

Web Programming

Lecture 15

Writing a REST API: Exposing the MongoDB
database to the Application

Part 2

Today's topics

- CRUD functions (update, delete)
- Using Express and Mongoose to interact with MongoDB
- Testing API endpoints

PUT methods: Updating data in MongoDB

- PUT methods are all about updating existing documents or subdocuments in the database, and then returning the saved data as confirmation.

Action	Method	Parameters	Example
Update a specific location	PUT	locationid	http://loc8r.com/api/locations/123
Update a specific review	PUT	locationid reviewid	http://loc8r.com/api/locations/123/reviews/abc

Using Mongoose to update a document in MongoDB

- Steps to update data:
 - Find the relevant document
 - Make some changes to the instance
 - Save the document
 - Send a JSON response
- An instance of mongoose model maps directly to a document in MongoDB.
- When your query finds the document, you get a model instance
- If you make changes to this instance and save it, Mongoose will update the original document in the database

Using the Mongoose save method

```
1 Loc
2   .findById(req.params.locationid)
3   .exec(
4     function(err, location) {
5       location.name = req.body.name;
6       location.save(function(err, location) {
7         if (err) {
8           sendJsonResponse(res, 404, err);
9         } else {
10           sendJsonResponse(res, 200, location);
11         }
12       });
13     }
14   );
15 };
```

Making changes to an existing document in MongoDB

```
1 module.exports.locationsUpdateOne = function(req, res) {
2   if (!req.params.locationid) {
3     sendJsonResponse(res, 404, {
4       "message": "Not found, locationid is required"
5     });
6     return;
7   }
8   Loc
9     .findById(req.params.locationid)
10    .select('-reviews -rating')
11    .exec(
12      function(err, location) {
13        if (!location) {
14          sendJsonResponse(res, 404, {
15            "message": "locationid not found"
16          });
17          return;
18        } else if (err) {
19          sendJsonResponse(res, 400, err);
20          return;
21        }

```

Making changes to an existing document in MongoDB

```
22     location.name = req.body.name;
23     location.address = req.body.address;
24     location.facilities = req.body.facilities.split(",");
25     location.coords = [parseFloat(req.body.lng),
26     parseFloat(req.body.lat)];
27     location.openingTimes = [{
28         days: req.body.days1,
29         opening: req.body.opening1,
30         closing: req.body.closing1,
31         closed: req.body.closed1,
32     }, {
33         days: req.body.days2,
34         opening: req.body.opening2,
35         closing: req.body.closing2,
36         closed: req.body.closed2,
37     }
38     ];
39     location.save(function(err, location) {
40         if (err) {
41             sendJsonResponse(res, 404, err);
42         } else {
43             sendJsonResponse(res, 200, location);
44         }
45     });
46 });
47 };
```

Updating an existing subdocument in MongoDB

- Steps
 - Find the relevant document
 - Find the relevant subdocument
 - Make some changes to the subdocument
 - Save the document
 - Send a JSON response

Updating a subdocument in MongoDB

```
1 module.exports.reviewsUpdateOne = function(req, res) {
2   if (!req.params.locationid || !req.params.reviewid) {
3     sendJsonResponse(res, 404, {
4       "message": "Not found, locationid and reviewid are both required"
5     });
6     return;
7   }
8   Loc
9     .findById(req.params.locationid)
10    .select('reviews')
11    .exec(
12      function(err, location) {
13        var thisReview;
14        if (!location) {
15          sendJsonResponse(res, 404, {
16            "message": "locationid not found"
17          });
18          return;
19        } else if (err) {
20          sendJsonResponse(res, 400, err);
21          return;
22        }
```

