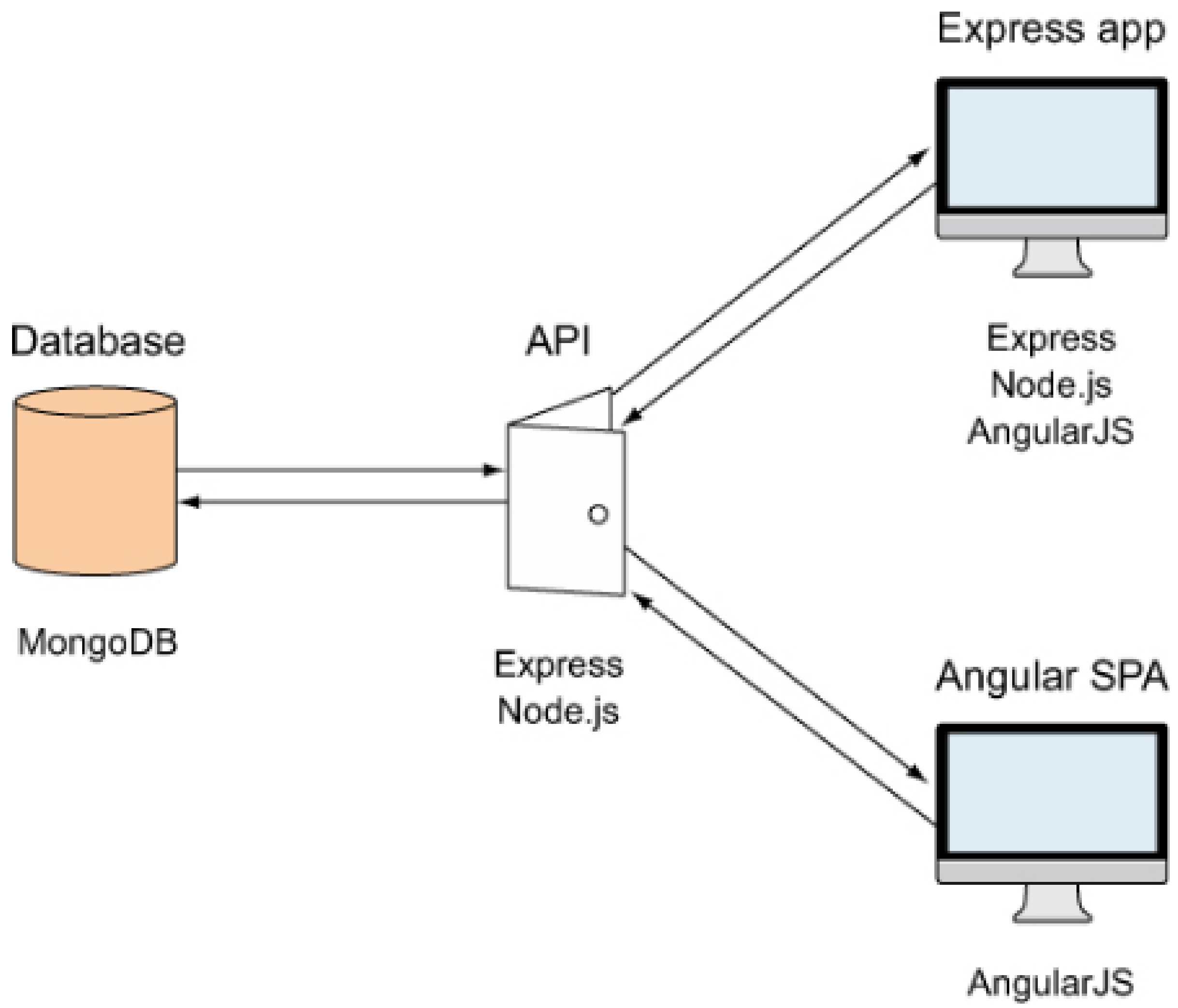
If you've been and you like it - or if you don't - please leave a review to help other people just like you.

