Puzzle Hoster

Eoin Laffan Downes

N00210357

Supervisor: Sue Reardon

Second Reader: Cyril Connolly

Year 4 2024/25

DL836 BSc (Hons) in Creative Computing

Abstract

The goal of this project was to create an application that would allow its users to create and share multiple different types of puzzles. The types of puzzles that this application allows its users to create are wordsearches and crosswords. Users are then able to communicate with each other both individually and with comments at the bottom of a puzzle. These communications can have images or just be text. Areas where the application could be improved would include the implementation of more puzzles such as sudokus and being linked to a database with a higher threshold limit than the free MongoDB database that was used.

Acknowledgements

Sue Reardon my supervisor

Cyril Connolly my Second Reader

And my family and friends who participated in the user testing.

The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism.

If you have received significant help with a solution from one or more colleagues, you should document this in your submitted work and if you have any doubt as to what level of discussion/collaboration is acceptable, you should consult your lecturer or the Course Director.

WARNING: Take care when discarding program listings lest they be copied by someone else, which may well bring you under suspicion. Do not to leave copies of your own files on a hard disk where they can be accessed by other. Be aware that removable media, used to transfer work, may also be removed and/or copied by others if left unattended.

Plagiarism is considered to be an act of fraudulence and an offence against Institute discipline.

Alleged plagiarism will be investigated and dealt with appropriately by the Institute. Please refer to the Institute Handbook for further details of penalties.

The following is an extract from the B.Sc. in Creative Computing (Hons) course handbook. Please read carefully and sign the declaration below

Collusion may be defined as more than one person working on an individual assessment. This would include jointly developed solutions as well as one individual giving a solution to another who then makes some changes and hands it up as their own work.

DECLARATION:

I am aware of the Institute's policy on plagiarism and certify that this thesis is my own work.

Student: N00210357

Signed Eoin Laffan Downes

Failure to complete and submit this form may lead to an investigation into your work.

1 Table of Contents

2 Introduction

The objective of this project was to create an application that will allow users to create, edit and upload multiple different types of puzzles, examples of which would include crosswords or word searches. Users would then be able to attempt and comment on each other's puzzles. This will mean that the application will be designed as a hub that will allow for users to inspire and improve their creation and solving skills with each other.

The technologies that were used during the development of this project are

- Visual studio code (A development environment that allows for coding in multiple languages on several operating systems. As such most of the coding will be done within it).
  https://code.visualstudio.com/

- MongoDB (A NoSQL database that uses JSON. This will be used as the application database).
  https://www.mongodb.com/

- Vercel (This is a company that for this application will be used to host the backend on the cloud).
  The hosted backend

- Netlify (This is a company that for this application will be used to host the frontend on the cloud).
  The application

- Amazon web services (This will used to store the application's images in the cloud using Amazons simple storage (S3)).
  https://aws.amazon.com/

- Insomnia rest API (An application used to test the functionality of other applications)
  https://insomnia.rest/

- Microsoft Forms (Used to create and host surveys for user testing)
  The first survey

When it came to project management I used:

- GitHub (Cloud storage site for storing and spreading code. It will store the frontend and backend codes)

  Backend's Github

  Frontend's Github

- Miro (A work board that was used as an idea board and to document work done)

  The miro board

- Figma (An application which focuses on User Interface design)

  The figma page

3 Research

3.1 Introduction

There are three areas of research done for this project topics (looking through research papers about the general topics), functionality (research into the code that runs the application) and UI design (the look of the application).

3.2 Topics

The research was covered mostly in the Reseach report which was split into three categories these being the research going into user generated content (UGC) and the other being the design and detail of puzzles.

UGC Reseach

The UGC research was split into general information, how users are encouraged to make UGCs and how UGCs influence other users. Through this I learned the UGCs count as anything on an application made by its userbase as opposed to its developers / owners.

The first question to do with UGCs was why application owners would want UGC, a simple reason as to why was because it is often the easiest and cheapest way in which to keep new content coming to the site. Another reason might be the fact that a lot of the most popular sites on the internet rely almost entirely on UGCs to produce the majority of their content e.g. YouTube (videos) and Reddit (posit / discussions).

When it comes to encouraging user testing, my research found that the biggest encourager is generally agreed to be the media that the user was consuming. This comes in the form of correcting misinformation e.g. a nature documentary says an animal has the wrong number of toes or users might make a UGC about their love of a piece of media e.g. a user might post a drawing based on a scene from a book.

After media my research suggests that community is another highly encouraging when it comes to getting users to create UGCs. This often comes out through the form of mimicry and or comparison, e.g. a weight loss photo may encourage other users to post their weight loss photo. Communities can also encourage UGCs through rivalry e.g. a post about a steak house may provoke a vegan to respond.

Another requirement to what can encourage users to make UGCs was their own psychology or attitude some examples of this would be if a user was personally and or emotionally invested themselves into something will increase how likely they would be encouraged to make UGC defending or defaming it. Finally, when it comes to encouraging users to make UGCs often it is from the influences of other UGCs.

Now for how UGCs influence users, this can be split into two categories. UGCs are created to intentionally influence other users and those that are not. An example of a UGC created to intentionally influence other users would be a how to guide while an example of an unintentional influence would be a user showing a picture of their pet influence another user to get a pet. The rest of the research done into UGCs influencing users was done with how it encourages users to do so within the same application which as mentioned previously leads to the creation of more UGCs.

Puzzle Research

The puzzle research is split into four categories these being what was puzzle, what defines a good puzzle, how a puzzle catches people's attention and how puzzle improve learning capabilities.

In terms of being in an online environment a puzzle at its most basic components can be described as a problem that has a planned or multiple planned solution designed to be solved. An example of puzzles like this are sudokus (a 9 by 9 grid where the puzzle solver tries to insert numbers into the grid where no number horizontally or vertically aligns with a copy of itself).

Puzzles however can be made more complicated by making through the implementation of more steps to solve the puzzle (e.g. you need two keys to open chest) or they increase the amount of tools use to complete the puzzle (e.g. you would need to solve a mathematical equation to get the number of a wordsearch key).

What makes a good puzzle is a very opinionated topic as some people will prefer short but hard puzzles while others may want a long but simple puzzle while some would like several short puzzles and other would rather try to solve a long puzzle.

Then there are does who enjoy perception puzzle games like work search (e.g. a puzzle about finding words that are hidden amongst random letters), cross words (e.g. a puzzle about trying to fill in a word while only knowing its description and length) and scrabble (e.g. a puzzle game about trying to make words using limit letters).

Then there is an attention-grabbing puzzle. The research in this category was covered in two sections those being getting people to try the puzzle and keeping people's attention while they attempt the puzzle. The first is often achieved by reviews, word of mouth and traditional advertising.

As for keeping people's attention, the research I did suggests that this is mostly based on the quality of the puzzle and if the solver likes that type of puzzle.

Finally, there was the learning capabilities. This is useful as it not only aids in a person's ability to learn and remember a topic a puzzle is about but can also be used to make ever difficult puzzles.

Misc

This section is just here to reference topic research done that was not included in Reseach report. Found here research links (miro). This includes two app research links, two storage links, three control links, five UI design links, three hosting links and the rest are UGC or puzzle links.

3.3 Functionality

General react.js aid

https://www.youtube.com/playlist?list=PLC3y8-rFHvwgg3vaYJgHGnModB54rxOk3

Bootstrap used for application spacing, look and some functionality

https://getbootstrap.com/

Eclipse Crossword. An example of a puzzle creator
https://apps.microsoft.com/detail/9wzdncrfj9kp?hl=en-US&gl=US

Word Search Maker An example of a puzzle creator
https://play.google.com/store/apps/details?id=com.puzzlopolis.wordsearch.maker&hl=en_IE

Geometry dash level editor guide. This can be used to demonstrate how a user creation system should function and what features it should have.
https://www.robtopgames.com/files/GDEditor.pdf

Slideshow assets

https://www.w3schools.com/howto/howto_js_slideshow.asp

3.4 UI design

A list of the UI asset

The application UI assets

The Figma page

https://www.figma.com/design/AckQR4krWgZQvYWG4Zwebu/Untitled?node-id=0-1

Reddit for page layout guide

https://www.reddit.com/

Were the loadings spinner icon was grabbed from.

https://getbootstrap.com/docs/5.3/components/spinners/

Requirements

3.5 Introduction

My application was created with the idea of solving the lack of a universal platform for the creation and sharing of puzzles amongst users. While there are puzzle creators that allow users to share their creations, a lot of them are only for a single type of puzzle (e.g. it's only a crossword maker or a word search creator) not for multicable different types of puzzles.

As such my application was built and designed with the idea of looking and function much like a social media (like for example https://www.reddit.com/ , https://x.com/ or https://www.facebook.com/) for puzzles as appose to being about sharing users sharing their messages or photos. However, the application would probably still require users to be able to communicate with each other.

However, while these types of sites only cover the overall purpose and functionality of the application but not the creation or solving of the puzzles. Just I started looking at the individual puzzle creators to figure out each puzzle's creation requirements.

3.6 Requirements gathering

Reddit

www.Reddit.com

Screen shots

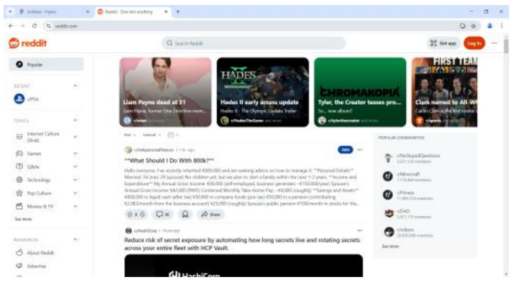


Fig 1

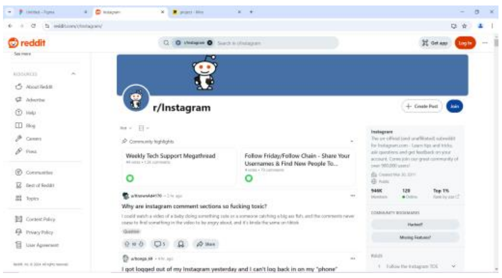


Fig 2

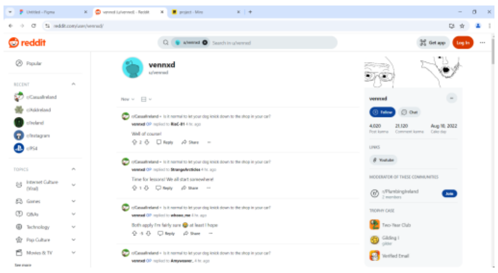


Fig 3

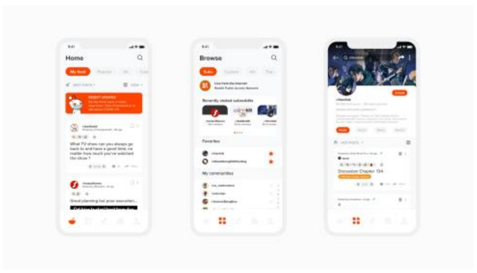


Fig 4

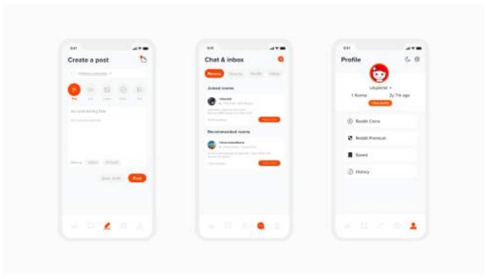


Fig 5

Description

Reddit is an American social media platform that focuses on its users sharing posts in the form of photo, videos and messages. With about 100 million users and 2000 employees.