Updating a subdocument in MongoDB

```
23     if (location.reviews && location.reviews.length > 0) {
24         thisReview = location.reviews.id(req.params.reviewid);
25         if (!thisReview) {
26             sendJsonResponse(res, 404, {
27                 "message": "reviewid not found"
28             });
29         } else {
30             thisReview.author = req.body.author;
31             thisReview.rating = req.body.rating;
32             thisReview.reviewText = req.body.reviewText;
33             location.save(function(err, location) {
34                 if (err) {
35                     sendJsonResponse(res, 404, err);
36                 } else {
37                     updateAverageRating(location._id);
38                     sendJsonResponse(res, 200, thisReview);
39                 }
40             });
41         }
42     } else {
43         sendJsonResponse(res, 404, {
44             "message": "No review to update"
45         });
46     }
47 }
48 );
49 };
```

DELETE method: Deleting data from MongoDB

Action	Method	Parameters	Example
Delete a specific location	DELETE	locationid	http://loc8r.com/api/locations/123
Delete a specific review	DELETE	locationid reviewid	http://loc8r.com/api/locations/123/reviews/abc

Deleting documents in MongoDB

- Mongoose makes deleting a document in MongoDB extremely simple by giving us the method *findByIdAndRemove*.
- This method expects just a single parameter—the ID of the document to be deleted.

Deleting documents in MongoDB

```
1 module.exports.locationsDeleteOne = function(req, res) {
2   var locationid = req.params.locationid;
3   if (locationid) {
4     Loc
5       .findByIdAndRemove(locationid)
6       .exec(
7         function(err, location) {
8           if (err) {
9             sendJsonResponse(res, 404, err);
10            return;
11          }
12          sendJsonResponse(res, 204, null);
13        }
14      );
15   } else {
16     sendJsonResponse(res, 404, {
17       "message": "No locationid"
18     });
19   }
20 };
```

Deleting documents in MongoDB: 2 step

```
1 Loc
2   .findById(locationid)
3   .exec(
4     function (err, location) {
5       // Do something with the document
6       Loc.remove(function(err, location){
7
8         // Confirm success or failure
9       });
10  }
11 );
```

Deleting a subdocument from MongoDB

- Steps:
 - Find the parent document
 - Find the relevant subdocument
 - Remove the subdocument
 - Save the parent document
 - Confirm success or failure of operation
- We can find a subdocument by its ID with the `id` method like this:
 - **`location.reviews.id(reviewid)`**
- Mongoose allows you to chain a *remove* method to the end of this statement like this:
 - **`location.reviews.id(reviewid).remove()`**

Finding and deleting a subdocument from MongoDB

```
1 module.exports.deleteOne = function(req, res) {
2   if (!req.params.locationid || !req.params.reviewid) {
3     sendJsonResponse(res, 404, {
4       "message": "Not found, locationid and reviewid are both required"
5     });
6     return;
7   }
8   Loc
9     .findById(req.params.locationid)
10    .select('reviews')
11    .exec(
12      function(err, location) {
13        if (!location) {
14          sendJsonResponse(res, 404, {
15            "message": "locationid not found"
16          });
17          return;
18        } else if (err) {
19          sendJsonResponse(res, 400, err);
20          return;
21        }

```


Finding and deleting a subdocument from MongoDB

```
22     if (location.reviews && location.reviews.length > 0) {
23         if (!location.reviews.id(req.params.reviewid)) {
24             sendJsonResponse(res, 404, {
25                 "message": "reviewid not found"
26             });
27         } else {
28             location.reviews.id(req.params.reviewid).remove();
29             location.save(function(err) {
30                 if (err) {
31                     sendJsonResponse(res, 404, err);
32                 } else {
33                     updateAverageRating(location._id);
34                     sendJsonResponse(res, 204, null);
35                 }
36             });
37         }
38     } else {
39         sendJsonResponse(res, 404, {
40             "message": "No review to delete"
41         });
42     }
43 }
44 );
45 };
```

CONSUMING A REST API: USING AN API FROM INSIDE EXPRESS

Calling an API from an Express application

- We tie the front end to the back end
- We'll remove the hard-coded data from the controllers
- End up showing data from the database in the browser instead.
- We'll push data back from the browser into the database via the API, creating new subdocuments.

