Coursework Report

Cool Student
4008000@napier.ac.uk
Edinburgh Napier University - Module Title (SET00000)

1 Introduction

In recent decades, blogging platforms have become a popular medium for the sharing a range of information; from journalistic reports and story telling, to food recipes and hobby focused resources. The scope of this assignment was to create a blogging platform with the fundamental functionality to create, read, update and delete (CRUD) blog posts. Aside from CRUD, the addition of other blog features was encouraged. In order to decide which features these would be, it was suggested that research be done into current popular blog platforms to see what functionality they offer.

With regards to technologies, the assignment specified the use of Node JS as the primary technology for the back-end web development of the site, but the addition of other supplementary libraries was allowed. For the front-end, HTML, CSS and JavaScript were required for development of the client interface, but again, other appropriate resources were allowed as long as these were appropriate and justifiable.

Although there was a lot of freedom with the structure of the blog, it was necessary that the platform had some kind of theme to give constraint and focus to the project. The theme for this project was a recipe blog platform where users could create and share recipe posts. Knowing the sites theme meant that more specific research could be done into what features and functionality other recipe platforms offered. Whilst the majority of the inspiration for the project was found from specific blog sites such as Tumblr [1] and Medium [2], BBC Good Food [3] and other recipe sites came into use as a guide for what to plan for in the project.

With the understanding that CRUD was to serve as the foundation of the site, other blog features were planned knowing it was to serve as a recipe platform. A multi-user platform seemed to be important and for some areas of the site to restricted to registered users only. Unlike typical blog posts, more consideration had to be given for the type of information that recipe posts comprise of. Although some of the site would be restricted to registered users only, the core of site would be available to all users so that the content was highly accessible.

2 Software Design

Knowing the rough direction of the blog site being recipe based, planning the implementation approach was the next step before development. This plan will include decision making about what technologies and structure the platform will use as well as sketches and designs used to inform front-end development later down the line.

2.1 Description of the plan

With CRUD established as the core for the blog platform, the planning and design for the site is based around prioritising the implementation of these foundational features.

Create To comply with the assignment specification, the site will include an interface for a user to create a new blog post with at least two input fields; one for a title and one for the body of a post. It is currently undecided specifically what other information a recipe post should include so technology should be chosen with the intention of adding other input fields relatively easily later in development. It seems important for usability that 'create post' interface resemble to interface for viewing a post so that a user understands how their information will be formatted

Read Using both Medium.com [2] and Tumblr.com [1], this site will have a page to display popular posts which will be available to both registered and unregistered users. Tumblr.com [1] seems to prioritise the acquisition of new users by having the default landing page a sign-up/login page. Medium.com [2] on the other hand opts to show popular or selected posts to users on the landing page, with a 'sign-up' banner displayed to users who are not logged in. The approach of Medium.com [2] seems to suit the site better by using the actual blog content as the driving force for acquiring users. In this 'home' page, a user will be able to click through to read the full post.

Update To a logged-in user who is viewing their own post, the option to both edit and delete that post should be available. Obviously this is an option that should only be available to the author of that post and some kind of authentication mechanism will need to oversee that. Like with creating a post, the 'edit post' interface should resemble the interface for reading to maintain consistency and help users create well-formatted recipes.

Delete Like with creating and updating, this is a feature that will only be available to users who are logged-in and so will need some authentication. It also seems like a good idea to use a UI feature such as a confirmation modal to prevent accidental post deletion. This is likely something that will need to be done client side with JavaScript.

Database choice Since posts are recipes and not typical blog posts, this should be reflected in appropriate choice of supporting technologies. Recipes vary widely in terms of their complexity and what information each includes. They typically have images accompanying them as well as meta information such as recipe time and number of servings. There might also be the potential for the expansion of what information is included with a recipe. Taking all of this into account, it seems appropriate to use a NOSQL database structure to support the storing of recipes due to it offering higher flexibility and requiring less planning. A cloud based MongoDB service such as mLab seems appropriate, at least for the development stage as little setup is required. Data could perhaps be moved to a more permanent solution in the future. Seeing as mLab will be used to store recipe data, it also makes sense to use this for storing users and any other data necessary for the site.

Multi-user After CRUD implementation, a good blog site will have multi-user support. This will mean creating a storage system for the user data - mLab and MongoDB will serve nicely for this. More complexly, the server program will need to support user authentication with security features. It would be wise to also encrypt user passwords on registration and create validator checks for submitted user information. Node JS has modules to help with this such as Passport JS. Most of the site will need to be accessible to users who have not registered and so it is important to keep the files handling user authentication seperate from the bulk of the site.

Login/sign-up Although the landing page will be a collection of recipes instead of a login/sign-up interface, a link to these should still be included in the header so that a user can easily navigate to them. Separate HTTP routes have the benefit that they can be linked to directly. However, a login/sign-up modal could be displayed from any page without diverting too much from a users experience of the site. For this site, both options would be ideal, however, absolute addresses for login/sign-up pages is the priority so modals will not be developed until later stages.

Recipe information what information to include with a recipe post As specified above, for the purposes of ease of development, initially recipes will just include a title and content. In later stages of the project, more information will be added to these posts. BBC Good Food [3] as well as Medium.com [2] and Tumblr.com [1] served as good sources for deciding what information was necessary for a post to include. A good balance is needed between displaying helpful information to a reader of a recipe whilst not requiring too much information when creating one. Recipe information should include:

- recipe title
- recipe steps
- date the recipe was published
- author of the recipe
- recipe image
- cooking time
- difficulty level of the recipe
- list of ingredients
- number of 'likes'

Most of this information should be easily obtained from a user when they create the recipe and information such as the publishing date and the author of the recipe can be collected automatically when the recipe is created.