Advantages

The advantages of Reddit as a source of data gathering for the applications requirements if is in how it displays, sorts and presents user messages.

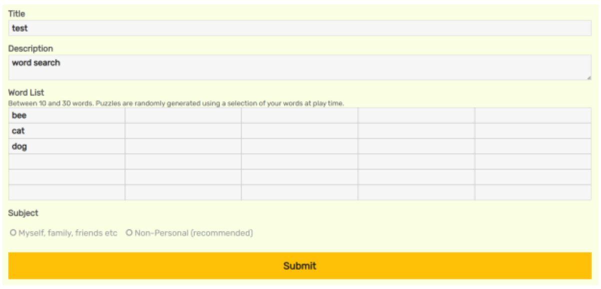
Disadvantages

Reddit being only a message based social media means that it is not useful when it comes to the look and design of the puzzles and their creation tool.

The wordsearch maker

https://thewordsearch.com/maker/

Screen shots



Title

test

Description

word search

Word List

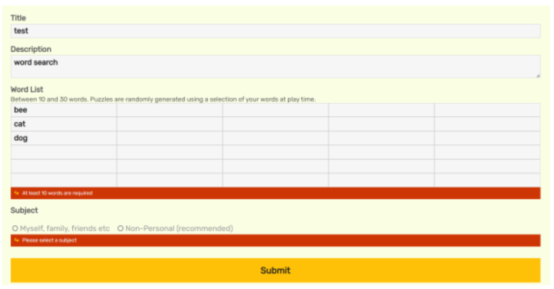
Between 10 and 30 words. Puzzles are randomly generated using a selection of your words at play time.

bee				
cat				
dog				

Subject

○ Myself, family, friends etc ○ Non-Personal (recommended)

Fig 6



Title

test

Description

word search

Word List

Between 10 and 30 words. Puzzles are randomly generated using a selection of your words at play time.

bee				
cat				
dog				

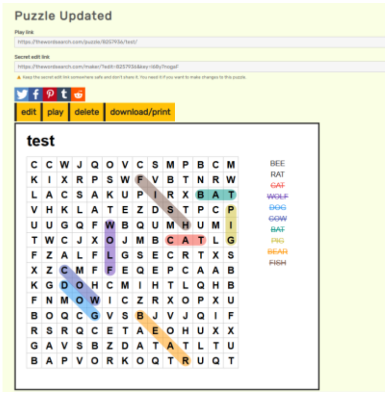
At least 10 words are required

Subject

○ Myself, family, friends etc ○ Non-Personal (recommended)

Please select a subject

Fig 7



Puzzle Updated

Play link

https://thewordsearch.com/puzzle/8257936/test/

Secret edit link

https://thewordsearch.com/maker/?edit=8257936&key=i6By7nogsF

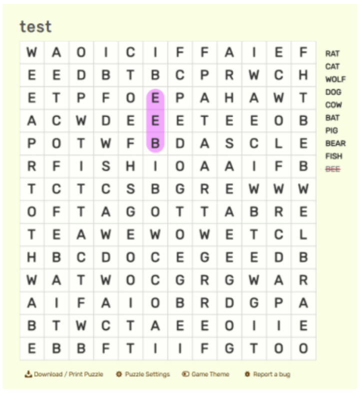
Keep the secret edit link somewhere safe and don't share it. You need it if you want to make changes to this puzzle.

test

```
C C W J Q O V C S M P B C M
K I X R P S W F V B T N R W
L A C S A K U P I R X B A T
V H K L A T E Z D S T P C P
U U G Q F W B Q U M H U M I
T W C J X O J M B C A T L G
F Z A L F L G S E C R T X S
X Z C M F F E Q E P C A A B
K G D O H C M I H T L Q H B
F N M O W I C Z R X O P X U
B O Q C G V S B J V J Q I F
R S R Q C E T A E O H U X X
G A V S B Z D A T A T L T U
B A P V O R K O Q T R U Q T
```

BEE
RAT
~~CAT~~
~~WOLF~~
~~DOG~~
~~COW~~
~~BAT~~
~~PIG~~
~~BEAR~~
~~FISH~~

Fig 8



test

```
W A O I C I F F A I E F
E E D B T B C P R W C H
E T P F O E P A H A W T
A C W D E E E T E E O B
P O T W F B D A S C L E
R F I S H I O A A I F B
T C T C S B G R E W W W
O F T A G O T T A B R E
T E A W E W O W E T C L
H B C D O C E G E E D B
W A T W O C G R G W A R
A I F A I O B R D G P A
B T W C T A E E O I I E
E B B F T I I F G T O O
```

RAT
CAT
WOLF
DOG
COW
BAT
PIG
BEAR
FISH
~~BEE~~

Fig 9

Description

The wordsearch maker is an online word search creator and hosted that allows users to make, share and download their word search.

Advantages

The advantages of the wordsearch maker are that it shows off the minimum requirements that are needed to make a puzzle creator. It also is good example what I should be included in a freer word search creator then it.

Disadvantages

Its disadvantages are that its UI was designed always creates a word search that was the same size, it limits the number of words that the user gets to have in the goal along with it demanding that that the goal not have less than ten. Also, it does not allow users to decide where to place the goal words within the word search.

Sudoku 9x9

https://www.sudoku9x9.com/dailysudoku/

Screen shots

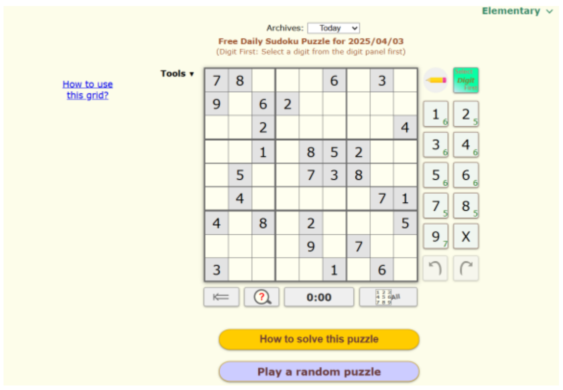
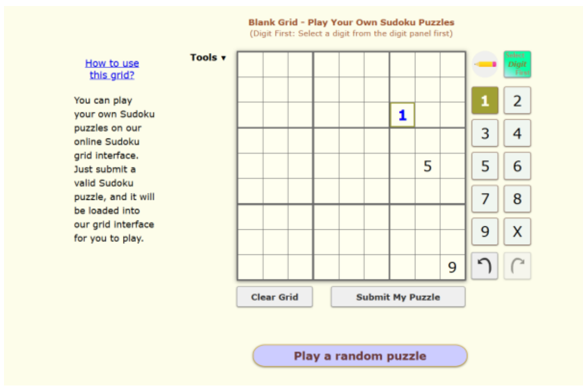


Fig 10



Blank Grid - Play Your Own Sudoku Puzzles
(Digit First: Select a digit from the digit panel first)
How to use this grid?
Tools
Clear Grid
Submit My Puzzle
Play a random puzzle

You can play your own Sudoku puzzles on our online Sudoku grid interface. Just submit a valid Sudoku puzzle, and it will be loaded into our grid interface for you to play.

Fig 11

Description

Sudoku 9x9 is a basic 9 by 9 sudoku creator

Advantages

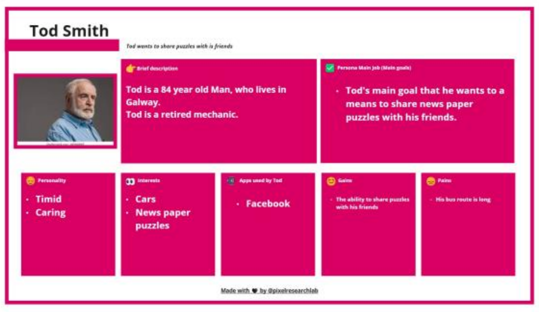
Sudoku 9x9 advantages are that it demonstrates the bare minimum a sudoku maker should have such as a how to solve button and the ability to mark what numbers could be in the grid.

Disadvantages

It only has a basic sudoku size of 9x9 and it has minimum tools to search existing sudokus.

3.7 Requirements modelling

3.7.1 Personas



Tod Smith

Tod wants to share puzzles with is friends

Brief description

Tod is a 84 year old Man, who lives in Galway.
Tod is a retired mechanic.

Persona Main job (Main goals)

- Tod's main goal that he wants to a means to share news paper puzzles with his friends.

Personality

- Timid
- Caring

Interests

- Cars
- News paper puzzles

Apps used by Tod

- Facebook

Gains

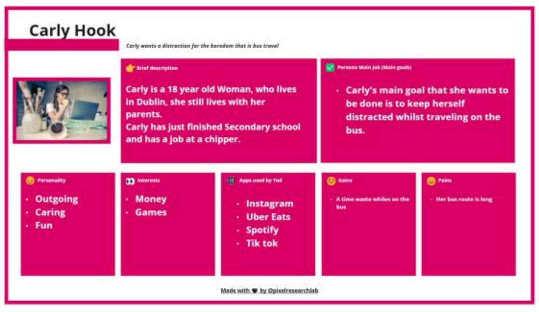
- The ability to share puzzles with his friends

Pains

- His bus route is long

Made with ♥ by @pixelresearchlab

Fig 12



Carly Hook

Carly wants a distraction for the boredom that is bus travel

Brief description

Carly is a 18 year old Woman, who lives in Dublin, she still lives with her parents.
Carly has just finished Secondary school and has a job at a chipper.

Persona Main job (Main goals)

- Carly's main goal that she wants to be done is to keep herself distracted whilst traveling on the bus.