How to call an API from Express

- Our Express application needs to be able to call the API URLs that we set up, sending the correct request method and interpreting the response.
- To do this, we need a module called *request*

Adding the request module to our project

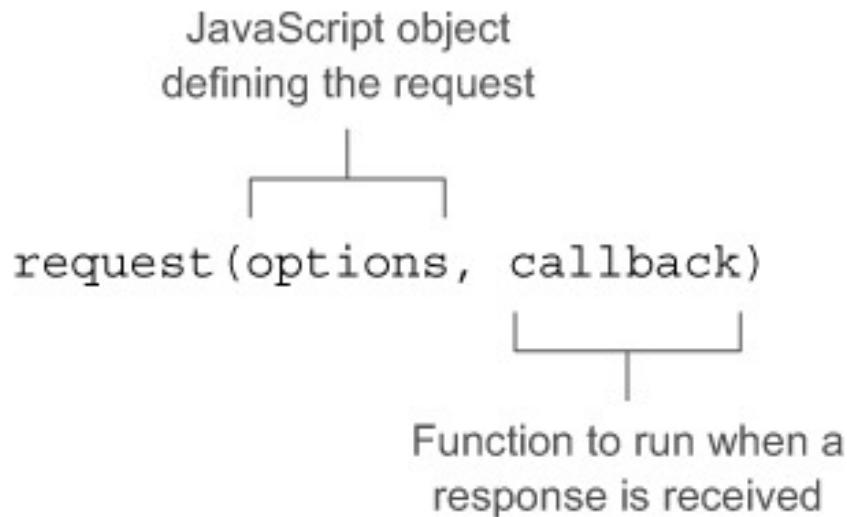
- `$ npm install --save request`
- We only have one file that needs to make API calls, that's the file with all of the controllers
- In `locations.js` in `app_server/controllers` add the following line to require request:
`var request = require('request');`

Setting up default options

- Every API call with *request* must have a fully qualified URL, meaning that it must include the full address and not be a relative link.
- But this URL will be different for development and live environments.
- We can set a default configuration option once at the top of the controllers file.

```
1 var request = require('request');
2 var apiOptions = {
3   server : "http://localhost:3000"
4 };
5 if (process.env.NODE_ENV === 'production') {
6   apiOptions.server = "https://getting-mean-loc8r.herokuapp.com";
7 }
```

Using the request module



The callback argument gets 3 arguments:

1. An error when applicable (usually from [http.ClientRequest](#) object)
2. An [http.IncomingMessage](#) object (Response object)
3. The third is the response body (String or Buffer, or JSON object if the `json` option is supplied)

Using the request module

Four common request options for defining a call to an API

Option	Description	Required
url	Full URL of the request to be made, including protocol, domain, path, and URL parameters	Yes
method	The method of the request, such as GET, POST, PUT, or DELETE	No—defaults to GET if not specified
json	The body of the request as a JavaScript object; an empty object should be sent if no body data is needed	Yes—ensures that the response body is also parsed as JSON
qs	A JavaScript object representing any query string parameters	No

Skeleton for making API calls

```
1 var requestOptions = {  
2   url : "http://yourapi.com/api/path",  
3   method : "GET",  
4   json : {},  
5   qs : {  
6     offset : 20  
7   }  
8 };  
9 request(requestOptions, function(err, response, body) {  
10   if (err) {  
11     console.log(err);  
12   } else if (response.statusCode === 200) {  
13     console.log(body);  
14   } else {  
15     console.log(response.statusCode);  
16   }  
17 });
```

Define options for
request

Make request,
sending through
options, and
supplying a callback
function to use
responses as needed

Using lists of data from an API: The Loc8r homepage

- Now we update the controllers to call the API and pull the data for the pages from the database
- Current homepage controller contains a `res.render` statement sending hard-coded data to the view.
- Now we want to render the homepage after the API has returned some data.