2.2 Page templates and artefacts

As with any blog, most of the pages will be dynamically loaded and populated with back-end data such as the recipe pages. Therefore, a templating engine will need to be used to declare where information should be placed on each page. There are a few different engines to choose from such as Pug.js (formerly Jade.js), but the chosen engine for this site is Handlebars.js which most closely resembles stock HTML. This engine will enable functionality like the 'home' page being filled with indefinite numbers of recipes and each recipe page being displayed from one template. Handlebars also supports partial views so that one piece of HTML code can be loaded across many pages - useful for elements such as a header or a footer. For the recipe site, a header with a logo and navigation links will be used as a partial across all the pages.

Landing page [figure 2] As specified above, the landing page for the site will primarily show recipes from the site - as a showcase of what the platform offers. The recipes will be displayed as thumbnails on the landing page which will display to the user necessary information to tell them about what the recipe includes. A title and the author of the recipe but also the recipe length, difficulty and how many it serves. A picture of the meal will also be displayed on this thumbnail. For users who have not logged in, a banner will be displayed below the recipe thumbnail section. This will be a prompt for users to create an account and provide some information about what comes with the account. The overall design for the site won't contain much colour as most of the colour will come from the images of food. This was a design pattern noticed when researching similar platforms. Coloured images on a site are made weaker the more that other colours are used and, seeing as the site includes pictures of food, it seemed a fitting decision.

Login/Sign-up page [figure 3] The login/sign-up interface has been designed as a modal although the finished site will also have a unique URL for logging in and signing up. The modal is triggered from the header of the site which appears across all pages. The intention with this is to invite users to sign up at any point in their exploration of the site. Triggering the modal fades out the site below to give focus to the action at hand whilst still showing the user where they are in the site. Completing the form from one of these modals triggers the same login/sign-up 'post' method as a static page.

Recipe page [figure 4] Like the recipe thumbnails, recipe pages are to be filled with data on page load. However, these will include much more detailed information such as the date the recipe was publish and the list of ingredients. There will also be related recipes at the side of the recipe to encourage users to read more - especially useful if a user hasn't found what recipe they were lookig for.

404 Error page [figure 5] Traditionally, 404 pages have negative connotations of confusion or irritation and can also indicate a poorly made site. Making a custom 404 page to greet a user is an excellent way to retain a user. It's also an opportunity to add personality to a site which reinforces a brand. The 404 page will catch unknown site URIs and will have some suggested recipes to guide the user back to familiar territory. The site header will also be included using a Handlebars partial to let the user know they've not left the site completely.

3 Implementation

Having planned the site, implementation involved development according to the requirements laid out in the plan. Most of the initial stages of implementation were foundational and back-end, involving structural work such as file structure and setting up the database.

3.1 File Structure

As required in the specification, Node JS was used as the server for the application which was then added to with other modules. One of the primary modules used as a framework for the application was Express JS [4] which provides file structure to a new project [figure 1]. It helps organise assets and manages routing which can be tricky with any HTTP server. To help manage the file structure and to develop the range of file types necessary for this project, Atom text editor was the choice for development environment. Atom also has plugins which allows for command line use within the application - very helpful for developing a web server.

3.2 Database Setup

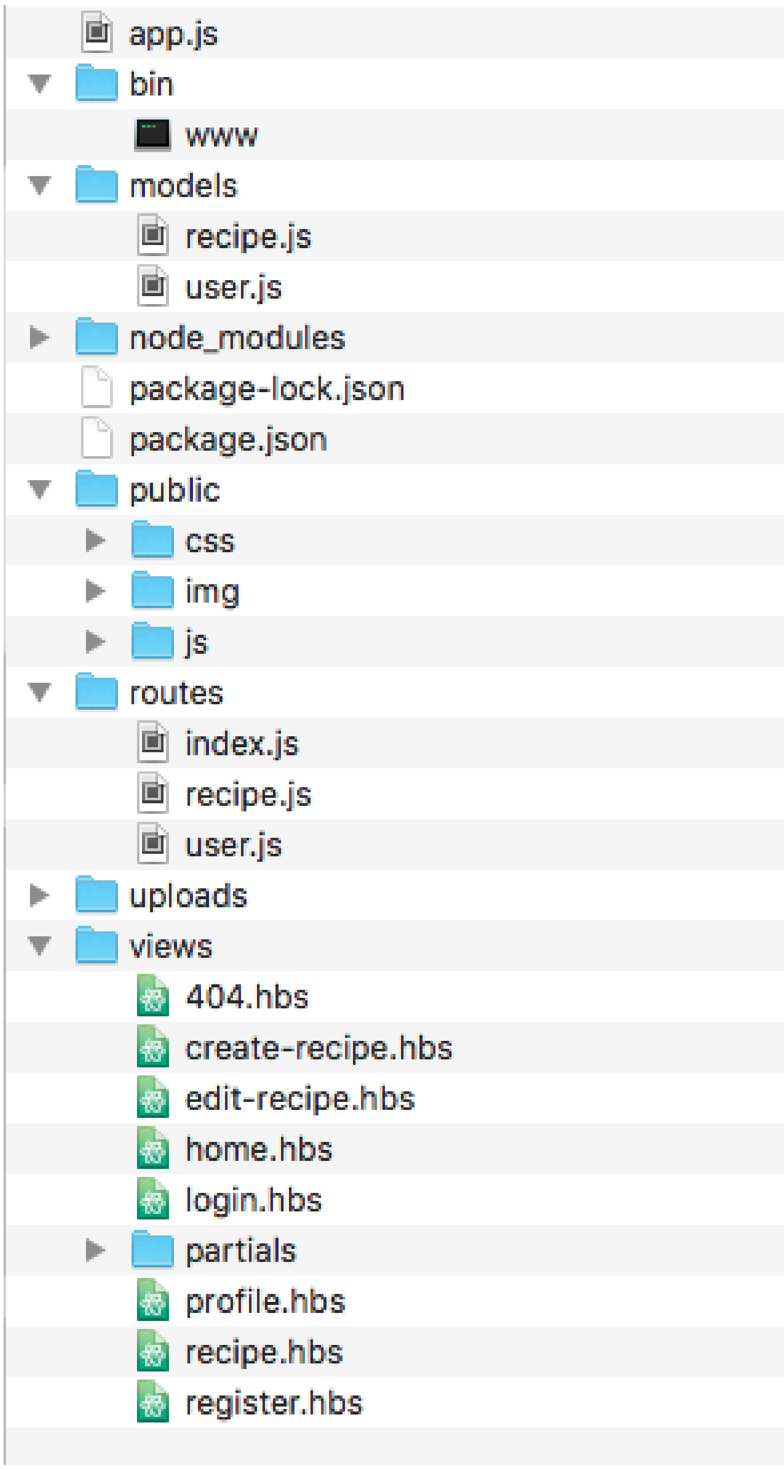
After setting up the local file system, a Mongo NOSQL database hosted remotely using mLab was created. This database was linked to recipe application using a URL and another NPM module called Mongoose [5] was used which makes working with Mongo databases much more straightforward partly by reducing the amount of syntax necessary for common Mongo tasks. With Mongoose, a connection is made using the database URL and then schemas written to model how the data for different objects should be stored in the database (see listing 1).