Personality

- Outgoing
- Caring
- Fun

Interests

- Money
- Games

Apps used by Tod

- Instagram
- Uber Eats
- Spotify
- Tik tok

Gains

- A time waste whiles on the bus

Pains

- Her bus route is long

Made with ♥ by @pixelresearchlab

Fig 13

3.7.2 Functional requirements

1. The ability to solve a puzzle (word search).
2. The ability to create a puzzle (word search).
3. The ability to edit and delete a puzzle (word search).
4. A user should be able to register and login to the application.
5. A user should be able to comment on a puzzle.
6. A user can edit and delete their comments.
7. A comment can have an image.
8. Users should be able to reply to a comment.
9. Users should be able to message each other.
10. Messages should be able to be edited and delete be their poster.
11. Users should be able to report bugs.
12. Users should have the ability to solve different types of puzzles (e.g. sudoku).
13. Users should have the ability to make different types of puzzles (e.g. sudoku).
14. Users cand edit and delete their accounts.

3.7.3 Non-functional requirements

1. Account passwords should be encoded.
2. Users email should only be used for one account.
3. Only the create of puzzles, comments and messages should be able to delete them

3.7.4 Use Case Diagrams

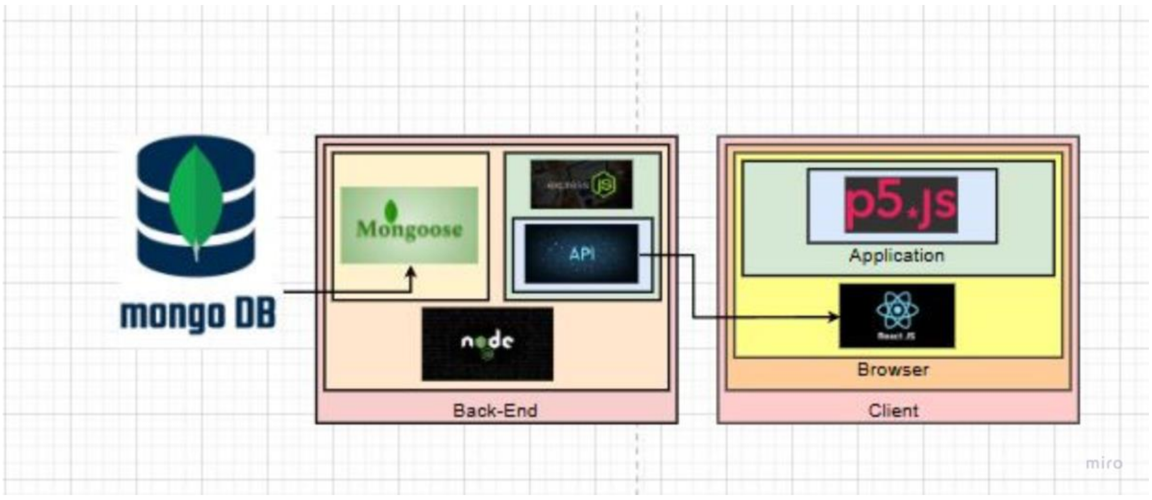


Fig 14

3.8 Feasibility

This was made feasible by first working on the backend, this was started by picking a database system. Due to it being an online cloud database I went with MongoDB. This will be connected to a node.js server using the node module mongoose. Then it will use express.js to make the database data useable as a restful API for the front end.

As for the feasibility of the front end the base of the application was coded in react.js, with bootstrap and CSS bring used for the UI elements. As for the functionality of the puzzles I will use p5.js (possessing https://p5js.org/ , the npm package to get p5 to work with react https://www.npmjs.com/package/react-p5). All this means that most of the project will be coded in JavaScript with spacing and design being done in CSS and html.

3.9 Conclusion

In conclusion, the requirements and feasibility of this project is to create an application that will allow users to create puzzles using p5 as the constantly updating function inside a react.js. Outside of the puzzle functions of the application there will be multiple forms social media features that won't be using p5 (comments, messages, accounts, etc.).

The non puzzle parts (those not built in the p5 functions) of the application will therefore get its functionality and look based off of social media sites. They will have their requirements and functionality created in JavaScript. The frontend gets and sends data to a MongoDB database via a node.js back end.

The way the puzzles will work was by having a string that stores the data as a single string that for example word search with its grids x and y axis would store the placement of the let's like this (0, 0, A 0, 1, b). This will make the grid squares x 0 y 0 equal A and x 0 y 1 equal B.

4 Design

4.1 Introduction

The design of the application can be broken down into two sections. These being the User Interface (UI) design and the program design. For the UI design, I choose to go with a simple mostly black and white look for the application's interactive elements. This is show that it remains simplistic and hopefully intuitive for its users.

As for the program design I will go about in multiple steps. Step one being two get the base back-ends functionality running and hosted. Step two will be getting the basic of the front end and a single puzzle functioning. The third step will then be getting some users to test it application. The next step will be addressing what the tests brought up. Then I implemented the more detailed UI. The step afterwards I added a second puzzle (sudoku). For this point on the plan would mostly become a state of doing user testing and bug fixing.

4.2 Program Design

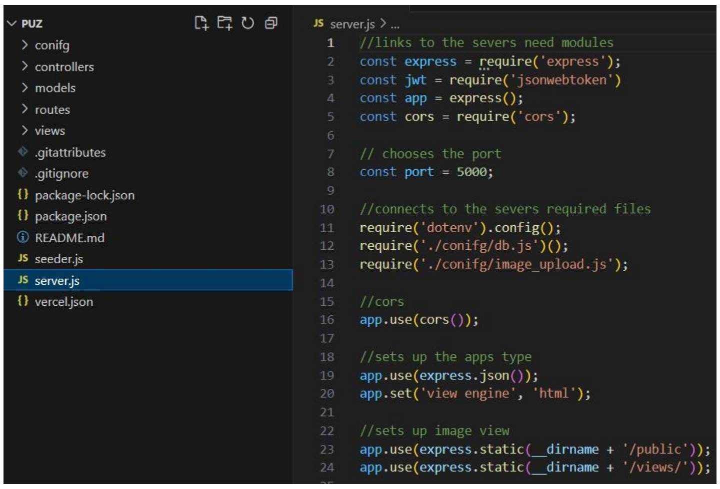
4.2.1 Technologies

The technologies being used to create this application are:

The back end of the Application

- Visual studio code (A development environment that allows for coding in multiple languages on several operating systems. As such most of the coding will be done within it). https://code.visualstudio.com/

  Visual studio code was chosen because of its ability to have a GitHub account connect to it and as such it allows you to upload a repository from within. This along with its allowing for the coding of all languages within this project made its use a simple choice.

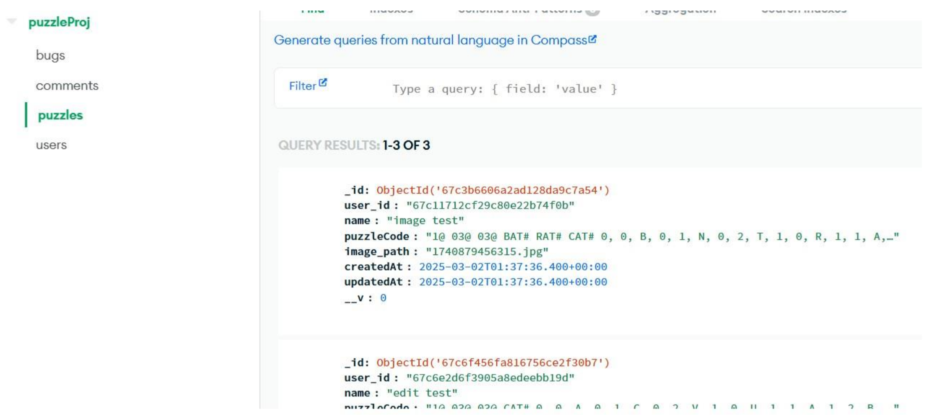


```
//links to the severs need modules
const express = require('express');
const jwt = require('jsonwebtoken')
const app = express();
const cors = require('cors');

// chooses the port
const port = 5000;

//connects to the severs required files
require('dotenv').config();
require('./conifg/db.js')();
require('./conifg/image_upload.js');

//cors
app.use(cors());

//sets up the apps type
app.use(express.json());
app.set('view engine', 'html');

//sets up image view
app.use(express.static(__dirname + '/public'));
app.use(express.static(__dirname + '/views/'));
```

Fig 15

- MongoDB (A NoSQL database that uses JSON. This will be used as the application database). https://www.mongodb.com/

MongoDB was the database system I ended up going with because it is a website and just always accessible. This means that I can more easily get users to test the application remotely which means that I can have a greater number of testers overall.



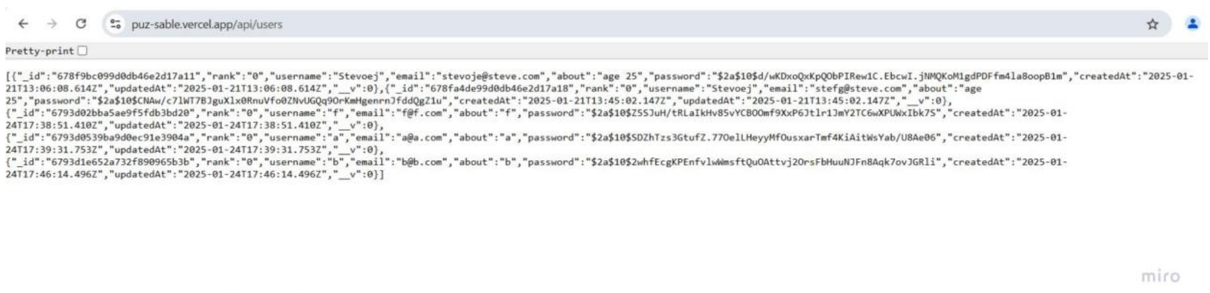
```
Type a query: { field: 'value' }

QUERY RESULTS: 1-3 OF 3

_id: ObjectId('67c3b6606a2ad128da9c7a54')
user_id : "67c11712cf29c80e22b74f0b"
name : "image test"
puzzleCode : "1@ 03@ 03@ BAT# RAT# CAT# 0, 0, B, 0, 1, N, 0, 2, T, 1, 0, R, 1, 1, A,…"
image_path : "1740879456315.jpg"
createdAt : 2025-03-02T01:37:36.400+00:00
updatedAt : 2025-03-02T01:37:36.400+00:00
__v : 0

_id: ObjectId('67c6f456fa816756ce2f30b7')
user_id : "67c6e2d6f3905a8edeebb19d"
name : "edit test"
```

Fig 16

- Vercel (This is a company that for this application will be used to host the backend on the cloud).