Separating concerns: Moving the rendering into a named function

Listing 7.2. Moving the contents of the `homelist` controller into an external function

```
1 var renderHomepage = function(req, res){  
2   res.render('locations-list', {  
3     title: 'Loc8r - find a place to work with wifi',  
4     ...  
5   });  
6 };  
7 module.exports.homelist = function(req, res){  
8   renderHomepage(req, res);  
9 };
```

Building the API request

Parameter	Value
URL	SERVER:PORT/api/locations
Method	GET
JSON body	null
Query string	lng, lat, maxDistance

Update the homelist controller to call the API before rendering the page

```
1 module.exports.homelist = function(req, res){
2   var requestOptions, path;
3   path = '/api/locations';
4   requestOptions = {
5     url : apiOptions.server + path,
6     method : "GET",
7     json : {},
8     qs : {
9       lng : -0.7992599,
10      lat : 51.378091,
11      maxDistance : 20
12    }
13  };
14  request(
15    requestOptions,
16    function(err, response, body) {
17      renderHomepage(req, res);
18    }
19  );
20 };
```

Using the API response data

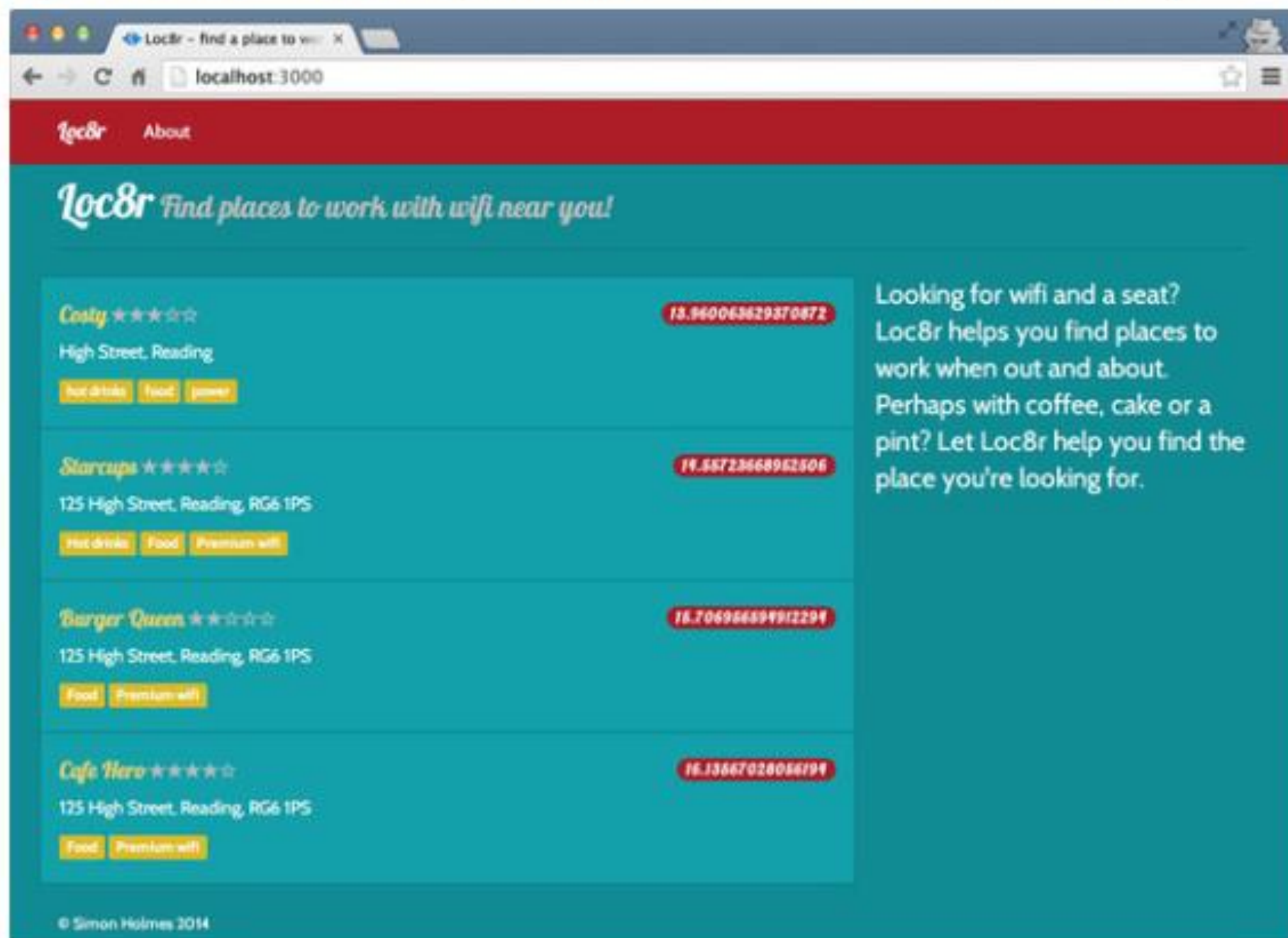
Listing 7.4. Update the contents of the `homelist` controller to use the API response

```
1 request(  
2     requestOptions,  
3     function(err, response, body) {  
4         renderHomepage(req, res, body);  
5     }  
6 );
```

