SET08101 Coursework Report

Kinga Kieczkowska (40205426)

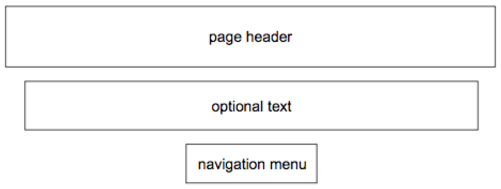
April 2018

1 Introduction

The aim of this coursework is to produce a blog web application with the functionality to view, create, update and delete entries. The technologies used are Node.js for server configuration, MongoDB (1) for data storage and HTML (Pug (2), JavaScript and CSS as well as the Express (3) framework for the application skeleton and the front end. The API which facilitates the CRUD character of the web application has been designed with reference to the REST guidelines as per Microsoft documentation (4) and Hackernoon article (5).

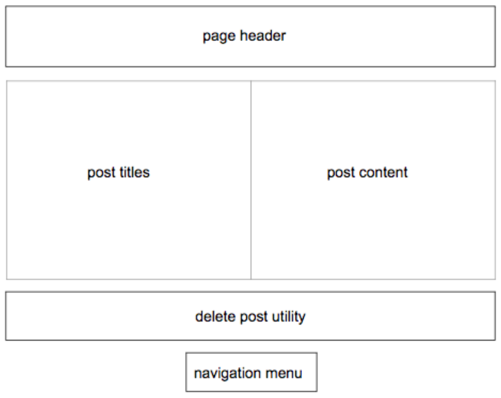
2 Software design

2.1 Homepage



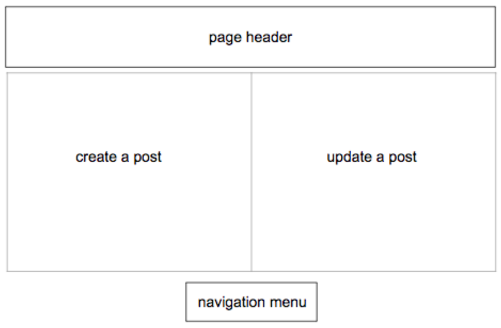
The homepage is supposed to be the first point of contact the user has with the web application, hence its simplicity. The main function of it is showcasing the design style and providing a navigation menu for the user to be able to proceed to the interactive part of the web application.

2.2 Posts page



The Posts page does not include any text at the beginning, as its main role is to display all posts published on the blog. It will also feature a delete post functionality below the post list, and a navigation menu.

2.3 Interaction page



The Interaction page is the part of the web application that allows the user to create a new post or update an existing one. These two options will be placed side by side due to their similarity and for the sake of layout consistency with the Posts page.

3 Implementation

The website has been built with the following technologies: Node.js for the server configuration, Express framework to create a web application skeleton and MongoDB for data storage.

3.1 Routing: index.js

```
// GET the home page
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Home' });
});

// GET the posts page
router.get('/posts', posts.findAll);

// DELETE a post
router.post('/uposts', posts.deleteOne);

// UPDATE a post
router.post('/upposts', posts.update);

// GET the add post site
router.get('/postform', function(req, res, next) {
  res.render('addpost', { title: 'Posts' });
});

//create a new post
router.post('/posts', posts.create);
```

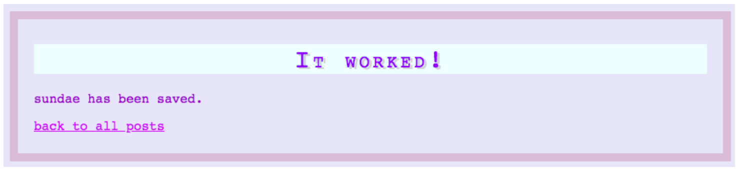
The code above presents the contents of the file routes/index.js, which contains the API routing information (3) (5). The HTTP verbs used are GET for fetching a webpage and POST for supplying user input.