  [The hosted backend]

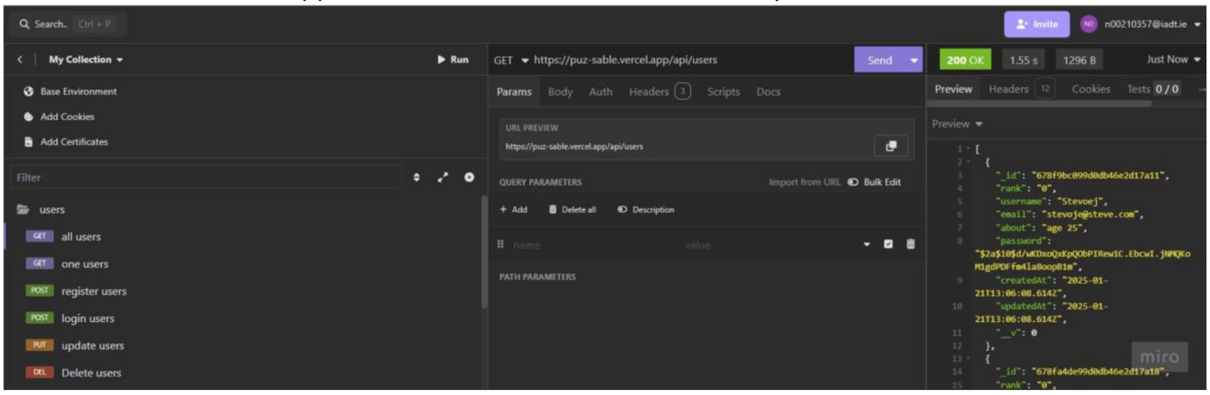
  Vercel was chosen because it was what I already knew.



[{"_id":"678f9bc099d0db46e2d17a11","rank":"0","username":"Stevoej","email":"stevoje@steve.com","about":"age 25","password":"$2a$10$d/wKDxoQxKpQObPIRew1C.EbcwI.jNMQKoM1gdPDFfm4la8oopB1m","createdAt":"2025-01-21T13:06:08.614Z","updatedAt":"2025-01-21T13:06:08.614Z","__v":0},{"_id":"678fa4de99d0db46e2d17a18","rank":"0","username":"Stevoej","email":"stefg@steve.com","about":"age 25","password":"$2a$10$CNAw/c7lWT7BJguXlx0RnuVfo0ZNvUGQq9OrKmHgenrnJfddQgZ1u","createdAt":"2025-01-21T13:45:02.147Z","updatedAt":"2025-01-21T13:45:02.147Z","__v":0},
{"_id":"6793d02bba5ae9f5fdb3bd20","rank":"0","username":"f","email":"f@f.com","about":"f","password":"$2a$10$Z5SJuH/tRLaIkHv85vYCBOOmf9XxP6Jtlr1JmY2TC6wXPUWxIbk7S","createdAt":"2025-01-24T17:38:51.410Z","updatedAt":"2025-01-24T17:38:51.410Z","__v":0},
{"_id":"6793d0539ba9d0ec91e3904a","rank":"0","username":"a","email":"a@a.com","about":"a","password":"$2a$10$SDZhTzs3GtufZ.77OelLHeyyMfOusxarTmf4KiAitWsYab/U8Ae06","createdAt":"2025-01-24T17:39:31.753Z","updatedAt":"2025-01-24T17:39:31.753Z","__v":0},
{"_id":"6793d1e652a732f890965b3b","rank":"0","username":"b","email":"b@b.com","about":"b","password":"$2a$10$2whfEcgKPEnfvlwWmsftQuOAttvj2OrsFbHuuNJFn8Aqk7ovJGRli","createdAt":"2025-01-24T17:46:14.496Z","updatedAt":"2025-01-24T17:46:14.496Z","__v":0}]

Fig 17

- Insomnia rest API (An application used to test the functionality of other applications)
  https://insomnia.rest/
  The insomnia API is the application I used to test the functionality of the backend.



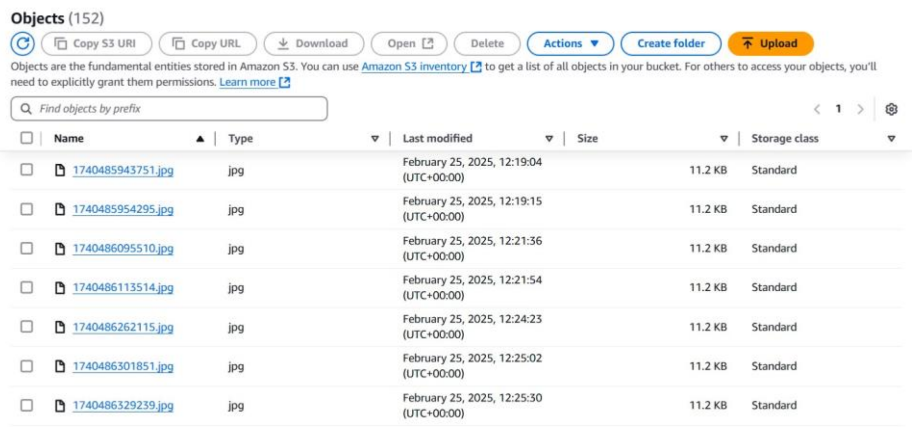
```
[
  {
    "_id": "678f9bc099d0db46e2d17a11",
    "rank": "0",
    "username": "Stevoej",
    "email": "stevoje@steve.com",
    "about": "age 25",
    "password":
"$2a$10$d/wKDxoQxKpQObPIRew1C.EbcwI.jNMQKo
M1gdPDFfm4la8oopB1m",
    "createdAt": "2025-01-
21T13:06:08.614Z",
    "updatedAt": "2025-01-
21T13:06:08.614Z",
    "__v": 0
  },
  {
    "_id": "678fa4de99d0db46e2d17a18",
    "rank": "0",
```

Fig 18

- Amazon web services AWS (This will used to store the application's images in the cloud using Amazons simple storage (S3)).

  https://aws.amazon.com/

  AWS is used purely to hold the applications image files.



Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

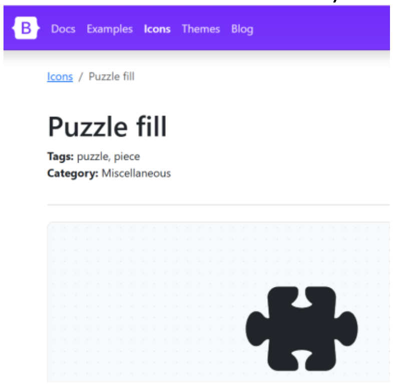
Name	Type	Last modified	Size	Storage class
1740485943751.jpg	jpg	February 25, 2025, 12:19:04 (UTC+00:00)	11.2 KB	Standard
1740485954295.jpg	jpg	February 25, 2025, 12:19:15 (UTC+00:00)	11.2 KB	Standard
1740486095510.jpg	jpg	February 25, 2025, 12:21:36 (UTC+00:00)	11.2 KB	Standard
1740486113514.jpg	jpg	February 25, 2025, 12:21:54 (UTC+00:00)	11.2 KB	Standard
1740486262115.jpg	jpg	February 25, 2025, 12:24:23 (UTC+00:00)	11.2 KB	Standard
1740486301851.jpg	jpg	February 25, 2025, 12:25:02 (UTC+00:00)	11.2 KB	Standard
1740486329239.jpg	jpg	February 25, 2025, 12:25:30 (UTC+00:00)	11.2 KB	Standard

Fig 19

The front end of the Application

- Visual studio code (For the same reasons that it was used in the backend).
  https://code.visualstudio.com/

- Bootstrap (Used in the UI of the application)
  https://getbootstrap.com/
  Bootstrap was used for the spacing (how far away each button is from each other, the curve and thickness of the border) and the icons.



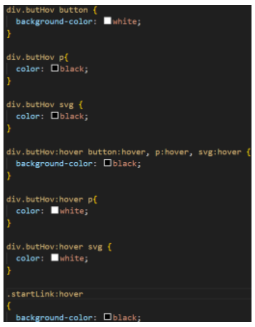
Puzzle fill

Tags: puzzle, piece
Category: Miscellaneous

Fig 20

- CSS / Cascading style sheets and SCSS / Sassy cascading style sheets (Style sheet languages)

  These were used for the applications UI interactions (changing the background and text when hovering over a button and set size limits for objects).



```css
div.butHov button {
  background-color: white;
}

div.butHov p{
  color: black;
}

div.butHov svg {
  color: black;
}

div.butHov:hover button:hover, p:hover, svg:hover {
  background-color: black;
}

div.butHov:hover p{
  color: white;
}

div.butHov:hover svg {
  color: white;
}

.startLink:hover
{
  background-color: black;
```

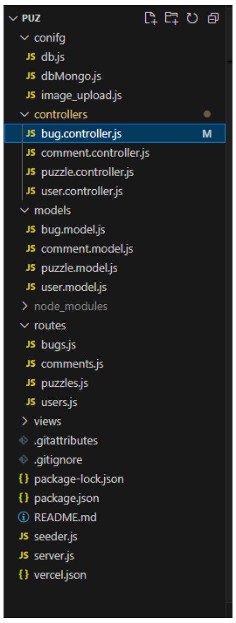
Fig 22

- React (A front-end JavaScript library)
  https://react.dev/
  React makes up the base of the front end of the application. This means it controls the applications rooting and rendering of the page's interfaces.

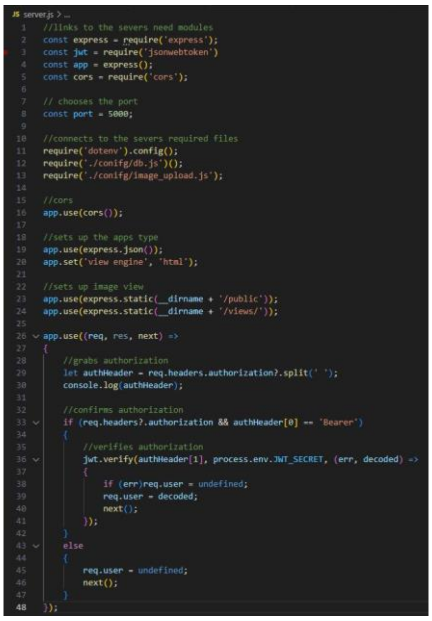
- Netlify (A remote could hosting site)
  Netlify is used to host the front end.

