

MacArthur Museum: ARCHIVISE

...

Sarah Aldrich Rory Bell Amber Emeny

Purpose

- ★ Museum w/ small budget
- ★ Consulted with them in the past
- ★ How to store their digital photos?
- ★ If Janel can do it in a week, can we do it in three?



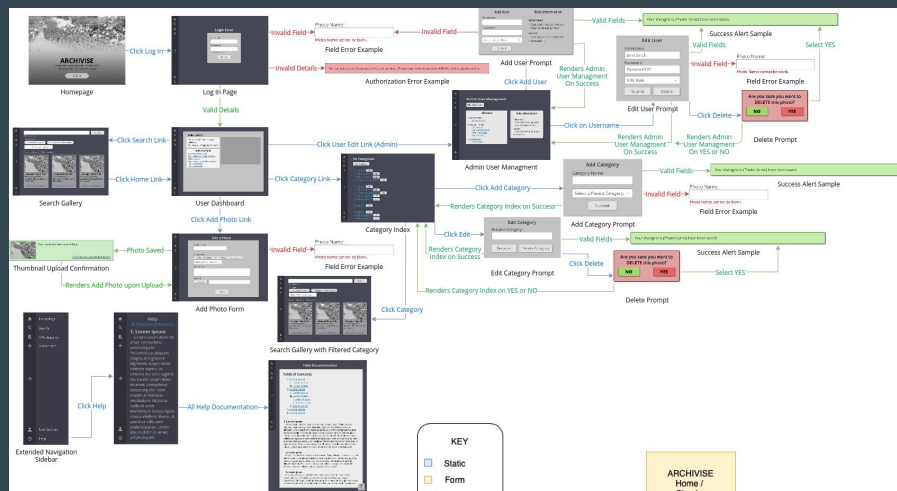
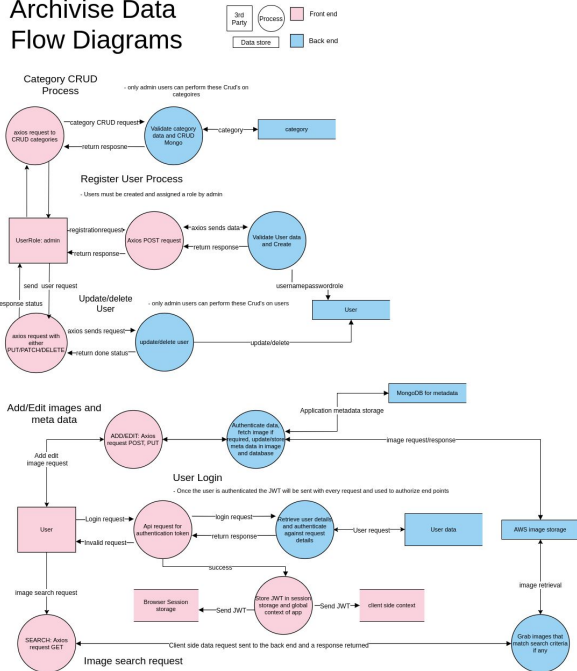
Project Planning

- ★ Yay A-Team!
- ★ Familiar tools give way to Jira & Confluence
- ★ Meeting with the Client
- ★ Detailed Wireframes

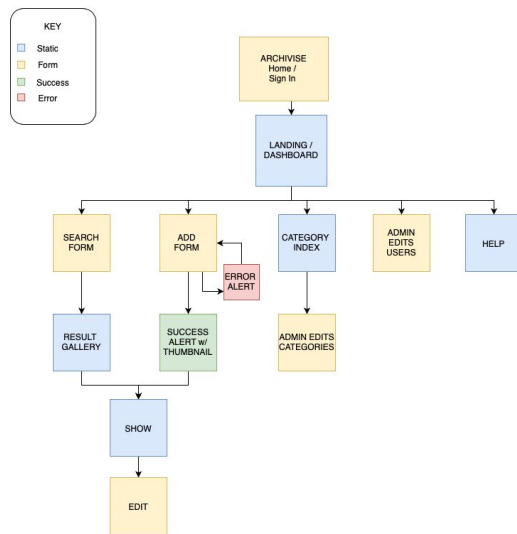
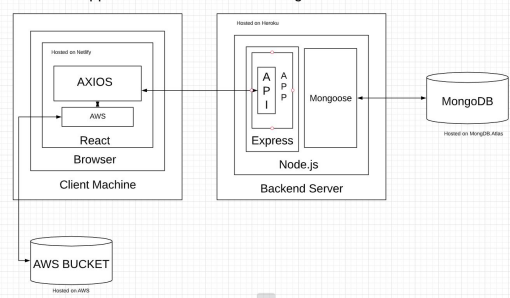


DFD / Architecture

Archivise Data Flow Diagrams



Archivise Application Architecture Drawing



Wireframes



Home Page



Search



All Categories



Add an Item



User Settings



Help



{Photo Name}



Description: This is a short description of the photo describes what's happening.