Listing 1: User schema

```
// Schema for users
var userSchema = new Schema({
    name: {type: String, required: true},
    email: {type: String, required: true},
    username: {type: String, required: true, index: true, unique: ↩
       true},
    password: {type: String, required: true}
},{ collection: 'users' });
```

3.3 Users

Next, interactions with the front-end began by creating a form for a user to sign up to the site. This included a



```
    app.js
▼ bin
        www
▼ models
        recipe.js
        user.js
▶ node_modules
    package-lock.json
    package.json
▼ public
    ▶ css
    ▶ img
    ▶ js
▼ routes
        index.js
        recipe.js
        user.js
▶ uploads
▼ views
        404.hbs
        create-recipe.hbs
        edit-recipe.hbs
        home.hbs
        login.hbs
    ▶ partials
        profile.hbs
        recipe.hbs
        register.hbs
```

Figure 1: File structure mostly produced by Node JS and Express JS

full name, email address, username and a password. This mapped to the user database schema and on creation, that users details get passed to the database. All of the user object fields in the database were set as 'required' and some other requirements were needed to be set for the user data. This was done on the back-end with the help of another module called Express-Validator [6] which provides checks for common data types such as checking for valid email addresses or specific string lengths. Any errors produced from these checks were passed back to the user in the form of descriptive error messages. Another process to the user data was hashing passwords with a module called Bcrypt [7]. This ensures that the password data stored in a database is encrypted. The module also included validator check functions for when a user wants to login to an existing account. The last check that the registration form does is to see if a username is unique. This is because the username field is required to be unique by the database schema and therefore must be validated as so.

3.4 Sessions and authentication

Although the application had been setup to persist user data through user accounts, any changes on the server side or a user leaving the recipe site and returning would mean a user having to log in repeatedly which is frustrating and inefficient. Therefore, session storage was implemented so that a users site session information persists to the database and is pulled up when a user accesses the site. Session persistence being implemented also meant that the site could check if a session was logged in and therefore apply restrictions to areas of the site depending on the login state. This was done using a module called Passport JS [8] which adds the capability of using a function on 'get' and 'post' methods that need to be restricted. Creating, editing and deleting recipes, for example, is restricted to registered users only and so access to these features is blocked for sessions that do not belong to a logged in user. This same authentication could be also used to only display 'register' and 'sign in' buttons and information to users who are not signed in.

3.5 Recipes

Edit and delete recipe With the implementation of user access completed, the next task was allowing users to CRUD recipes. Just the same as with users, a schema was created for recipes (see listing 2). This schema had many more fields which increased as development progressed and with the addition of new features. There was also more variety of data types including date and number. A 'create recipe' page is used by a user to create a recipe object from this schema which is then passed to the database.

Listing 2: Recipe schema

```
var recipeSchema = new Schema({
    title: {type: String, required: true},
    description: {type: String, required: true},
    steps: [
        {type: String, required: true}
    ],
    author: {type: String, required: true},
    published: {type: Date, default: Date.now(), required: true},
    recipeTimeHrs: {type: String, required: true},
    recipeTimeMins: {type: String, required: true},
    difficulty: {type: String, required: true},
    serves: {type: String, required: true},
    image: {type: String, required: true},
    favourites: {type: Number, default: 0}
},{ collection: 'recipes' });
```

The input fields for the recipe creation required some more rigorous data validator which was again facilitated by Express-Validator [6] but in a more extensive and thorough form. After a user has created a recipe, they are redirected to the 'home' page which, on load, is populated with all recipes from the database which is passed through the get method. Handlebars is used to loop through all these recipes (see listing 3) and populate thumbnails with the data.

Listing 3: Handlebars recipe thumbnail loop

```
{{# each recipes }}
<article class="recipe−thumbnail recipe−thumbnail−portrait">
    <a href="/recipe/{{ this._id }}"><img class="recipe−img" ↩
src="{{ this.image }}" alt=""></a>
    <div class="caption">
        <div>
            <h4><a href="/recipe/{{ this._id }}">{{ this.title ↩
}}</a></h4>
            <p><a href="/user/{{ this.author }}">{{ this.author↩
 }}</a></p>
        </div>
        <div>
            <ul>
                <li><img src="/img/time.svg">{{ this.↩
recipeTimeHrs }} hrs {{ this.recipeTimeMins }} mins</li>
                <li><img src="/img/difficulty.svg">{{ this.↩
difficulty }}</li>
                <li><img src="/img/people.svg">{{ this.serves }}↩
 Serving</li>
            </ul>
            <button class="btn−icon" type="button" name="↩
favourite"><img src="/img/favourite.svg"><br>{{ this.↩
favourites }}</button>
        </div>
    </div>
</article>
{{/each}}
```

Not all of the recipe data is shown in these thumbnails but clicking on either the title of the recipe or the picture takes a user to the full recipe page (see figure 4). The full recipe page is populated using handlebars once again except without a loop and only one recipe object is passed. Recipes are navigated to using unique URLs created from each recipes unique database ID. The recipe data passed is searched from the database using this same database ID. After the structure for handling the main recipe data was added such as the title and body, other recipe data was handled like the cooking time and, importantly, the recipe image. This image is uploaded by a user on recipe creation and is stored on the server using a module called Multer [9]. The path to the image file on the server is stored with that recipe in the database. When that recipe is loaded, the server looks for an image path matching the path stored in the recipe object and then produces the matching image. The actual image file is not stored in the database.