Rapid Prototype dev stages

- Stage 1: Build a static site
  - Aims
    - Quickly figure out the layout
    - Ensure that the user flow makes sense
- Stage 2: Design the data model and create the database
- Stage 3: Build our data API
  - Create a REST API that will allow our application to interact with the database
- Stage 4: Hook the database into the application
  - Get our application to talk to our API to get a data-driven app
- Stage 5: Augment
  - Add additional functionality
    - Validation, authentication, etc

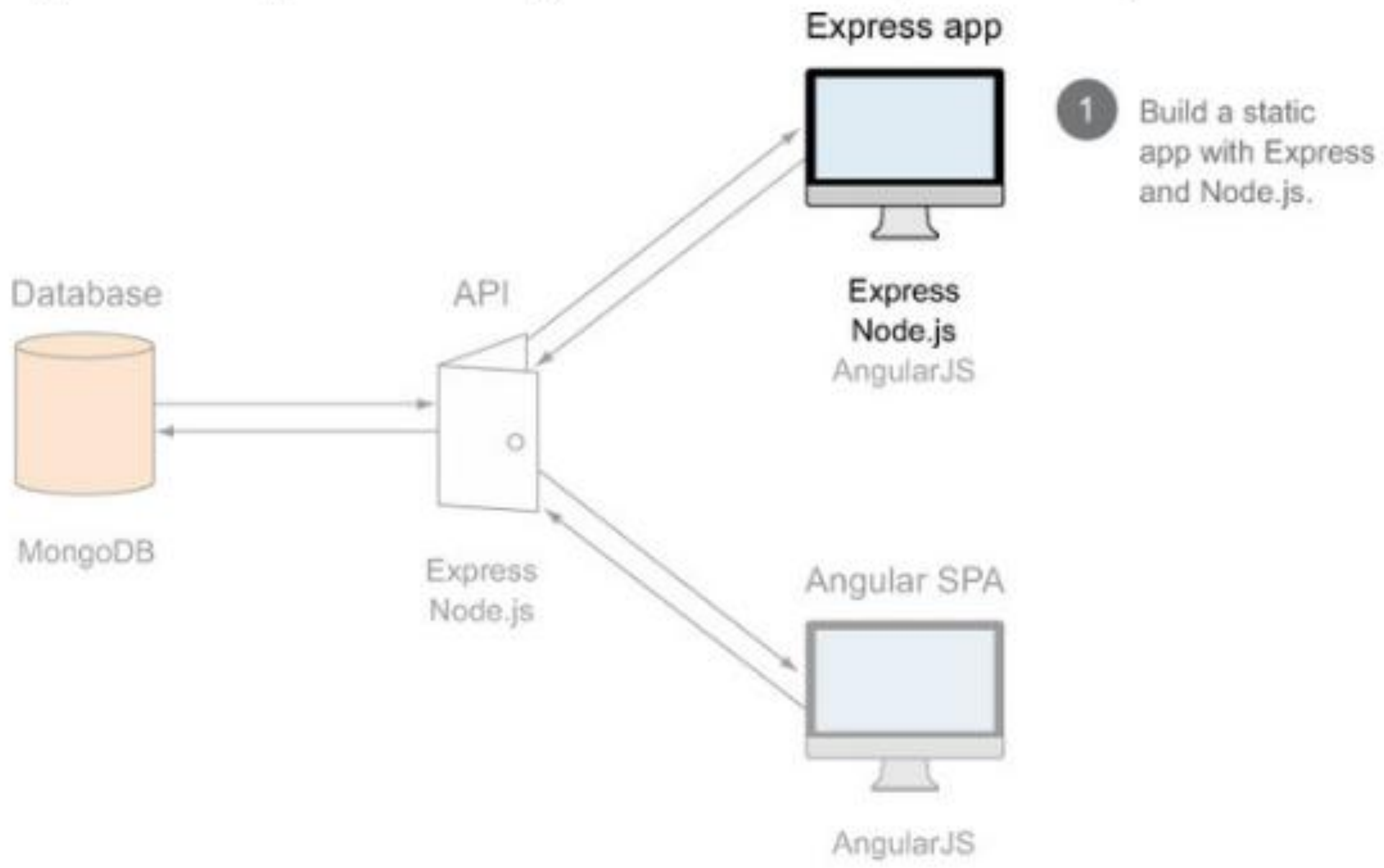
Steps to build Loc8r

Figure 2.14. Proposed architecture for Loc8r as we'll build it throughout this book