Item Location: This is a field that lists the physical location of the photo.

Categories: Category 1

Sub-category 1

Sub-category 2

Sub-sub-category 1

Category 2

Sub-category 3

Editing {Photo Name}

Photo Name:

Categories:

× Selected Category × Category > Sub-category

Please Select a Category: ▾

Location:

Description:

Submit

Edit Information

Delete Photo

Edit Category

Rename Category:

Rename

Delete Category



Search

Add Filters

Categories:

× Selected Category

× Category > Sub-category

Please Select a Category: ▾

Search Results (36 results)



Example Photo Title

This an example photo description that describes the photo above.

[This is a link to the photo](#)



Example Photo Title

This an example photo description that describes the photo above.

[This is a link to the photo](#)



Example Photo Title

This an example photo description that describes the photo above.

[This is a link to the photo](#)

authController.js ×

controllers > authController.js > authenticateUser > userModel.findOne() callback > errorMessage

```
1 const userModel = require('../database/schemas/userSchema')
2 const bcrypt = require('bcrypt')
3 const JWTService = require('../servicesHelpers/JWTgenerator')
4
5 // User authentication function
6 const authenticateUser = async (req, res) => {
7   let { username, password } = req.body
8   userModel.findOne({ username: `${username}` }, (err,
9     if (err) {
10       res.send(err.name)
11     }
12     if (!user) {
13       res.status(401).json({ errorMessage: 'That u
14     }
15
16     if (user) {
17       bcrypt.compare(password, user.password, asyn
18         if (err) {
19           res.status(500).json({ message: 'Som
20         }
21         if (auth === false) {
22           res
23             .status(401)
24             .json({ errorMessage: 'Please pr
25         } else if (auth === true) {
26           let token = await JWTService.generat
27           res.status(200).json({ token: token
28         }
29       })
30     }
```

usersController.js ×

controllers > usersController.js > ...

```
36 const createUser = (req, res) => {
37   // Destructure the username, password and role from req.body
38   let { username, password, role } = req.body
39
40   // create a user instance with the destructured params.
41   let user = new userModel({
42     username,
43     password,
44     role
45   })
46
47   // manually validate to ensure that the password meets the validations before hashing the pas
48   user.validate(function(err) {
49     // assign the error message depending on which error it is. i.e username, password or rol
50     if (err) {
51       let path = Object.keys(err.errors)[0]
52       let message = err.errors[path].message
53       res.status(400).json(message)
54       // If there is no error then hash the password with bcrypt and assign the hash to t
55       // before saving.
56     } else {
57       bcrypt.hash(user.password, 10, function(err, hash) {
58         if (err) {
59           // If the password doesnt hash then throw an internal server error
60           res.status(500).send()
61           // If hash is successful then assign the new hashed password and save
62         } else {
63           user.password = hash
64           user.save(function(err) {
65             if (!err) {
66               res.status(201).json('User successfully created!')
67               // there shouldnt be any errors but to avoid problems
68               // send a 500 if save errors occur
69             } else {
70               res.status(500).json('something went wrong')
71             }
72           })
73         }
```

EXPLORER

> OPEN EDITORS

✓ MERN-ASSESSMENT-BAC...

> config

> controllers

authController.js

usersController.js

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

Category one is saved!
✓ should be unique (94ms)
Parent validation
category one saved!
✓ only accepts existing categories as a parent

Photo schema tests
Connected to archive-test
photo name validation
✓ should be required
✓ should be a string
idNumber validation
✓ should be required
✓ should confirm photo ID # is a string
category validation
✓ should be required
✓ should be an array
location validation
✓ should be a string
description validation
✓ should be a string
✓ should be required

User schema Tests
username validation
✓ should return a custom error message
✓ should be required
✓ should confirm username is a string
✓ should be over 3 characters long
✓ should return a custom error message if not over 3 characters
✓ should not contain spaces & return a custom message
✓ Should be unique
password validation
✓ should be required
✓ should return a custom error message
✓ should be at least 6 characters
✓ should have at least one number and 1 special character
role validation
✓ should be required
✓ should be either admin, volunteer or guest