The recipe page also includes a button to edit the recipe and one to delete it. The delete button triggers a dialogue box for the user to confirm the deletion of that recipe. Confirming this takes the client to a URL which deletes that recipe and then redirects to the home page. The edit button takes the user to a page using a very similar template as the 'create recipe' page but this is populated with the selected recipe's data. After pressing the 'save' button on that page, the server overwrites the old recipe with a new one created with the new data. Both the edit and delete paths are restricted to only be accessible to the user matching that recipe's author.

3.6 CSS Styling

The final stage in the main development of the site was to style and fine tune the front-end of the website. Before now, the styling and layout had been controlled with Bootstrap which worked well for testing and mocking up the site. Each bootstrap element was replaced piece by piece with custom CSS styling which complied with the designs shown in the Software Design section. The styling for the site is fairly minimal because of the use of Handlebars templates and classes. For example, the recipe thumbnails are only styled once but the same styling is used across multiple pages such as the home page and the 404 errors page. Styling was split between two files, a main file holding the main classes and the header styling, and a recipe stylesheet which styles all three recipe pages, creating, reading and editing.

3.7 Bug Fixing and Testing

Lastly, the site needed testing and some bug fixing. Most of the testing was testing the URLs and routing of the site as this area would cause the most problems if it broke or had any errors. There were some errors relating to the restriction of pages where more specific solutions had to be coded along side the functions provided by Passport JS. Another bug included a problem with making the paths to user uploaded images work across the whole site. This was solved by making the 'uploads' folder static.

4 Critical Evaluation

A critical evaluation of the blog platform implementation can be performed by comparing the project with two metrics. Firstly, how the blog platform compares to the original specification and requirements laid out in the planning and design stage. Secondly, how the blog platform compares to similar platforms in the field, especially those used as inspiration or reference initially. These comparisons and other reflections can be used to produce a list of possible improvements.

4.1 Comparing to requirements

Technology and libraries The original specification required a blog platform built with Node JS as the back-end server software with HTML, CSS and JavaScript. It specifies that other modules and libraries can be used but must be justified. The finished platform meets this and is built on Node JS and uses HTML, CSS and JavaScript for the front end. The use of additional libraries is extensive with this project, especially on the back-end which is a big time saver and ensures that important features such as password hashing is done correctly. The downside of relying heavily on libraries is it produces a less flexible, bloated platform, but for speedy development of a general platform such as a blog, the pros out-weight the cons.

Recipe steps ingredient lists The core requirements of the site have been met as well as most of the extra features that were specified such as multi-user support and additional recipe information. However, some of the features defined in the requirements are lacking. Recipes do not fully support multiple steps in their instructions. Some of the framework for this is there and using MongoDB made adding an array of recipe steps easy from a database point of view. The problems came with trying to validate this process, especially with a variable number of steps from recipe to recipe. The same problem would have been found with having lists of ingredients with recipes. Both of these features could have been implemented with more time, but it was clear that validating and trouble-shooting problems with these would have been excessively time-consuming.

Favouriting system In the original plan and designs, a system for 'favouriting' was planned. This is a feature common across many social platforms namely Medium.com [2]. Working out how to relay front end DOM interactions to the server and database was a new element to learn, so similar to the recipe lists, this was not developed past cosmetics and database framework.

Recipe search This was a feature that also seemed important to a site with a potentially indefinite amount of content. Realistically however, building a good custom search into a site is hugely time-consuming and complicated and, while there are modules to help with this, it wasn't a priority in comparison to the core site features.

4.2 Comparing to similar platforms

BBC Good Food [3] BBC Good Food is the most similar site that was used as direct reference and inspiration for this recipe site in terms of content and structure. The core structure is very similar with recipes show cased on the home page to both registered and unregistered users. Recipes include a lot of the same information apart from recipe steps and an ingredients list. Obviously BBC Good Food is a much more developed site, but there are some key features where the assignment site falls down. The assignment site does not have much in the way of recipe categorisation. BBC Good Food has recipe categories such as 'healthy', 'seasonal', vegetarian' and 'everyday'. The assignment site has some limited categorisation such as the difficulty of the recipe but it is not as extensive as BBC Good Food. Having categories like this helps reduce the complexity of a site with extensive content.

Medium.com [2] A lot of the inspiration for the design choices for the assignment site were taken from Medium.com. The minimal colour and clean design made sense for a site that was content lead, where the aim was to emphasise the pictures of food. One of the problems with a minimal site is that it can lack personality or brand strength. This is one of the main design differences between the assignment site and Medium.com. Medium has a stronger brand presence brought out by the use of graphics and tasteful addition of colour whereas the assignment site does not really have this. However, one thing that the recipe site does do well is how the flow from home page to recipe to profile feels similar to that of Medium.com.

4.3 Improvements

The core structure of the recipe site feels well developed and relatively clean, but there are some improvements and additions that could be made to the platform in the future.

Content organisation Any site with a large amount of content, especially one that is used for reference such as recipe sites, needs to be well organised and structured. The current recipe site lacks this which becomes increasingly obvious the more content that is added to the site. This could be solved by using recipe categories like the ones seen on BBC Good Food. Even five or six categories would help reduce the complexity a considerable amount. Another organisation improvement could be filtering what is shown on the home page by popularity or publish date. This is where the favourite button would come into play.

User shopping list A feature that was considered in the design was providing support for a user shopping list. A user could add items from a recipe to their shopping list and then they'd have an aggregated list of ingredients that they needed for recipes. This feature would be more useful to a mobile version of the platform.