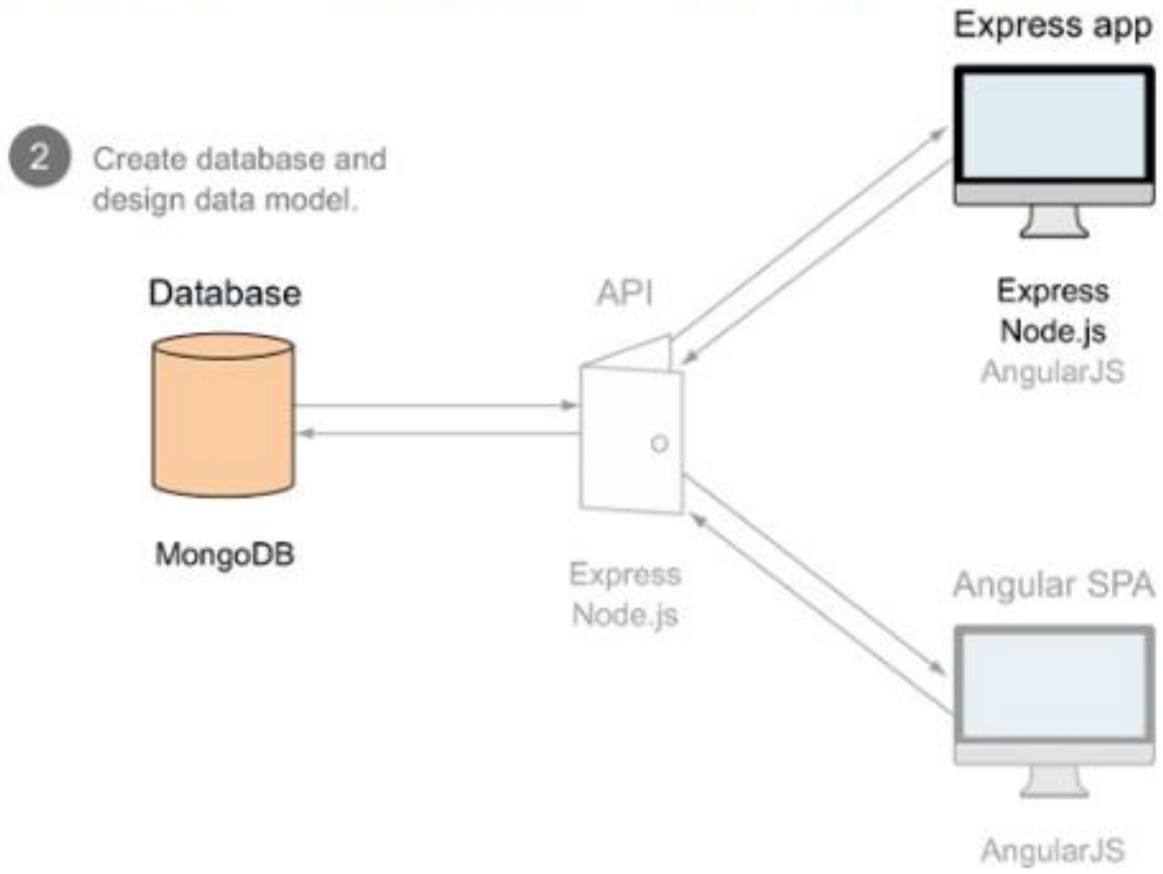
Stage 1

Figure 2.15. The starting point for our application is building the user interface in Express and Node.js.