4.2.2 Structure

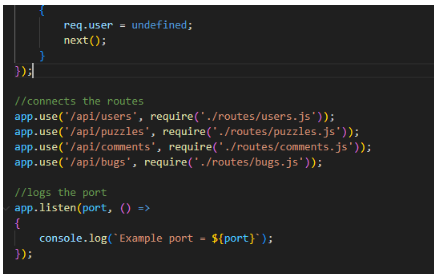
The structure of the back end starts at the sever.js folder. Inside the server folder it set up and links throughout the back-end application. It starts this process by setting up the express app, Cors and the Json token whilst doing this it grabs the all the requires' from config folder (the database code and image uploads code (all three files of which were taken from a previous project)). After that server.js checks for authorization after which it then grabs the routes.



```
PUZ
  conifg
    db.js
    dbMongo.js
    image_upload.js
  controllers
    bug.controller.js        M
    comment.controller.js
    puzzle.controller.js
    user.controller.js
  models
    bug.model.js
    comment.model.js
    puzzle.model.js
    user.model.js
  node_modules
  routes
    bugs.js
    comments.js
    puzzles.js
    users.js
  views
  .gitattributes
  .gitignore
  package-lock.json
  package.json
  README.md
  seeder.js
  server.js
  vercel.json
```



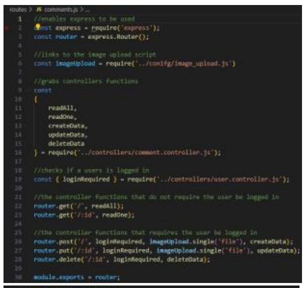
```js
//links to the severs need modules
const express = require('express');
const jwt = require('jsonwebtoken')
const app = express();
const cors = require('cors');

// chooses the port
const port = 5000;

//connects to the severs required files
require('dotenv').config();
require('./conifg/db.js')();
require('./conifg/image_upload.js');

//cors
app.use(cors());

//sets up the apps type
app.use(express.json());
app.set('view engine', 'html');

//sets up image view
app.use(express.static(__dirname + '/public'));
app.use(express.static(__dirname + '/views/'));

app.use((req, res, next) =>
{
    //grabs authorization
    let authHeader = req.headers.authorization?.split(' ');
    console.log(authHeader);

    //confirms authorization
    if (req.headers?.authorization && authHeader[0] == 'Bearer')
    {
        //verifies authorization
        jwt.verify(authHeader[1], process.env.JWT_SECRET, (err, decoded) =>
        {
            if (err)req.user = undefined;
            req.user = decoded;
            next();
        });
    }
    else
    {
        req.user = undefined;
        next();
    }
});
```



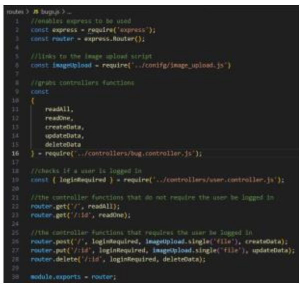
```js
    {
        req.user = undefined;
        next();
    }
});

//connects the routes
app.use('/api/users', require('./routes/users.js'));
app.use('/api/puzzles', require('./routes/puzzles.js'));
app.use('/api/comments', require('./routes/comments.js'));
app.use('/api/bugs', require('./routes/bugs.js'));

//logs the port
app.listen(port, () =>
{
    console.log(`Example port = ${port}`);
});
```

Fig 23 Fig 24 Fig 25

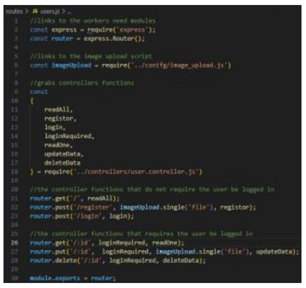
The routes a stored in the routes folder, for the most part these routes are almost identical to each other. They start by grabbing their required components (express, express's router and image upload.), then they create a blank functions' that will be used by that route's controller (read All, read One, create Data (not found on the user route), update Data, delete Data and for the user router only there is register, login and loginRequired). All these functions are then linked to their specific controller. Finally, every router (except user) grabs loginRequired from the user's controller and creates their routes for the sever.



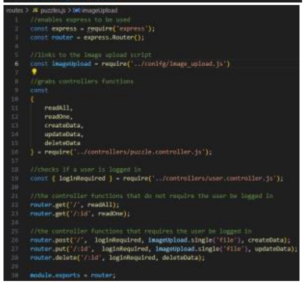
```
routes > comments.js > ...
//enables express to be used
const express = require('express');
const router = express.Router();

//links to the image upload script
const imageUpload = require('../conifg/image_upload.js')

//grabs controllers functions
const
{
    readAll,
    readOne,
    createData,
    updateData,
    deleteData
} = require('../controllers/comment.controller.js');

//checks if a users is logged in
const { loginRequired } = require('../controllers/user.controller.js');

//the controller functions that do not require the user be logged in
router.get('/', readAll);
router.get('/:id', readOne);

//the controller functions that requires the user be logged in
router.post('/', loginRequired, imageUpload.single('file'), createData);
router.put('/:id', loginRequired, imageUpload.single('file'), updateData);
router.delete('/:id', loginRequired, deleteData);

module.exports = router;
```



```
routes > bugs.js > ...
//enables express to be used
const express = require('express');
const router = express.Router();

//links to the image upload script
const imageUpload = require('../conifg/image_upload.js')

//grabs controllers functions
const
{
    readAll,
    readOne,
    createData,
    updateData,
    deleteData
} = require('../controllers/bug.controller.js');

//checks if a user is logged in
const { loginRequired } = require('../controllers/user.controller.js');

//the controller functions that do not require the user be logged in
router.get('/', readAll);
router.get('/:id', readOne);

//the controller functions that requires the user be logged in
router.post('/', loginRequired, imageUpload.single('file'), createData);
router.put('/:id', loginRequired, imageUpload.single('file'), updateData);
router.delete('/:id', loginRequired, deleteData);

module.exports = router;
```



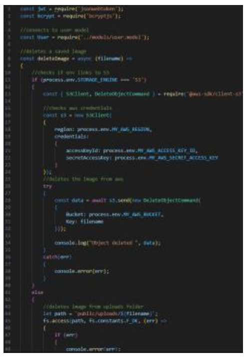
```
routes > users.js > ...
//links to the workers need modules
const express = require('express');
const router = express.Router();

//links to the image upload script
const imageUpload = require('../conifg/image_upload.js')

//grabs controllers functions
const
{
    readAll,
    registor,
    login,
    loginRequired,
    readOne,
    updateData,
    deleteData
} = require('../controllers/user.controller.js')

//the controller functions that do not require the user be logged in
router.get('/', readAll);
router.post('/register', imageUpload.single('file'), registor);
router.post('/login', login);

//the controller functions that requires the user be logged in
router.get('/:id', loginRequired, readOne);
router.put('/:id',  loginRequired, imageUpload.single('file'), updateData);
router.delete('/:id', loginRequired, deleteData);

module.exports = router;
```



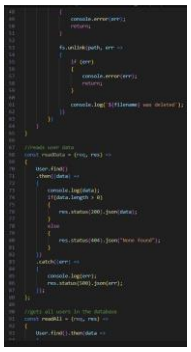
```
routes > puzzles.js > imageUpload
//enables express to be used
const express = require('express');
const router = express.Router();

//links to the image upload script
const imageUpload = require('../conifg/image_upload.js')

//grabs controllers functions
const
{
    readAll,
    readOne,
    createData,
    updateData,
    deleteData
} = require('../controllers/puzzle.controller.js');

//checks if a user is logged in
const { loginRequired } = require('../controllers/user.controller.js');

//the controller functions that do not require the user be logged in
router.get('/', readAll);
router.get('/:id', readOne);

//the controller functions that requires the user be logged in
router.post('/',  loginRequired, imageUpload.single('file'), createData);
router.put('/:id',  loginRequired, imageUpload.single('file'), updateData);
router.delete('/:id', loginRequired, deleteData);

module.exports = router;
```

Fig 26 Fig 27 Fig 28 Fig 29

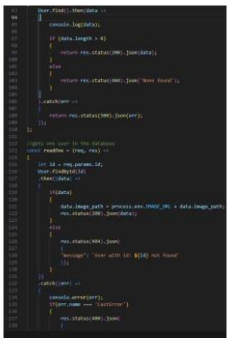
Next comes the controllers. The controllers are modules design to export the functions that each route had set up. For this application the functions for each one were mostly the same as each other. All controllers have delete Image (removes the image files linked to this item from the AWS S3 bucket), read All (shows all a route that exist within the database), read one (shows one a route that exist within the database), create Data (allows for the creates of one of the routes objects [not in user]), update Data (allows for the editing of one of the routes objects) and delete Date (Allows for the object to be deleted from the database). As for the user specific functions, register (users' version of createData with protection for passwords and email), login (how users will login into the front-end) and loginRequired (How the back-end checks if there is a user signed in).



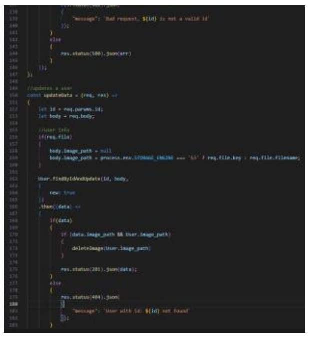
```
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');

//connects to user model
const User = require('../models/user.model');

//deletes a saved image
const deleteImage = async (filename) =>
{
    //checks if env links to S3
    if (process.env.STORAGE_ENGINE === 'S3')
    {
        const { S3Client, DeleteObjectCommand } = require('@aws-sdk/client-s3'

        //checks aws credentials
        const s3 = new S3Client(
        {
            region: process.env.MY_AWS_REGION,
            credentials:
            {
                accessKeyId: process.env.MY_AWS_ACCESS_KEY_ID,
                secretAccessKey: process.env.MY_AWS_SECRET_ACCESS_KEY
            }
        });
        //deletes the image from aws
        try
        {
            const data = await s3.send(new DeleteObjectCommand(
            {
                Bucket: process.env.MY_AWS_BUCKET,
                Key: filename
            }));

            console.log("Object deleted ", data);
        }
        catch(err)
        {
            console.error(err);
        }
    }
    else
    {
        //deletes image from uploads folder
        let path = `public/uploads/${filename}`;
        fs.access(path, fs.constants.F_OK, (err) =>
        {
            if (err)
            {
                console.error(err);
```