Listing 7.5. Update the `renderHomepage` function to use the data from the API

```
1 var renderHomepage = function(req, res, responseBody){  
2     res.render('locations-list', {  
3         title: 'Loc8r - find a place to work with wifi',  
4         pageHeader: {  
5             title: 'Loc8r',  
6             strapline: 'Find places to work with wifi near you!'  
7         },  
8         sidebar: "Looking for wifi and a seat? Loc8r helps you find places to  
9 work when out and about. Perhaps with coffee, cake or a pint? Let Loc8r  
10 help you find the place you're looking for.",  
11         locations: responseBody  
12     });  
13 };
```

Figure 7.2. The first look at using data from the database in the browser—it's pretty close!



The screenshot shows a web browser window with the address bar displaying 'localhost:3000'. The website has a red header with the 'Loc8r' logo and an 'About' link. Below the header, the main content area has a teal background with the tagline 'Find places to work with wifi near you!'. A list of four cafes is displayed, each with its name, a five-star rating, its address, and a phone number in a red pill-shaped button. The cafes are: Costy, Starcups, Burger Queen, and Cafe Hero. Each cafe entry also has small yellow buttons for 'Hot drinks', 'Food', and 'Premium wifi'.

Loc8r About

Loc8r Find places to work with wifi near you!

Name	Rating	Address	Phone Number	Features
Costy	★★★★☆	High Street, Reading	13.960063629370872	Hot drinks, Food, Power
Starcups	★★★★☆	125 High Street, Reading, RG6 1PS	14.55723668962506	Hot drinks, Food, Premium wifi
Burger Queen	★★★★☆	125 High Street, Reading, RG6 1PS	16.706966899912294	Food, Premium wifi
Cafe Hero	★★★★☆	125 High Street, Reading, RG6 1PS	16.13667028066794	Food, Premium wifi