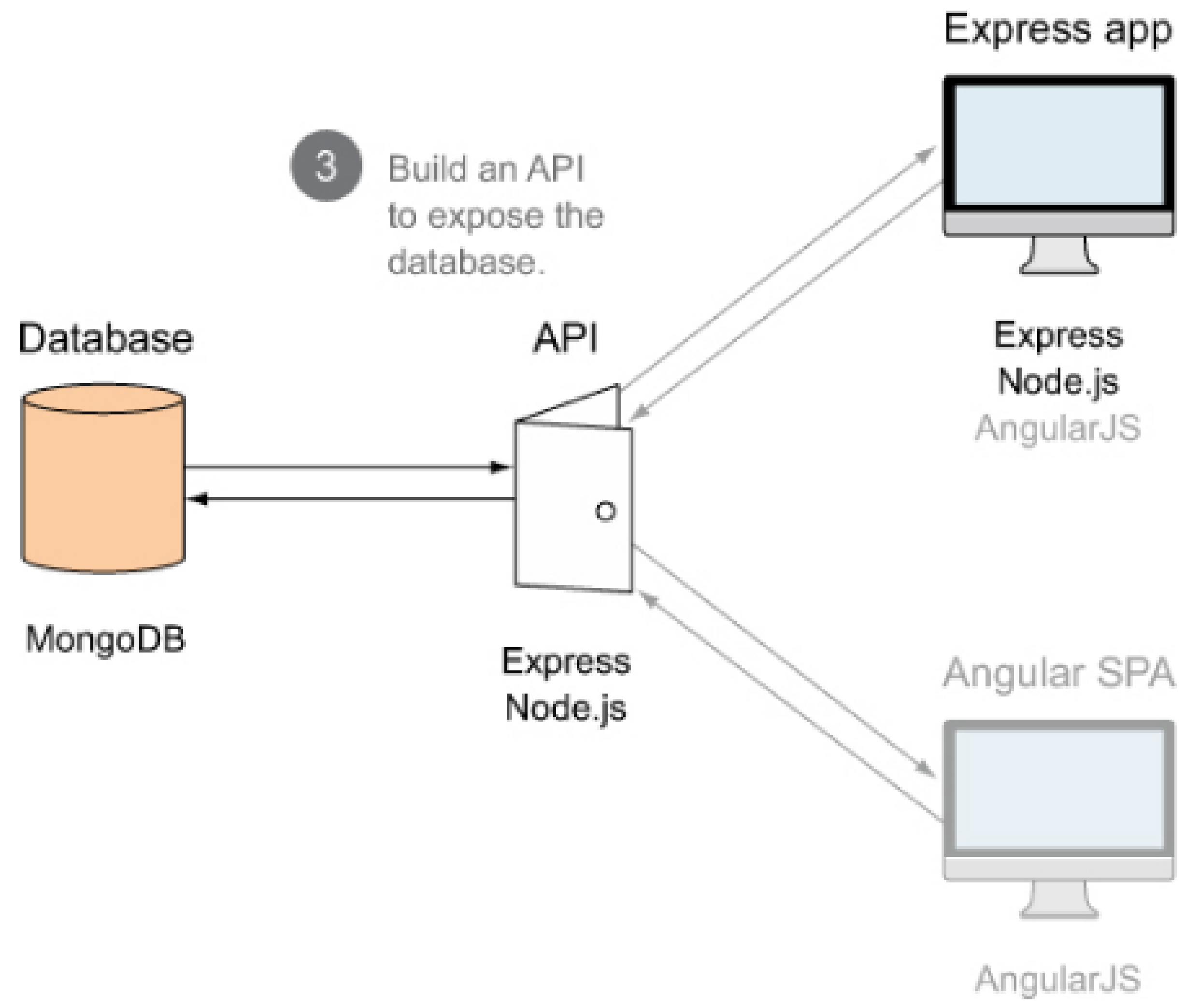
Stage 2

Figure 2.16. After the static site is built we'll use the information gleaned to design the data model and create the MongoDB database.