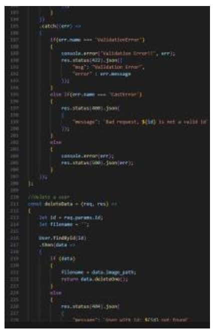
```
            {
                console.error(err);
                return;
            }

            fs.unlink(path, err =>
            {
                if (err)
                {
                    console.error(err);
                    return;
                }

                console.log(`${filename} was deleted`);
            })
        });
    }
}

//reads user data
const readData = (req, res) =>
{
    User.find()
    .then((data) =>
    {
        console.log(data);
        if(data.length > 0)
        {
            res.status(200).json(data);
        }
        else
        {
            res.status(404).json("None found");
        }
    })
    .catch((err) =>
    {
        console.log(err);
        res.status(500).json(err);
    });
};

//gets all users in the database
const readAll = (req, res) =>
{
    User.find().then(data =>
```



```
    User.find().then(data =>
    {
        console.log(data);

        if (data.length > 0)
        {
            return res.status(200).json(data);
        }
        else
        {
            return res.status(404).json('None found');
        }
    }).catch(err =>
    {
        return res.status(500).json(err);
    });
};

//gets one user in the database
const readOne = (req, res) =>
{
    let id = req.params.id;
    User.findById(id)
    .then((data) =>
    {
        if(data)
        {
            data.image_path = process.env.IMAGE_URL + data.image_path;
            res.status(200).json(data);
        }
        else
        {
            res.status(404).json
            ({
            "message": `User with id: ${id} not found`
            });
        }
    })
    .catch((err) =>
    {
        console.error(err);
        if(err.name === 'CastError')
        {
            res.status(400).json
            (
```

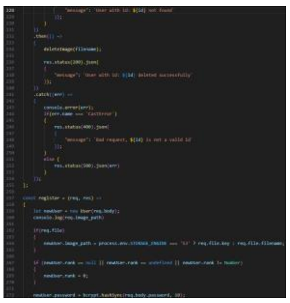


```
            {
                "message": `Bad request, ${id} is not a valid id`
            });
        }
        else
        {
            res.status(500).json(err)
        }
    });
};

//updates a user
const updateData = (req, res) =>
{
    let id = req.params.id;
    let body = req.body;

    //user info
    if(req.file)
    {
        body.image_path = null
        body.image_path = process.env.STORAGE_ENGINE === 'S3' ? req.file.key : req.file.filename;
    }

    User.findByIdAndUpdate(id, body,
    {
        new: true
    })
    .then((data) =>
    {
        if(data)
        {
            if (data.image_path && User.image_path)
            {
                deleteImage(User.image_path)
            }

            res.status(201).json(data);
        }
        else
        {
            res.status(404).json
            ({
                "message": `User with id: ${id} not found`
            });
        }
```

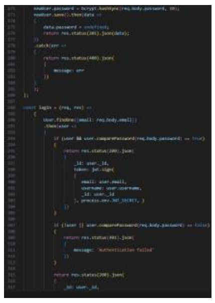


```
        }
    })
    .catch((err) =>
    {
        if(err.name === 'ValidationError')
        {
            console.error('Validation Error!!', err);
            res.status(422).json({
                "msg": "Validation Error",
                "error" : err.message
            });
        }
        else if(err.name === 'CastError')
        {
            res.status(400).json(
            {
                "message": `Bad request, ${id} is not a valid id`
            });
        }
        else
        {
            console.error(err);
            res.status(500).json(err);
        }
    });
};

//delete a user
const deleteData = (req, res) =>
{
    let id = req.params.id;
    let filename = '';

    User.findById(id)
    .then(data =>
    {
        if (data)
        {
            filename = data.image_path;
            return data.deleteOne();
        }
        else
        {
            res.status(404).json
            (
                "message": `User with id: ${id} not found`
```

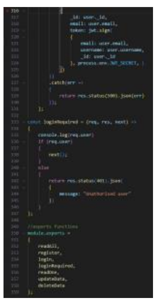
Fig 30 Fig 31 Fig 32 Fig 33 Fig 34



```
                "message": `User with id: ${id} not found`
            });
        }
    })
    .then(() =>
    {
        deleteImage(filename);

        res.status(200).json(
        {
            "message": `User with id: ${id} deleted successfully`
        });
    })
    .catch((err) =>
    {
        console.error(err);
        if(err.name === 'CastError')
        {
            res.status(400).json(
            {
                "message": `Bad request, ${id} is not a valid id`
            });
        }
        else {
            res.status(500).json(err)
        }
    });
};

const register = (req, res) =>
{
    let newUser = new User(req.body);
    console.log(req.image_path)

    if(req.file)
    {
        newUser.image_path = process.env.STORAGE_ENGINE === 'S3' ? req.file.key : req.file.filename;
    }

    if (newUser.rank == null || newUser.rank == undefined || newUser.rank != Number)
    {
        newUser.rank = 0;
    }

    newUser.password = bcrypt.hashSync(req.body.password, 10);
```



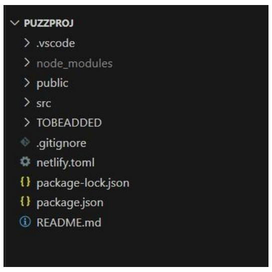
```
    newUser.password = bcrypt.hashSync(req.body.password, 10);
    newUser.save().then(data =>
    {
        data.password = undefined;
        return res.status(201).json(data);
    })
    .catch(err =>
    {
        return res.status(400).json(
        {
            message: err
        })
    }
    );
})

const login = (req, res) =>
    {
        User.findOne({email: req.body.email})
        .then(user =>
        {
            if (user && user.comparePassword(req.body.password) == true)
            {
                return res.status(200).json(
                {
                    _id: user._id,
                    token: jwt.sign(
                    {
                        email: user.email,
                        username: user.username,
                        _id: user._id
                    }, process.env.JWT_SECRET, )
                })
            }

            if (!user || user.comparePassword(req.body.password) == false)
            {
                return res.status(401).json(
                {
                    message: 'Authentication failed'
                })
            }

            return res.status(200).json(
            {
                _id: user._id,
```



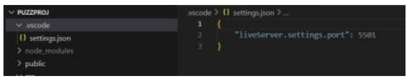
```
                {
                    _id: user._id,
                    email: user.email,
                    token: jwt.sign(
                    {
                        email: user.email,
                        username: user.username,
                        _id: user._id
                    }, process.env.JWT_SECRET, )
                })
            })
            .catch(err =>
            {
                return res.status(500).json(err)
            });
    };

const loginRequired = (req, res, next) =>
{
    console.log(req.user)
    if (req.user)
    {
        next();
    }
    else
    {
        return res.status(401).json(
        {
            message: "Unathorized user"
        });
    }
};

//exports functions
module.exports =
{
    readAll,
    register,
    login,
    loginRequired,
    readOne,
    updateData,
    deleteData
};
```

Fig 35 Fig 36 Fig 37

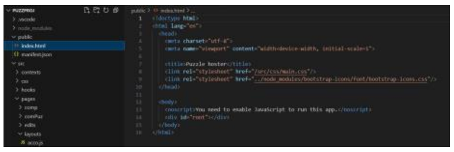
Then there is the front-end structure.



```
PUZZPROJ
  > .vscode
  > node_modules
  > public
  > src
  > TOBEADDED
    .gitignore
    netlify.toml
    package-lock.json
    package.json
    README.md
```



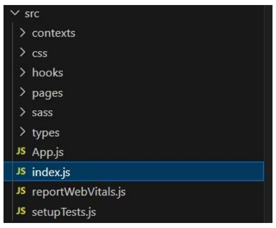
```
PUZZPROJ                 .vscode > {} settings.json > ...
  v .vscode              {
    {} settings.json         "liveServer.settings.port": 5501
  > node_modules         }
  > public
```



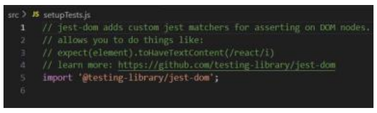
```
public > <> index.html > ...
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Puzzle Hoster</title>
    <link rel="stylesheet" href="/src/css/main.css"/>
    <link rel="stylesheet" href="../node_modules/bootstrap-icons/font/bootstrap-icons.css"/>
  </head>

  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

Fig 38 Fig 39 Fig 40

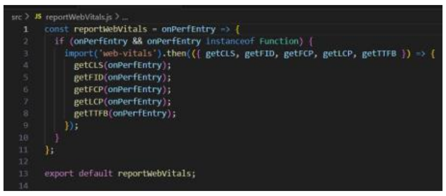
The based front-end structure was split into four folders (Vscode, Public, Node modules and Src). Vscode holds the live server port that is only used when the application is being locally hosted. Public holds a single page which its purpose is to tell the user that they need to enable JavaScript in order to use that application. Node modules hold all the node related modules that the application needs to run. Just leaving Src which holds the rest of the application.



```
v src
  > contexts
  > css
  > hooks
  > pages
  > sass
  > types
  JS App.js
  JS index.js
  JS reportWebVitals.js
  JS setupTests.js
```



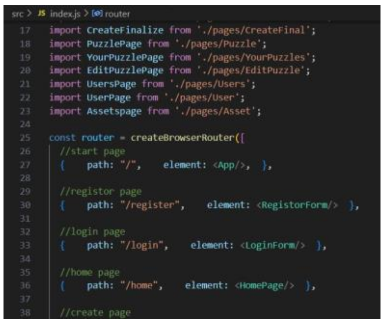
```
src > JS setupTests.js
// jest-dom adds custom jest matchers for asserting on DOM nodes.
// allows you to do things like:
// expect(element).toHaveTextContent(/react/i)
// learn more: https://github.com/testing-library/jest-dom
import '@testing-library/jest-dom';
```



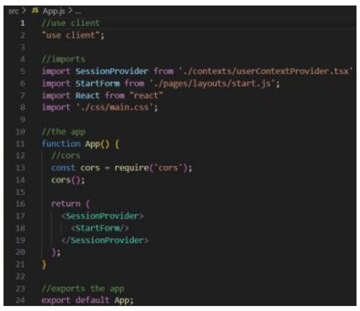
```
src > JS reportWebVitals.js > ...
const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};

export default reportWebVitals;
```

Fig 41 Fig 42 Fig 43

Inside the src folder there was four java script file (App.js, index.js, reportWebVital.js and setupTests.js) and six more folders (contexts, CSS, hooks, pages, sass and types). Starting with the setupTests.js all this does is import the testing library / just-Dom, then the reportWebVitals.js aids in the development by creating the applications vital metrics (both of these are prebuilt into react.js).



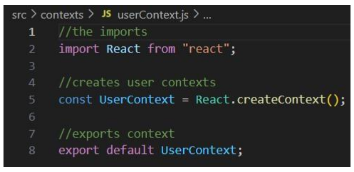
```
src > JS index.js > router
import CreateFinalize from './pages/CreateFinal';
import PuzzlePage from './pages/Puzzle';
import YourPuzzlePage from './pages/YourPuzzles';
import EditPuzzlePage from './pages/EditPuzzle';
import UsersPage from './pages/Users';
import UserPage from './pages/User';
import Assetspage from './pages/Asset';

const router = createBrowserRouter([
  //start page
  {    path: "/",    element: <App/>,  },