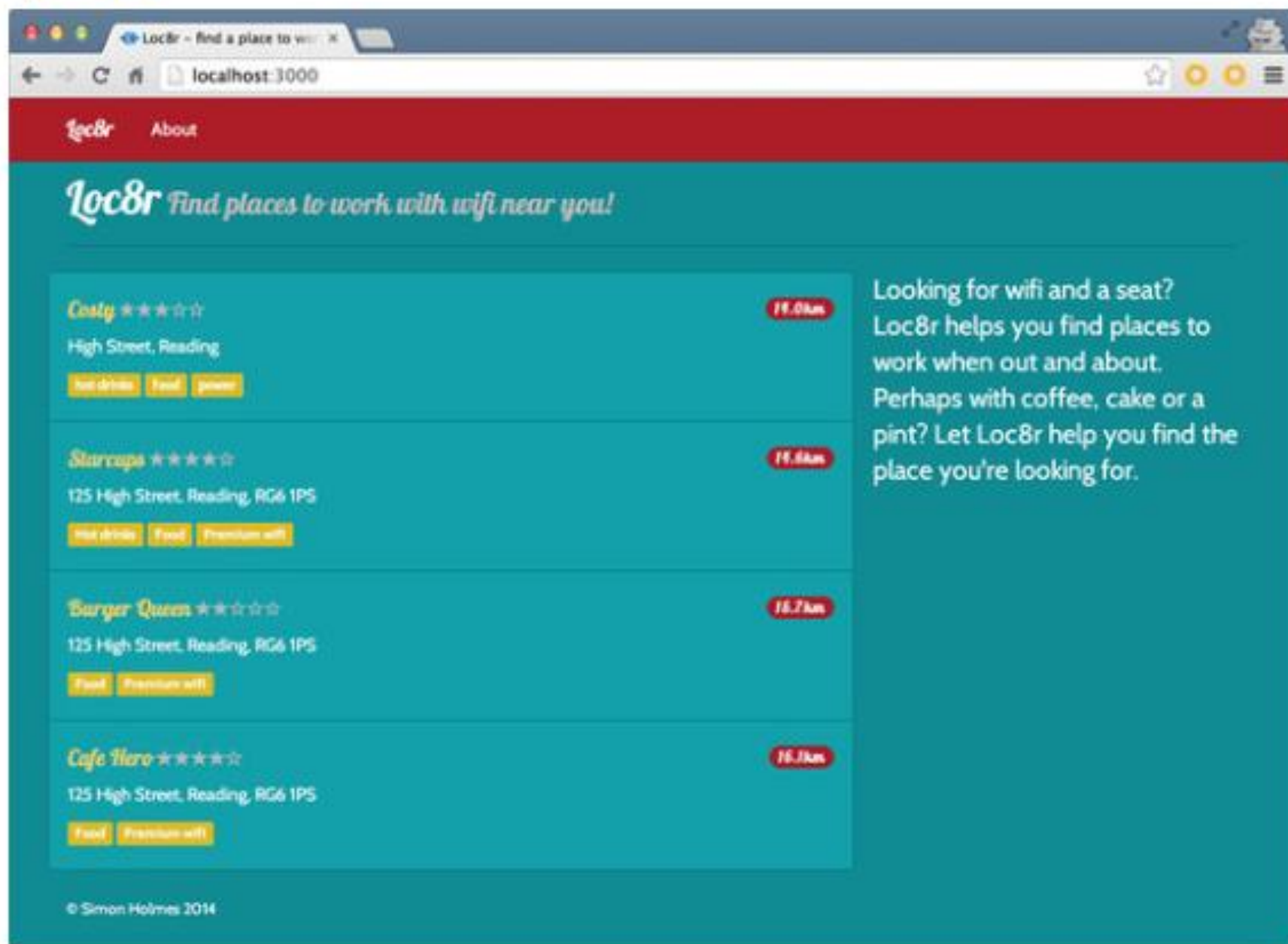
Looking for wifi and a seat?
Loc8r helps you find places to work when out and about.
Perhaps with coffee, cake or a pint? Let Loc8r help you find the place you're looking for.

© Simon Holmes 2014

Modifying data before displaying it: Fixing the distances

```
1 request(
2   requestOptions,
3   function(err, response, body) {
4     var i, data;
5     data = body;
6     for (i=0; i<data.length; i++) {
7       data[i].distance = _formatDistance(data[i].distance);
8     }
9     renderHomepage(req, res, data);
10  }
11 );
12 var _formatDistance = function (distance) {
13   var numDistance, unit;
14   if (distance > 1) {
15     numDistance = parseFloat(distance).toFixed(1);
16     unit = 'km';
17   } else {
18     numDistance = parseInt(distance * 1000,10);
19     unit = 'm';
20   }
21   return numDistance + unit;
22 };
```


Figure 7.3. The homepage is looking better again after formatting the distances returned by the API.



Next

- Getting single documents from an API
- Adding data to the database via the API
- Protecting data integrity with data validation