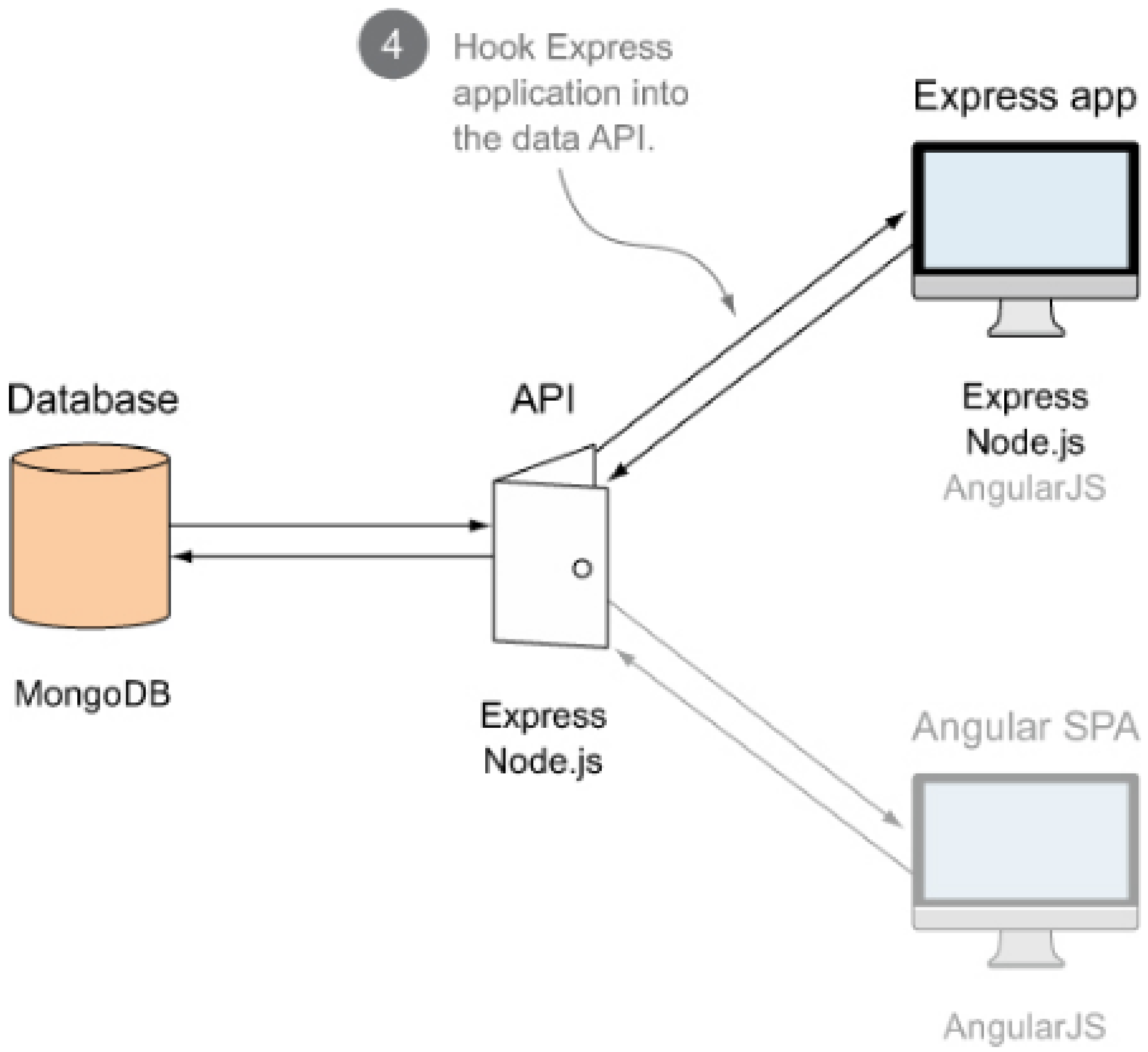
Stage 3

Figure 2.17. Use Express and Node.js to build an API, exposing methods of interacting with the database.