Server Tests
✓ welcomes user to the api

57 passing (3s)

BACKEND CODE...


```

41
42 function findChildren(current, tag, data, formlist) {
43   console.log('this is the current category: ${current}')
44   tag.push(current)
45   if (data.find(element => element.parent === current)) {
46     let newCategory = data.find(element => element.parent === current)
47     console.log('this is the newCategory: ${newCategory}')
48     if (formlist.includes(newCategory.name)) {
49       findChildren(newCategory.name, tag, data, formlist)
50     }
51   }
52   console.log('this is the tag before join: ${tag}')
53   let stringtag = tag.join(" > ")
54   console.log('this is the stringtag: ${stringtag}')
55   let stringtaglist = [...tags]
56   console.log('This should be existing tags: ${stringtaglist}')
57   tag.forEach(element => {
58     let matches = stringtaglist.filter(x => x.includes(element))
59     console.log('This matches the string: ${matches}')
60     let index = stringtaglist.indexOf(matches[0])
61     console.log('stringtaglist: ${stringtaglist}')
62     console.log('index: ${index}')
63     if (matches.length) stringtaglist.splice(index, 1)
64   })
65   stringtaglist.unshift(stringtag)
66   console.log('this should only have unique strings: ${stringtaglist}')
67   setTags(stringtaglist.filter(e => e))
68
69 function setTag(formlist, data) {
70   // for each element in formlist
71   let tag = []
72   let element = formlist[0]
73   findChildren(element, tag, data, formlist)
74   console.log('this is the new formlist: ${formlist}')
75 }
76
77 function populateTags(data) {
78   let formlist = [...formCategories]
79   console.log('the formlist for populatetags: ${formlist}')
80   setTag(formlist, data)
81 }
82
83 function findParent(category, formcategories) {
84   let parent = category.parent
85

```

page.js categories.page.js x addCategoryForm.component.js

categories.page.js > Categories

, [])

seEffect(() => {

if (data) {

let options = []

// Populates a string that is used for

// Passes data as an array of objects,

function populateList(data, parent = 'A

// assigns x as all items that have

const x = data.filter(item => item.

// if the parent is "All", push the

if (parent === 'All') {

x.forEach((element, index) => {

options.push(`\${element.name

populateList(data, element.

})

// This will run if the parent

} else {

// For each element that isn't

x.forEach((element, index) => {

let space = '\u00A0\u00A0'

for (let i = 0; i < level; i++) {

space += '\u00A0\u00A0'

}

// Push the element name with the appropriate indentation.

options.push(`\${space}\${element.name}`)

// Call the function again but pass in the element's name

populateList(data, element.name, level + 1)

})

dashboard.page.js

editUserForm.component.js x

addCategoryForm.component.js

src > components > forms > editUserForm.component.js > ...

1 import React, { useEffect, useState } from "react";

2 import "../../styles/components/forms/loginForm.style.scss";

3 import "../../styles/components/forms/categoryForm.style.scss";

4 import "../../styles/components/forms/userForm.style.scss";

5

6 import SubmitButton from "../buttons/standard_button.component";

7 import API from "../../axios.config";

8

9 export default function EditUserForm(props) {

10 const [errorMessage, setErrorMessage] = useState("");

11 useEffect(() => {});

12

13 const UpdateUser = event => {

14 event.preventDefault();

15 API.put(`/user/\${props.currentUser}`, {

16 username: event.target.username.value,

17 password: event.target.password.value,

18 role: event.target.role.value

19 })

20 .then(res => CloseAndPrompt(res))

21 .catch(err => setErrorMessage(err.response.data));

22 };

23

24 const DeleteUser = event => {

25 event.preventDefault();

26 API.delete(`/user/\${props.currentUser}`)

27 .then(res => CloseAndPrompt(res))

28 .catch(err => console.log(err.response.data.errorMessage));

29 };

30

31 const CloseAndPrompt = (res) => {

32 props.setConfirmPrompt(res.data.message)

33 props.Close()

34 }

35

36 return (

37 <div id="editUserFormContainer" className="formContainer">

38

39

40

41

FRONTEND CODE...

<archive.netlify.com>

LESSONS LEARNED:

ENJOYED:

- ★ Learning team Git
- ★ Drawing off each others strengths
- ★ Learning to successfully TDD
- ★ Breakthroughs in logic
- ★ Finishing

CHALLENGES:

- ★ Learning team Git
- ★ Working with a client and evolving ideas
- ★ Learning to successfully TDD
- ★ Self-imposed high expectations

THANKS!