  //registor page
  {    path: "/register",    element: <RegistorForm/>  },

  //login page
  {    path: "/login",    element: <LoginForm/>  },

  //home page
  {    path: "/home",    element: <HomePage/>  },

  //create page
```



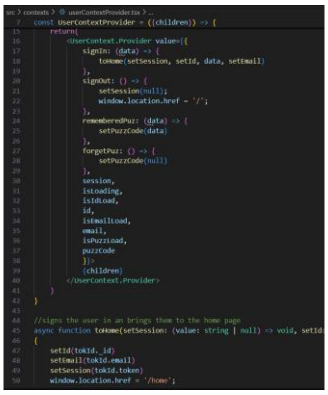
```
src > JS App.js > ...
//use client
"use client";

//imports
import SessionProvider from './contexts/userContextProvider.tsx'
import StartForm from './pages/layouts/start.js';
import React from "react"
import './css/main.css';

//the app
function App() {
  //cors
  const cors = require('cors');
  cors();

  return (
    <SessionProvider>
      <StartForm/>
    </SessionProvider>
  );
}

//exports the app
export default App;
```

Fig 44 Fig 45

Then there is index.js. This file was in charge of routing its users throughout the applications pages. Final of the JS files is App.js. In its most basic form this is the application / start page. Going onto the folders I'll start with the context folder. Within this folder there are two files userContext.js (this file creates and exports a React context called UserContext) and userContextProvider.tsx (this file grabs the context created be UserContext.js and gives it the variable it will need these being a sign in, sign out, remembered puz (remembers a puzzle) and forget puz (forgets the puzzle) functions and some variables that a logged in user would need to have e.g. the session token.).



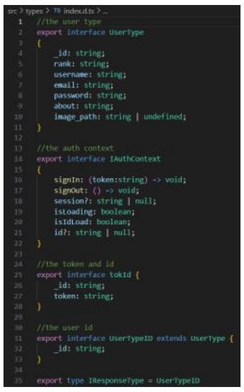
```
src > contexts > JS userContext.js > ...
//the imports
import React from "react";

//creates user contexts
const UserContext = React.createContext();

//exports context
export default UserContext;
```



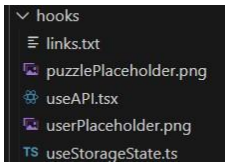
```
src > contexts > userContextProvider.tsx > ...
const UserContextProvider = ({children}) => {
    return(
        <UserContext.Provider value={{
            signIn: (data) => {
                toHome(setSession, setId, data, setEmail)
            },
            signOut: () => {
                setSession(null);
                window.location.href = '/';
            },
            rememberedPuz: (data) => {
                setPuzzCode(data)
            },
            forgetPuz: () => {
                setPuzzCode(null)
            },
            session,
            isLoading,
            isIdLoad,
            id,
            isEmailLoad,
            email,
            isPuzzLoad,
            puzzCode
            }}>
            {children}
        </UserContext.Provider>
    )
}

//signs the user in an brings them to the home page
async function toHome(setSession: (value: string | null) => void, setId
{
    setId(tokId._id)
    setEmail(tokId.email)
    setSession(tokId.token)
    window.location.href = '/home';
```

Fig 46 Fig 47

The next two folders are CSS and sass both of which only hold files used for the applications UI designs in the form of a SCSS and CSS files. Then there is the types folder which holds a single file index.d.ts. This file contains the type of data each type should have. The fifth folder hooks contain five files links.txt (a text file that holds images, icon and miscellaneous links), two placeholder Pngs (used if a puzzler or user does not have an image), useAPI.tsx (Holds version of axios call requests) and useStorageState.ts (is how the application stores certain forms of data).



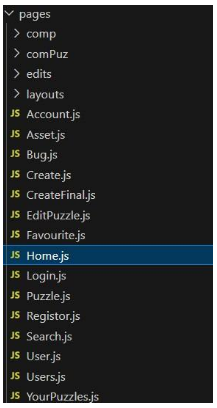
```
src > types > TS index.d.ts > ...
//the user type
export interface UserType
{
    _id: string;
    rank: string;
    username: string;
    email: string;
    password: string;
    about: string;
    image_path: string | undefined;
}

//the auth context
export interface IAuthContext
{
    signIn: (token:string) -> void;
    signOut: () -> void;
    session?: string | null;
    isLoading: boolean;
    isIdLoad: boolean;
    id?: string | null;
}

//the token and id
export interface tokId {
    _id: string;
    token: string;
}

//the user id
export interface UserTypeID extends UserType {
    _id: string;
}

export type IResponseType = UserTypeID
```



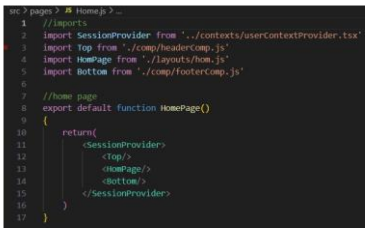
```
hooks
  links.txt
  puzzlePlaceholder.png
  useAPI.tsx
  userPlaceholder.png
  useStorageState.ts
```

Fig 48 Fig 49

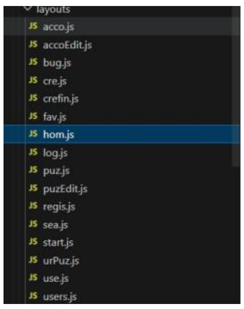
Finally, the leaves the final folder within src, Pages. The pages folder has four folders inside it (comp (holds the miscellaneous components), comPuz (holds the puzzle components), edits (holds the account edit page) and layouts (holds the layout components)) and all the JavaScript files that the index.js calls as pages. Almost all these files call the header and footer components (found within the comp folder) and its layout component.



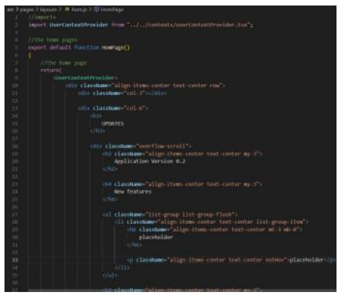
```
pages
  comp
  comPuz
  edits
  layouts
  Account.js
  Asset.js
  Bug.js
  Create.js
  CreateFinal.js
  EditPuzzle.js
  Favourite.js
  Home.js
  Login.js
  Puzzle.js
  Registor.js
  Search.js
  User.js
  Users.js
  YourPuzzles.js
```



```
src > pages > Home.js > ...
//imports
import SessionProvider from '../contexts/userContextProvider.tsx'
import Top from './comp/headerComp.js'
import HomPage from './layouts/hom.js'
import Bottom from './comp/footerComp.js'

//home page
export default function HomePage()
{
    return(
        <SessionProvider>
            <Top/>
            <HomPage/>
            <Bottom/>
        </SessionProvider>
    )
}
```



```
layouts
  acco.js
  accoEdit.js
  bug.js
  cre.js
  crefin.js
  fav.js
  hom.js
  log.js
  puz.js
  puzEdit.js
  regis.js
  sea.js
  start.js
  urPuz.js
  use.js
  users.js
```



```
src > pages > layouts > hom.js > HomPage
//imports
import UserContextProvider from "../../contexts/userContextProvider.tsx";

//the home pages
export default function HomPage()
{
    //the home page
    return(
        <UserContextProvider>
            <div className="align-items-center text-center row">
                <div className="col-3"></div>

                <div className="col-6">
                    <h3>
                        UPDATES
                    </h3>

                    <div className="overflow-scroll">
                        <h2 className="align-items-center text-center my-3">
                            Application Version 0.2
                        </h2>

                        <h4 className="align-items-center text-center my-3">
                            New features
                        </h4>

                        <ul className="list-group list-group-flush">
                            <li className="align-items-center text-center list-group-item">
                                <h6 className="align-items-center text-center mt-3 mb-0">
                                    placeholder
                                </h6>

                                <p className="align-items-center text-center notHov">placeholder</p>
                            </li>
                        </ul>