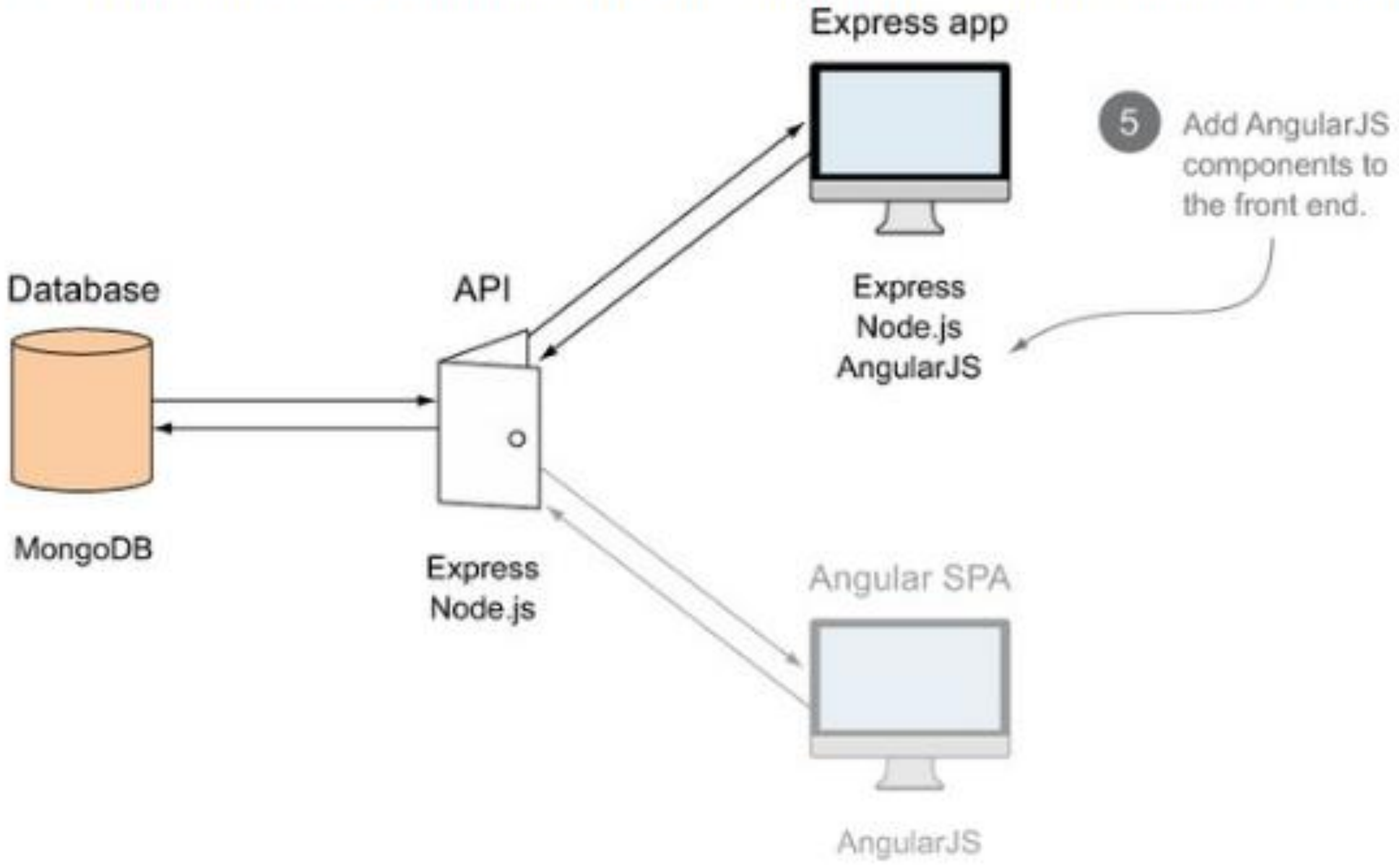
Stage 4

Figure 2.18. Update the static Express application by hooking it into the data API, allowing the application to be database-driven.



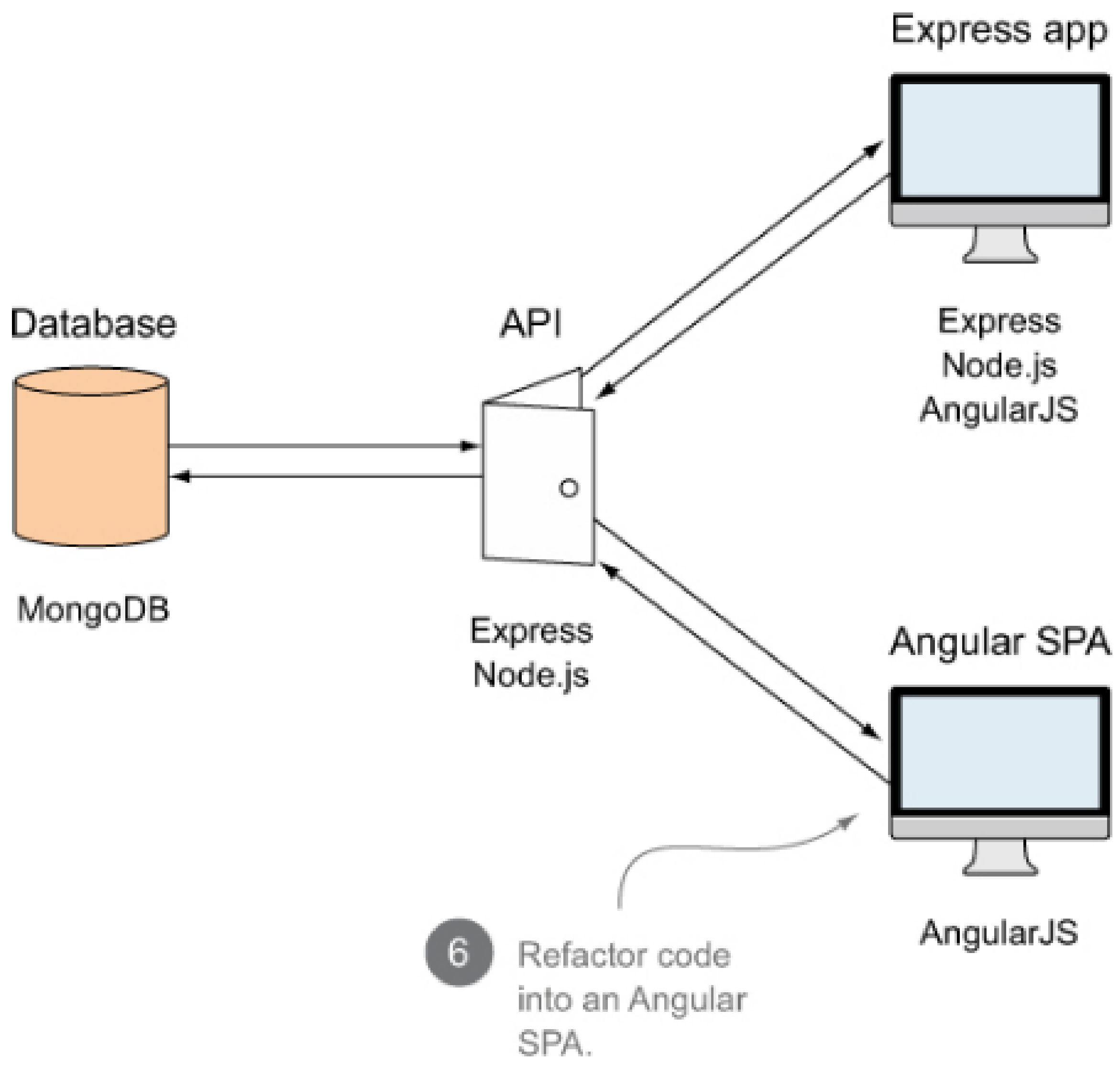
Stage 5

Figure 2.19. One way to use AngularJS in a MEAN application is to add components to the front end in an Express application.