5 Personal Evaluation

The process of developing this project has provided a lot of opportunity for education, challenge and self development. I learned lots of new technologies and platforms as well as developing a good work flow. I was forced to work around many problems by thinking of creative solutions which gave me personal skills that I can take into other areas of my life.

Node JS In terms of the technical aspect of the project, it was my first time working in back-end web development. I did my development on a Mac and initially, I had to learn a lot about the terminal and how it works in relationship to hosting a web server. This was very new to me and confusing at first, but once I grasped the basics, it became a powerful tool and I ended up preferring tasks on it to their GUI counterparts. Thankfully, I had a good amount of experience with JavaScript so using Node JS as a platform felt familiar even though I hadn't used it before. Part of learning a new technology is learning how to use the documentation that accompanies it, in this case, that was the NPM documentation. Once I understood the structure of that, I could use it to quickly read up on modules I needed to use or functions I didn't understand.

Database When it came to the database integration, it took a long time for me to decide on which service to use. I started the project with an SQL database as I had experience with this, but I encountered problems with the rigidity of it so I moved to a NOSQL database which was a new technology to me. Thankfully, MongoDB and the NPM specification has a large amount of literature explaining how to set up. Using the NPM module Mongoose made working with the database much more straightforward.

Handlebars Handlebars was a templating engine that I had heard about which informed my decision to use it over Pug which is the default with Node and Express JS. Although I had heard of it, I had never used it and did not understand its purpose. I had particular problems understanding how to integrate partials on all of the pages properly and how each template would relate to each other. A lot of trial and error was done to learn this but it also taught me a lot about how the file structure of the project worked.

CSS Styling Initially I used Bootstrap to mock up the format for the recipe site. It was also my first time using bootstrap and, although I had a good understanding of what it was, I never saw the point. Being able to quickly see the front-end result of back-end work in a clearly laid out way was hugely helpful for the initial setup of the project. Especially when trouble shooting problems, a clean interface and well formatted error messages helped show where the problems lay.

Shortcuts I also learned a lot about workflow and strategy. Shortcuts come back to bite you and working out problems is always better than finding solutions online. There were a few examples, especially early on, where I would find solutions to problems on forums or YouTube videos. These were solutions that I didn't understand and would more-or-less blindly copy them across. Later in the development process, I would come across a problem caused by the code I didn't understand. Not knowing how it worked meant that it would take a long time to troubleshoot and thus, take far longer in the long run. Having learned this, future problems were solved by working through the problem logically instead of using a quick fix. It meant that I had a much better knowledge of my code and increased the speed of development and the quality of my work.

Time management Things will always take longer than expected. Planning to have tasks completed on a project like this can be frustrating without the knowledge that they will take much longer than expected. Taking a break from a problem for a few hours and coming back to it refreshes the angle of attack and helps to produce creative and more rewarding solutions to problems instead of forcing through them. This is something I learned when solving the problem of the image paths from Multer. Leaving the problem and returning to it a few days later after working on something else meant that I considered the approach in a new light.

Starting earlier One big difference in this project compared to previous ones was the amount of time I allocated and how early I started. I started the actual development over two weeks before the deadline and started brainstorming even earlier than that. This meant that I had more time for planning the project and was able to prioritise core features of the site.

In summary I am much happier with the result of this assignment than the last web technology assignment. The process felt much more controlled and well planned out. I worked in a more controlled and less sporadic manner which meant I was less stressed when I inevitably encountered problems. In terms of the technologies I used, using Node JS in combination with all of the libraries and resources available with it showed me that surprisingly complex and powerful web applications can be developed in a short period of time. The documentation and forums for these made the development experience encouraging and motivating and it is a platform that I'm looking forward to using again in the future

References

[1] "Tumblr." https://www.tumblr.com/. [Online; accessed 03-April-2018].

[2] "Medium." https://medium.com/. [Online; accessed 03-April-2018].

[3] "BBC Good Food." https://www.bbcgoodfood.com/. [Online; accessed 03-April-2018].

[4] "Express." https://www.npmjs.com/package/express. [Online; accessed 27-March-2018].

[5] "Mongoose." https://www.npmjs.com/package/mongoose. [Online; accessed 27-March-2018].

[6] "Express Validators." https://www.npmjs.com/package/express-validators. [Online; accessed 27-March-2018].

[7] "Bcrypt." https://www.npmjs.com/package/bcryptjs. [Online; accessed 27-March-2018].

[8] "Passport." https://www.npmjs.com/package/passport. [Online; accessed 27-March-2018].

[9] "Multer." https://www.npmjs.com/package/multer. [Online; accessed 27-March-2018].

Inspiration
https://www.tumblr.com/
https://medium.com/
https://www.bbcgoodfood.com/