3.2 Database handling: post.controller.js

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

const MongoClient = require('mongodb').MongoClient, assert = require('assert');
//const MONGO_URL = <insert valid mongodb URL>

exports.create = (req, res) => {
 MongoClient.connect(MONGO_URL, (err,db) => {
   if (err) {
     return console.log(err);
   };
   db.collection('posts').insertOne(
     {
       title: req.body.post_title || "Untitled Post",
       text: req.body.post_content
     },
     function (err, res) {
       if (err) {
         db.close();
         return console.log(err);
       }

       db.close();
     }
   );
   return res.render('postchanged', {result: "worked", title: req.body.post_title, action:"saved."})
26    })
27
28   };
29
30   // Retrieve and return all posts from the database.
31   exports.findAll = (req, res) => {
32     MongoClient.connect(MONGO_URL, (err,db) => {
33       if (err) {
34         return console.log(err);
35       };
36       db.collection('posts').find().toArray(function(err, data) {
37         return res.render('posts', { posts: data });
38     });
39
40
41   db.close();
42   })
43   };
44
45
46   exports.deleteOne = (req, res) => {
47     MongoClient.connect(MONGO_URL, (err,db) => {
48       if (err) {
49         return console.log(err);
50       };
51
52       db.collection('posts').deleteOne({title: req.body.post_title}, function(err, obj) {
53       if (err) throw err;
54       if (obj.result.n == 1) {
55         return res.render('postchanged', {result: "worked", title: req.body.post_title, action:"deleted."});
56       } else {
57         return res.render('postchanged', {result: "didn't work", title: req.body.post_title, action:"lost in
             the universe :( Are you sure it exists? "});
58       };
59       db.close();
60     });
61   });
62   };
63
64   exports.update = (req, res) => {
65     MongoClient.connect(MONGO_URL, (err,db) => {
66       if (err) {
67         return console.log(err);
68       };
69       db.collection('posts').update({title: req.body.post_title}, {$set:{text: req.body.post_content}},
           function(err, obj) {
70       if (err) throw err;
71       if (obj.result.n == 1) {
72         return res.render('postchanged', {result: "worked", title: req.body.post_title, action:"updated."});
73       } else {
74         return res.render('postchanged', {result: "didn't work", title: req.body.post_title, action:"lost in
             the universe :( Are you sure it exists?"});
75       };
76       db.close();
77     });
78     });
79   };
```

The code above showcases four functions that have been implemented to provide the CRUD (Create, Read, Update, Delete) functionality to the application: create, findAll, deleteOne, update. They make use of the following db.collection methods(6): `insertOne()`, `find()`, `deleteOne()`, `update()`. After performing the required database operation, the functions render a `postchanged` view with appropriate result, title and action taken to confirm to the user that the required operation has completed successfully. This can be observed in lines 25, 57 and 74 in the above code listing. Below you can find examples of feedback given to the user on their actions.

- a successful post creation:

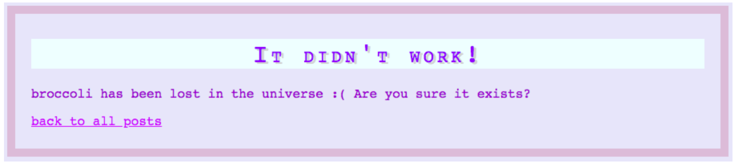


It worked!

sundae has been saved.

back to all posts

- trying to delete a nonexisting post:



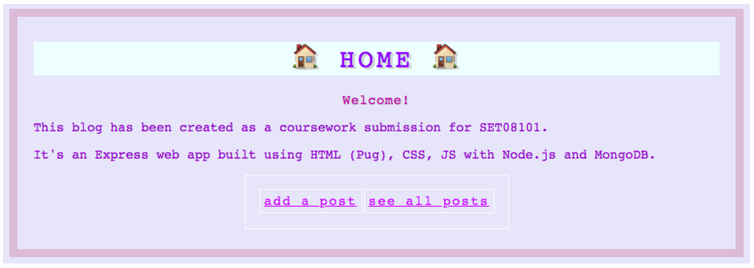
It didn't work!

broccoli has been lost in the universe :(Are you sure it exists?

back to all posts

3.3 Design

The site has been designed to be as simple and intuitive to use as possible. The colour palette chosen is a mix of light pastel colours such as lavender, rose and azure. The main font used is Courier, as it fits the retro feeling of the design. For additional styling, emojis were added, for example on the home page:



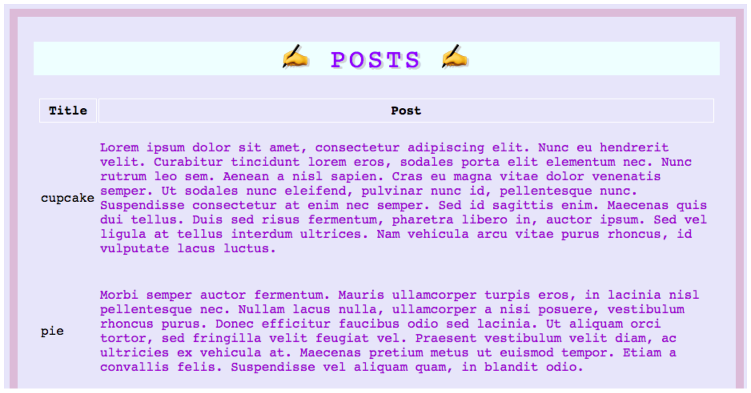
🏠 HOME 🏠

Welcome!

This blog has been created as a coursework submission for SET08101.

It's an Express web app built using HTML (Pug), CSS, JS with Node.js and MongoDB.

The Posts page displays posts in a table, with each post being a row with two cells - one for post title, one for post content.



✍ POSTS ✍

Title	Post
cupcake	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc eu hendrerit velit. Curabitur tincidunt lorem eros, sodales porta elit elementum nec. Nunc rutrum leo sem. Aenean a nisl sapien. Cras eu magna vitae dolor venenatis semper. Ut sodales nunc eleifend, pulvinar nunc id, pellentesque nunc. Suspendisse consectetur at enim nec semper. Sed id sagittis enim. Maecenas quis dui tellus. Duis sed risus fermentum, pharetra libero in, auctor ipsum. Sed vel ligula at tellus interdum ultrices. Nam vehicula arcu vitae purus rhoncus, id vulputate lacus luctus.
pie	Morbi semper auctor fermentum. Mauris ullamcorper turpis eros, in lacinia nisl pellentesque nec. Nullam lacus nulla, ullamcorper a nisi posuere, vestibulum rhoncus purus. Donec efficitur faucibus odio sed lacinia. Ut aliquam orci tortor, sed fringilla velit feugiat vel. Praesent vestibulum velit diam, ac ultricies ex vehicula at. Maecenas pretium metus ut euismod tempor. Etiam a convallis felis. Suspendisse vel aliquam quam, in blandit odio.

4 Critical Evaluation & Future Improvements

The resulting web application provides all four required functionalities. All pages in the project share an aestethically pleasing, consistent design. The navigation provided allows the user to navigate freely between all pages. However, during the work on this web application, a number of possible improvements and possible additional functionalities were noted:

- enrichment of the Post schema - could add fields such as date and time of creation, date and time of modification, author (when support for user accounts is added), etc.
- as mentioned above - create user accounts functionality
- adding comments both as a logged in user and anonymous
- share posts on social media
- improving security by sanitising user input

5 Personal evaluation

This project has enormously improved my understanding of the process and technologies involved in web development, including routing, which I previously found challenging to understand. The research required for its completion allowed me to dive into the intricacies of server-side programming which I have not had any experience with before. Choosing MongoDB as a data storage helped me refresh the knowledge of databases and showed a way to apply what I knew theoretically in a real-life, operational environment. As the project has many points which could be built upon as listed in the previous section, I am now very keen to continue the work on this application and possibly using it as my personal blog in the future.

References

[1] *MongoDB documentation*
`https://docs.mongodb.com/manual/`

[2] *Pug language reference*
`https://pugjs.org/api/getting-started.html`

[3] *Express.js official guide*
`https://expressjs.com/en/guide/routing.html`

[4] *Microsoft Azure API design guidance*
`https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design`

[5] *Hackernoon Restful API Design Guidelines*
`https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9`

[6] *MongoDB documetation - collection methods*
`https://docs.mongodb.com/manual/reference/method/js-collection/`

[7] *The CalliCoder tutorial*
`https://www.callicoder.com/node-js-express-mongodb-restful-crud-api-tutorial/`

[8] *MDN web docs*
`https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction`

[9] *JSON documentation*
`http://json.org/`