Stage 6

Figure 2.20. Effectively rewriting the application as an AngularJS SPA



Stage 7

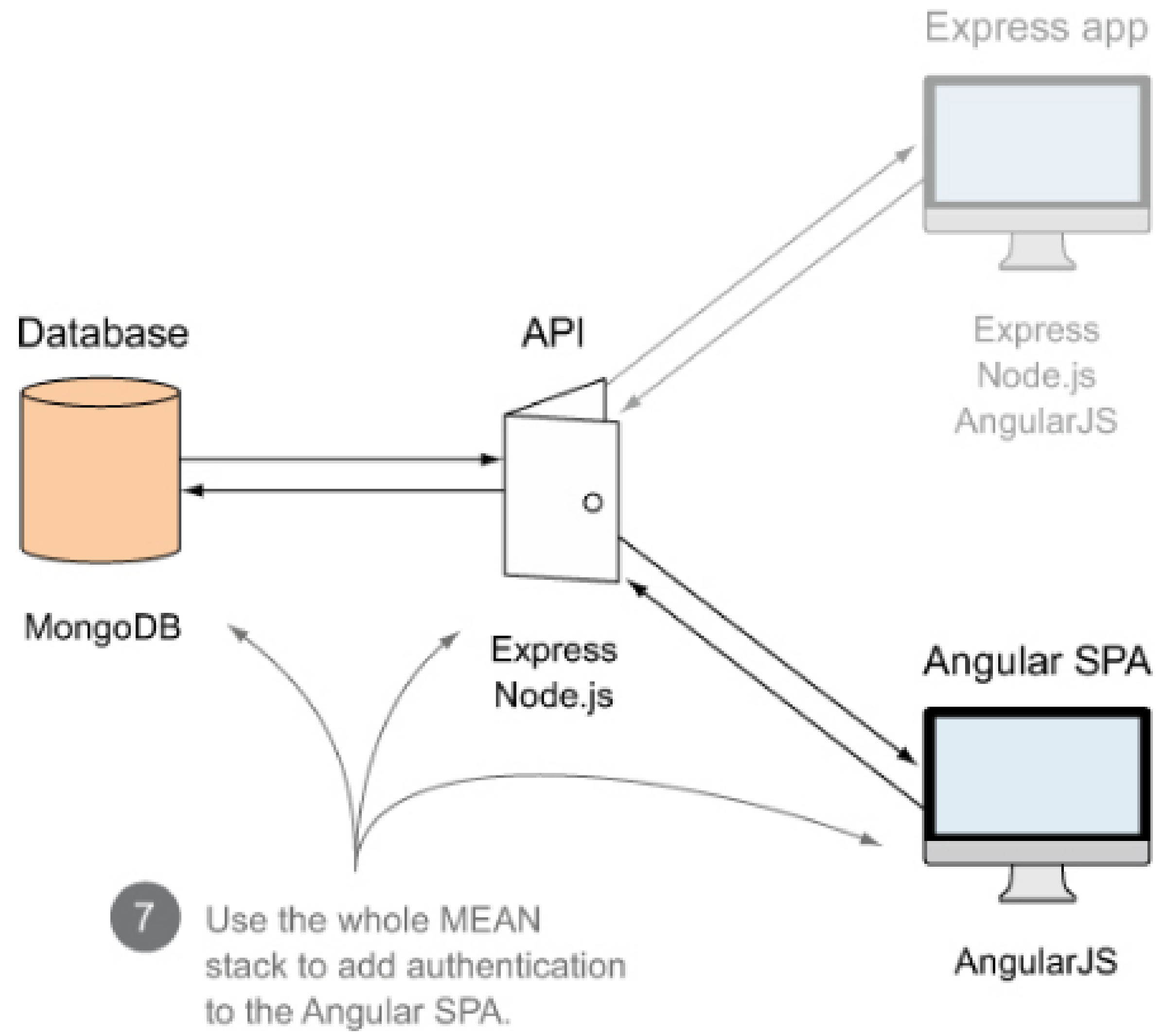


Figure 2.21. Using all of the MEAN stack to add authentication to the AngularJS SPA