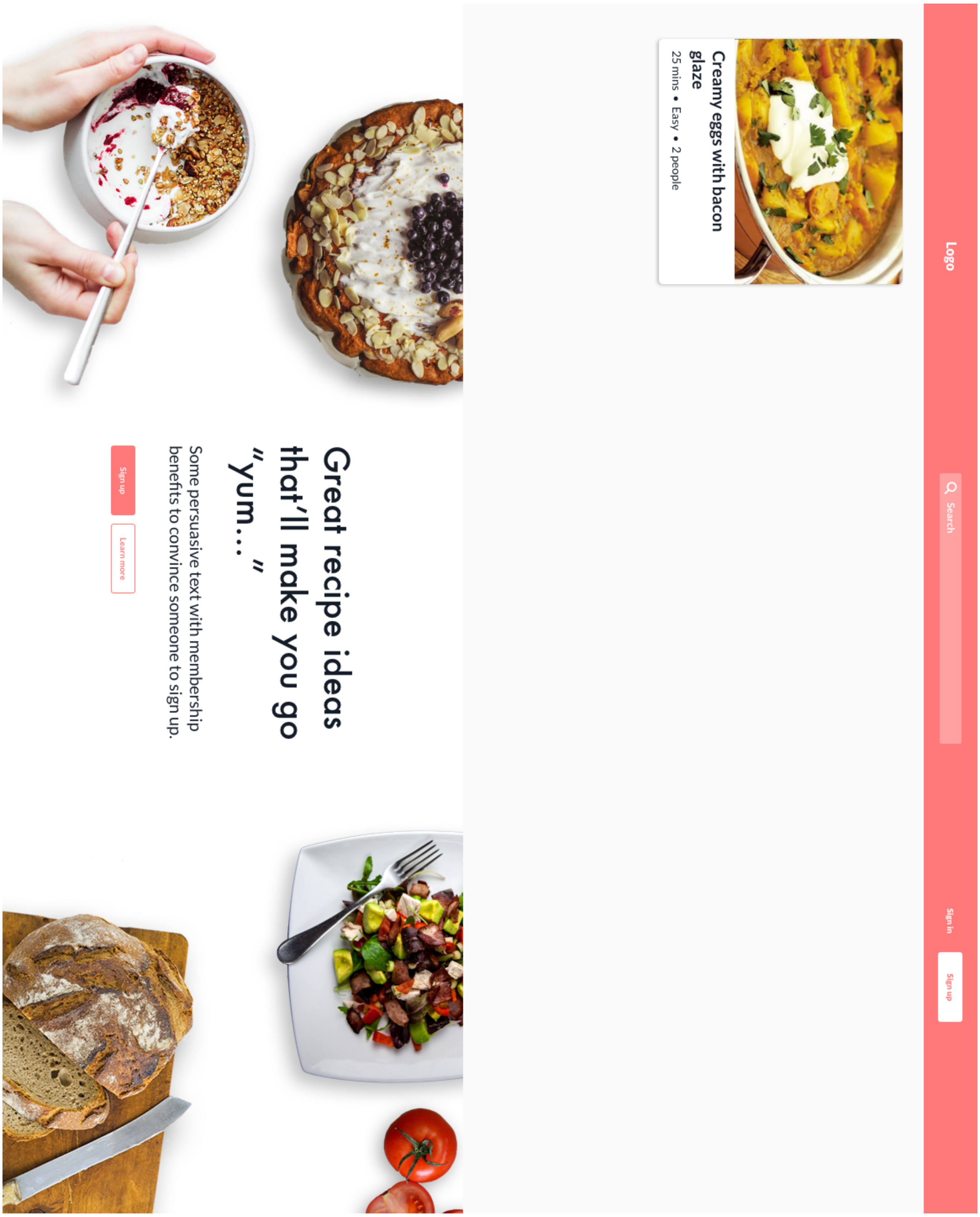
Fonts
https://fonts.google.com/specimen/Lato
https://fonts.google.com/specimen/Montserrat

Images
https://www.pexels.com/photo/person-mixing-cereal-milk-and-strawberry-jam-on-white-ceramic-bowl-704971/
https://www.pexels.com/photo/pastry-with-nuts-sliced-mangoes-and-blueberries-on-top-159887/
https://www.pexels.com/photo/board-close-up-cooking-delicious-373019/
https://www.pexels.com/photo/appetizer-avocado-bacon-cuisine-551997/
http://caramel.ddwcolor.com/wp-content/uploads/2015/08/2-bread-muffins-2.png

jQuery
https://jquery.com/

Bootstrap
https://getbootstrap.com/

Modules
https://www.npmjs.com/package/bcryptjs
https://www.npmjs.com/package/body-parser
https://www.npmjs.com/package/connect-mongo
https://www.npmjs.com/package/cookie-parser
https://www.npmjs.com/package/express-flash
https://www.npmjs.com/package/express-validators
https://www.npmjs.com/package/express
https://www.npmjs.com/package/handlebars-dateformat
https://www.npmjs.com/package/hbs
https://www.npmjs.com/package/http-errors
https://www.npmjs.com/package/mongoose
https://www.npmjs.com/package/morgan
https://www.npmjs.com/package/multer
https://www.npmjs.com/package/passport-local
https://www.npmjs.com/package/passport
https://www.npmjs.com/package/path
https://www.npmjs.com/package/serve-favicon
https://www.npmjs.com/package/session



Creamy eggs with bacon glaze

25 mins • Easy • 2 people

Great recipe ideas that'll make you go "yum..."

Some persuasive text with membership benefits to convince someone to sign up.

Figure 2: Landing page design

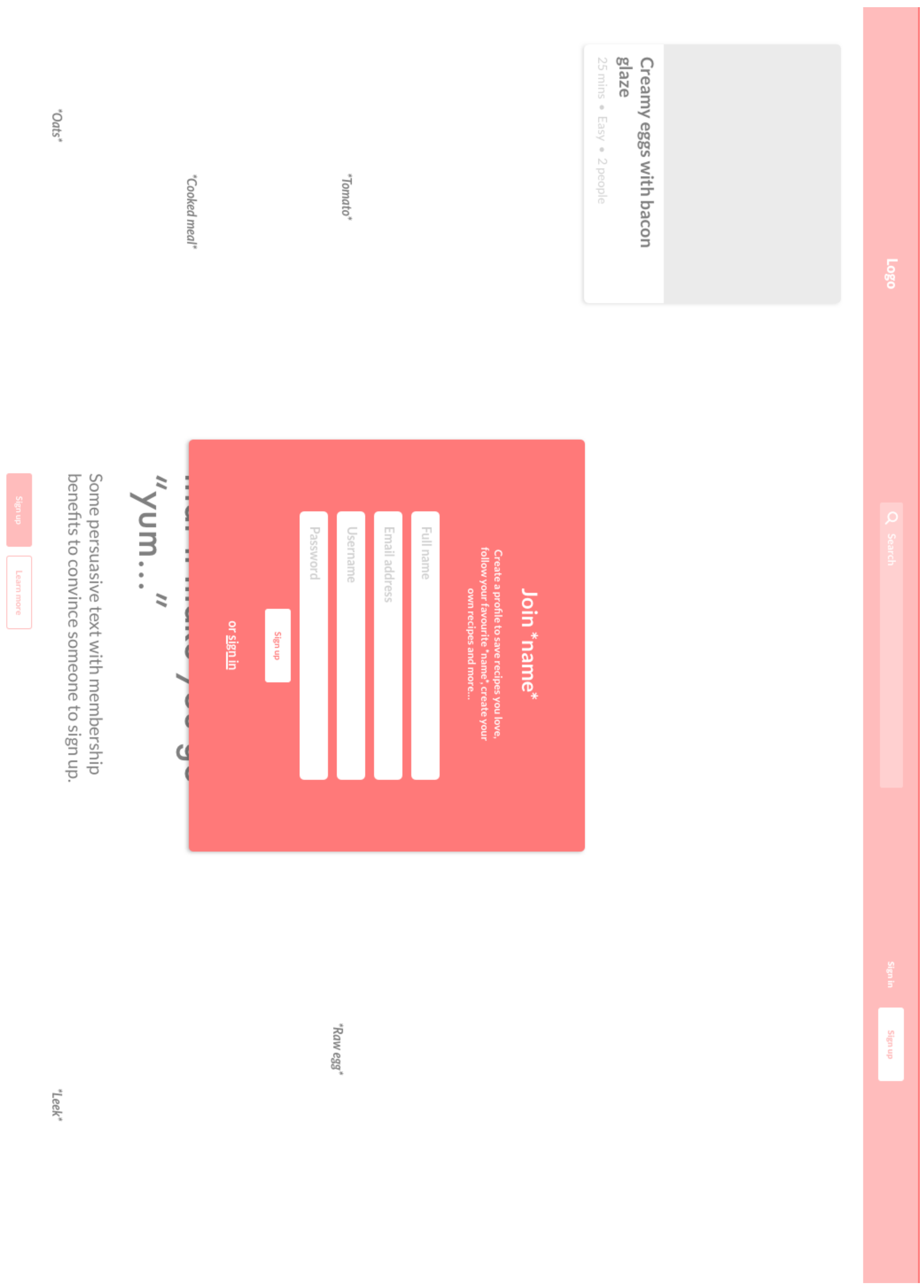
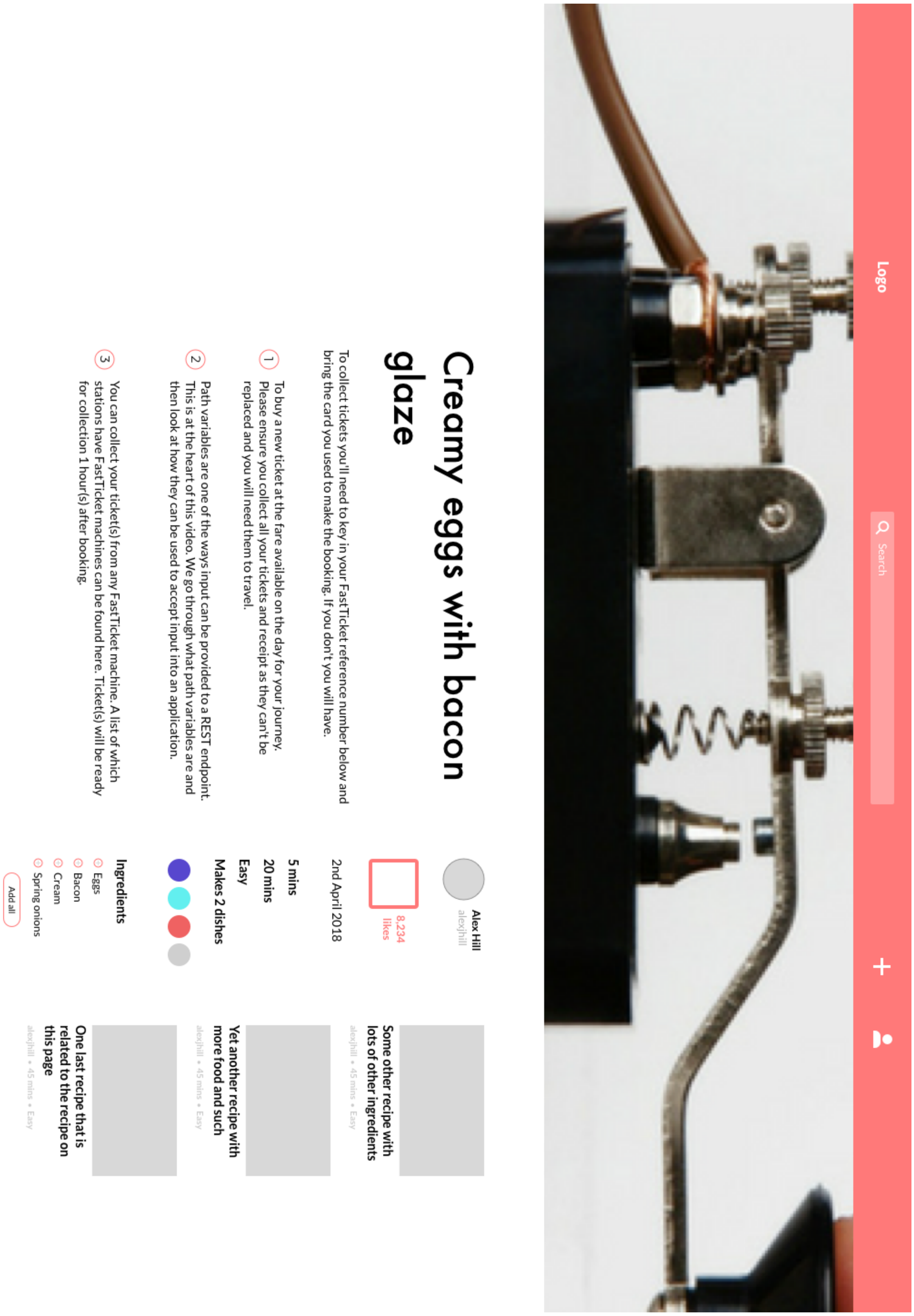

Logo
Search
Sign in
Sign up
Creamy eggs with bacon glaze
25 mins • Easy • 2 people
Join *name*
Create a profile to save recipes you love, follow your favourite *name*, create your own recipes and more...
Full name
Email address
Username
Password
Sign up
or sign in
"yum..."
Some persuasive text with membership benefits to convince someone to sign up.
Sign up
Learn more
Tomato
Cooked meal
Oats
Raw egg
Leek

Figure 3: Landing page design



Creamy eggs with bacon glaze

To collect tickets you'll need to key in your FastTicket reference number below and bring the card you used to make the booking. If you don't you will have.