Summary

- A common MEAN stack architecture with an AngularJS SPA using a REST API built in Node.js, Express, and Mongo
- Points to consider when deciding whether to build an SPA or not
- How to design a flexible architecture in the MEAN stack
- The best practice of building an API to expose a data layer
- The steps we’re going to take to build the sample application Loc8r
- Development and production hardware architectures

Installations

- DIY

Project: Deadline Wednesday 26th Sep

- Make groups of 3 to 5 members
- Choose a project to work on
  - Daraz.pk
  - LinkedIn
  - Tripadvisor
  - Facebook
  - Twitter
  - Your own custom app, then provide details
- Submit group member names and project title
  - https://goo.gl/hXj2B3

Hands On

- Build your own server
- Include modules
- Create and include your own modules
- Http module
  - allows Node.js to transfer data over HTTP
- File system module
- url module
  - splits up a web address into readable parts
- Npm
- Events
  - events module, EventEmitter object and emit method
- Upload files
  - Formidable module

Event driven programming example

- Although events look quite similar to callbacks, the difference lies in the fact that callback functions are called when an asynchronous function returns its result, whereas event handling works on the observer pattern.
- The functions that listen to events act as **Observers**.
- Whenever an event gets fired, its listener function starts executing.
- Node.js has multiple in-built events available through events module and EventEmitter class which are used to bind events and event-listeners

Event driven programming example

```
// Import events module
var events = require('events');


// Create an eventEmitter object
var eventEmitter = new events.EventEmitter();

// Create an event handler as follows
var connectHandler = function connected() {
   console.log('connection succesful.');

   // Fire the data_received event
   eventEmitter.emit('data_received');
}

// Bind the connection event with the handler
eventEmitter.on('connection', connectHandler);

// Bind the data_received event with the anonymous function
eventEmitter.on('data_received', function(){
   console.log('data received succesfully.');
});

// Fire the connection event
eventEmitter.emit('connection');

console.log("Program Ended.");
```

IT should produce the following result –

```
connection successful.

data received successfully.

Program Ended.
```

ExpressJS

- Express is a lightweight Node.js web application framework.
- Node provides a http module by default but it’s very low level.
- Express wraps this up to provide simple get/post methods and other routing.
- In the stack, it glues everything together… and it’s also written in JavaScript.

Installing Express

- $ npm install express --save
- The above command saves the installation locally in the **node_modules** directory and creates a directory express inside node_modules.
- You should install the following important modules along with express −
  - **body-parser**
    - This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.
  - **cookie-parser**
    - Parse Cookie header and populate req.cookies with an object keyed by the cookie names.
  - **multer**
    - This is a node.js middleware for handling multipart/form-data.
- $ npm install body-parser --save
- $ npm install cookie-parser --save
- $ npm install multer --save

Express: Request & Response

- Express application uses a callback function whose parameters are **request** and **response** objects.
- app.get('/', function (req, res) { // -- })
- Request Object
  - The request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.
- Response Object
  - The response object represents the HTTP response that an Express app sends when it gets an HTTP request.
- You can print **req** and **res** objects which provide a lot of information related to HTTP request and response including cookies, sessions, URL, etc.

Express: Basic routing

- Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

Express: Serving Static Files

- Express provides a built-in middleware **express.static** to serve static files, such as images, CSS, JavaScript, etc.
- You simply need to pass the name of the directory where you keep your static assets, to the **express.static** middleware to start serving the files directly.
- For example, if you keep your images, CSS, and JavaScript files in a directory named public, you can do this –
- app.use(express.static('public'));

REST

- REST stands for REpresentational State Transfer.
- REST is web standards based architecture and uses HTTP Protocol.
- It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.
- REST was first introduced by Roy Fielding in 2000.
- A REST Server simply provides access to resources and REST client accesses and modifies the resources using HTTP protocol.

REST

- HTTP methods
  - Following four HTTP methods are commonly used in REST based architecture.
    - **GET** - This is used to provide a read only access to a resource.
    - **PUT** - This is used to create a new resource.
    - **DELETE** - This is used to remove a resource.
    - **POST** - This is used to update a existing resource or create a new resource.

RESTful Web Services

- Web services based on REST Architecture are known as RESTful web services.
- These webservices uses HTTP methods to implement the concept of REST architecture.
- A RESTful web service usually defines a URI, Uniform Resource Identifier a service, which provides resource representation such as JSON and set of HTTP Methods.

Homework

- Read the articles
  - https://www.codecademy.com/articles/what-is-node
  - https://www.codecademy.com/articles/back-end-architecture

References

- Getting MEAN with Mongo, Express, Angular and Node by Simon Holmes
- W3schools
- tutorialspoint