1. To buy a new ticket at the fare available on the day for your journey. Please ensure you collect all your tickets and receipt as they can't be replaced and you will need them to travel.
2. Path variables are one of the ways input can be provided to a REST endpoint. This is at the heart of this video. We go through what path variables are and then look at how they can be used to accept input into an application.
3. You can collect your ticket(s) from any FastTicket machine. A list of which stations have FastTicket machines can be found here. Ticket(s) will be ready for collection 1 hour(s) after booking.

Alex Hill
alexjhill

8,234 likes

2nd April 2018

5 mins

20 mins

Easy

Makes 2 dishes

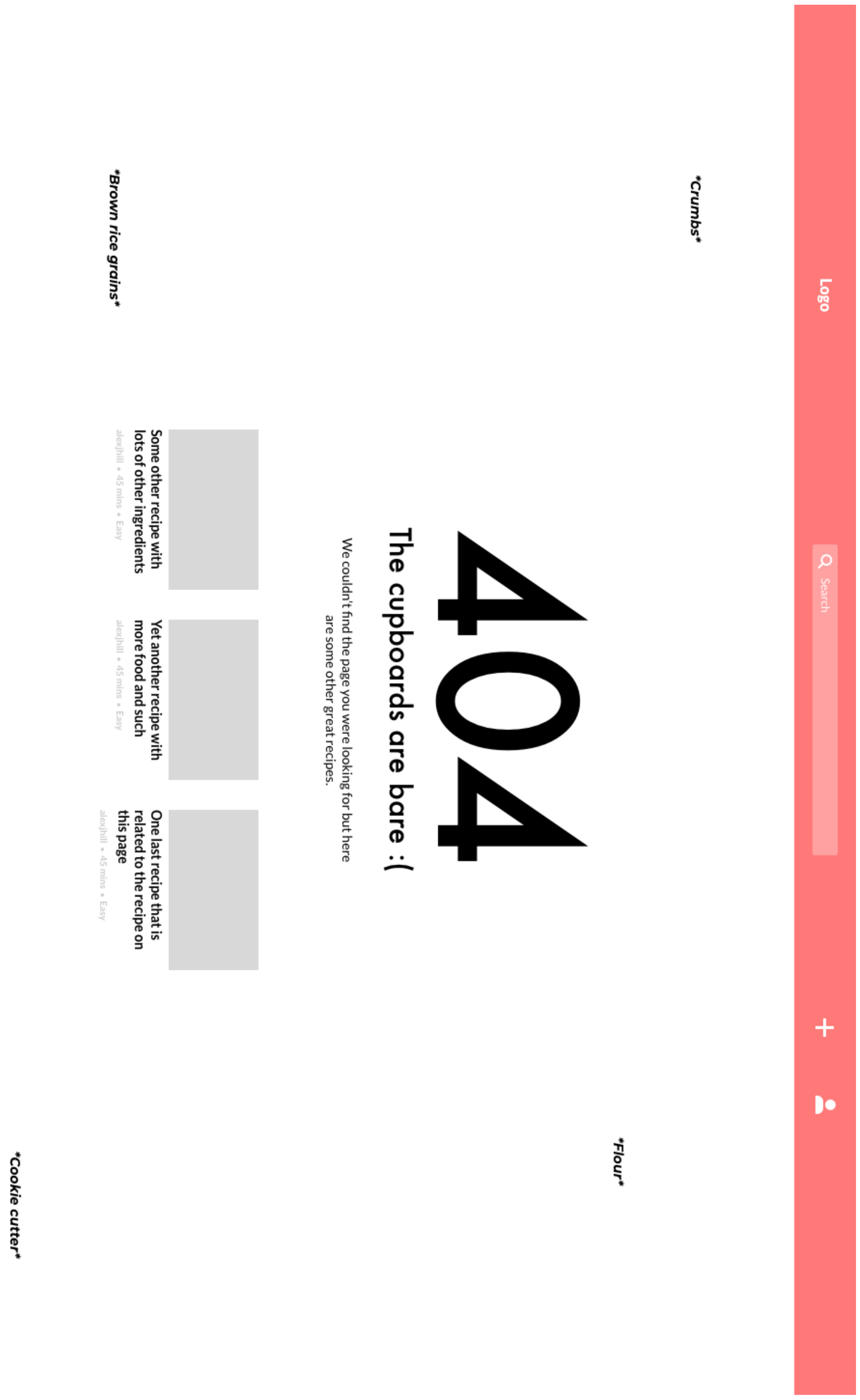
Ingredients
- Eggs
- Bacon
- Cream
- Spring onions

Some other recipe with lots of other ingredients
alexjhill • 45 mins • Easy

Yet another recipe with more food and such
alexjhill • 45 mins • Easy

One last recipe that is related to the recipe on this page
alexjhill • 45 mins • Easy

Figure 4: Landing page design



404

The cupboards are bare :(

We couldn't find the page you were looking for but here are some other great recipes.

Some other recipe with lots of other ingredients
alexjhill • 45 mins • Easy

Yet another recipe with more food and such
alexjhill • 45 mins • Easy

One last recipe that is related to the recipe on this page
alexjhill • 45 mins • Easy

Figure 5: Landing page design