                        <h2 className="align-items-center text-center my-3">
```

Fig 50 Fig 51 Fig 52 Fig 53

4.2.3 Application architecture

The applications architecture starts with MongoDB as its database of which it links to the node.js back end using mongoose. Mongoose was a third-party library made with the intended purpose of linking node.js and MongoDB together. Once linked with the back end by mongoose, node.js runs the server with JavaScript functions. Express.js is a node.js framework that was used to make the back end into a restful API.

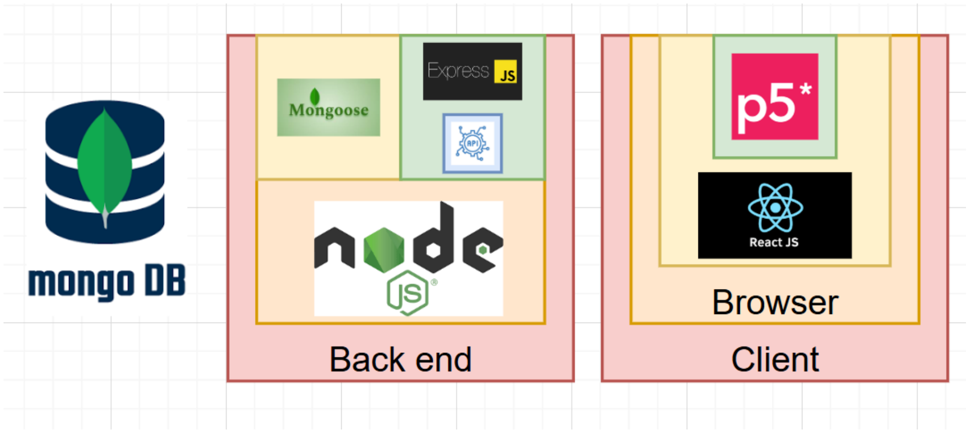


Fig 54

4.2.4 Database design

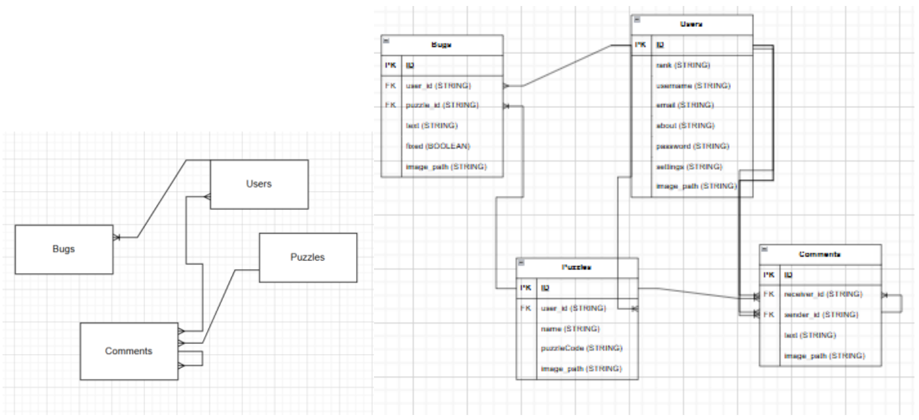
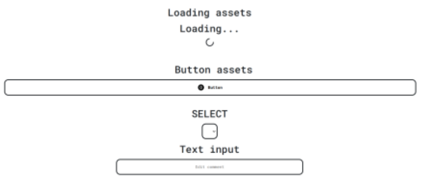
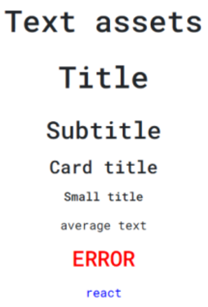


Fig 55

4.3 User interface design

The user Interface (UI) was designed to be simple but informative while also being able to adjust to different screen sizes. This was achieved by limiting the number of assets to only seven text types (Title (used on the home page), subtitle (Used to title less important info), card title (used to title puzzles and users), small title (used as a subtitle to the card title), average text (used for generic texts), ERROR (used for errors) and react (used for an interactable piece of text). Four image (two for users and two for every other image) assets and one button type.





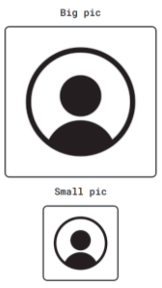
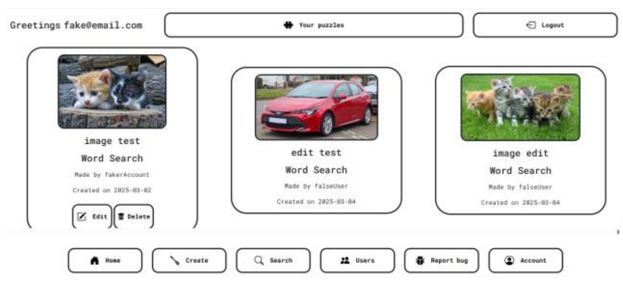


Fig 56 Fig 57 Fig 58 Fig 59



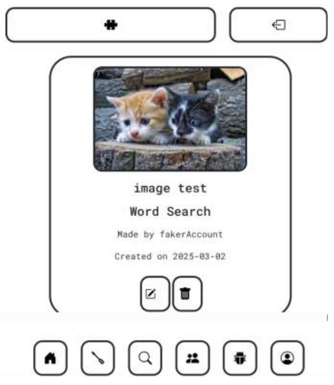


Fig 60 Fig 61

4.3.1 Wireframe

Draft 1

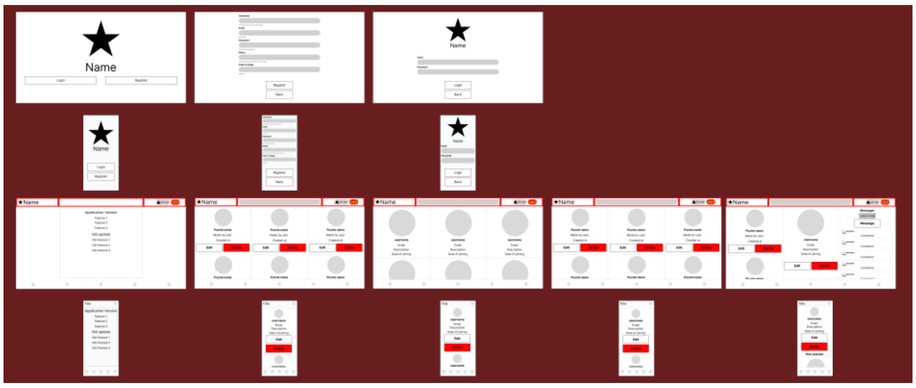


Fig 62

Draft 2

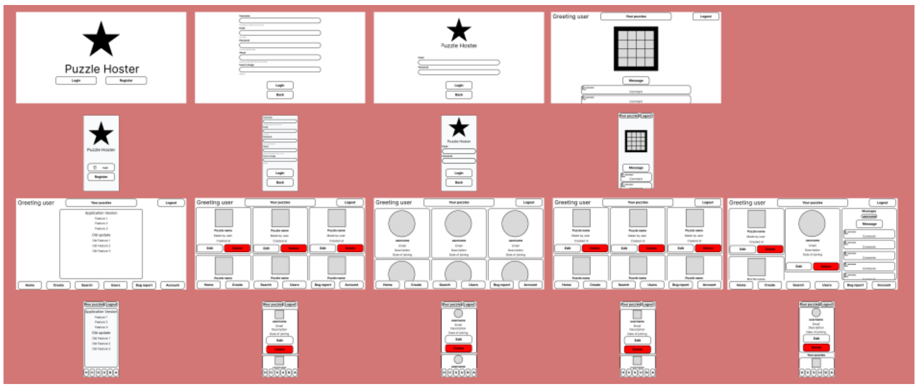


Fig 63

Draft 3

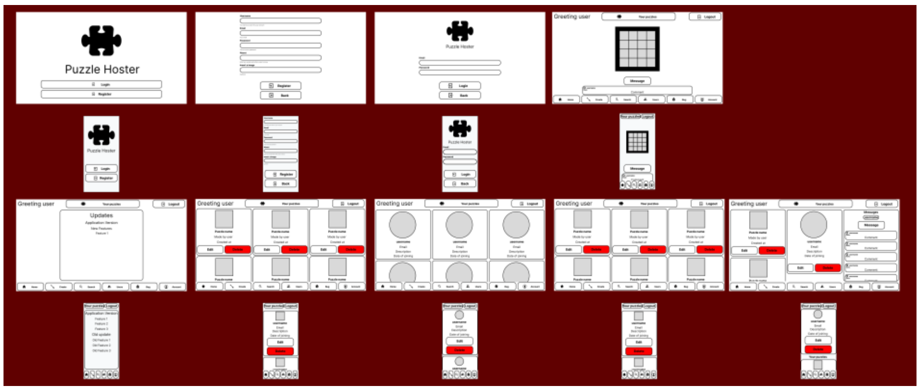


Fig 64

4.3.2 User Flow Diagram

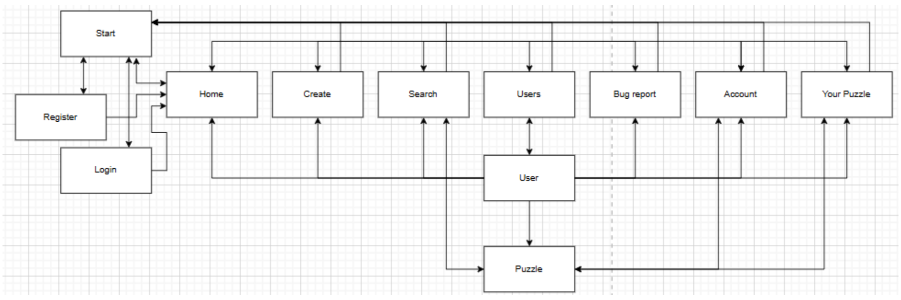
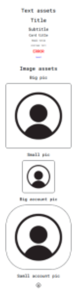


Fig 65

4.3.3 Style guide



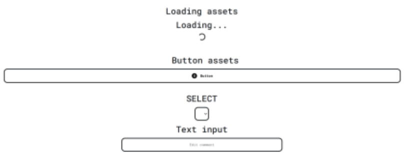


Fig 66 Fig 67

4.4 Conclusion

In conclusion the program design of the application was that of a simplistic minimalism that was focused more on bluntly giving the information. Just why the UI for example is for the most part only black and white with the only exception being the red for errors and different shade of blue for the interactable text.

Even just looking at the fact that the final database is only four tables down from the six of the original idea with the comments table performing multiple functions for the application (comments, messages and replies) when all three were at one point planned to be the own tables.

5 Implementation

5.1 Introduction

The application has been developed using the following technologies:

- React.js
  React.js is a user interface that can was used to shape he rendering and routing of the application
- Bootstrap
  Bootstrap is a design framework designed around CSS
- Node.js
  Node.js is a JavaScript tool that allows servers to use JavaScript functions for scripting
- Express.js
  Express.js is a back-end API build that works with Node.js

The application for this project was a puzzle based social media platform that allows its users to create and share puzzles (crosswords and word searches) with other users on the application. Users are able to comment on these puzzles and reply to these comments, both of which can be edit and delete by the user that posted them. The application's users can also message each other.

5.2 Development environment

The integrated development environment (IDE) used for this project was visual studio code (vs code). Vs code was developed by Microsoft and released in April 2015. It features include syntax highlighting version control of Git, debugging support and built in extension support.

Git was used to track, update, upload and transfer the versions of the back end and front end of the project. I used Git through GitHub this also is what allowed me to upload the files onto Vercel (back end) and Netlify (front end).

5.3 Sprint 1

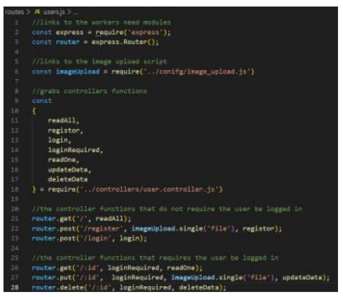
5.3.1 Goals

There were three goals set for the first sprint these being the users and puzzle backend functionality, complete the blueprint for the front-end word searches and to test the backend in insomnia API.

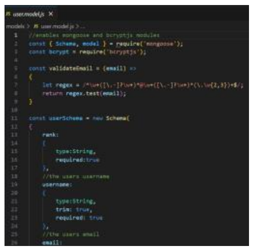
Sprint 1

5.3.2 Users and puzzles backend functionality

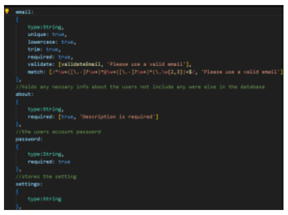
The functionality of this was to create the models (what variable each element has), controllers (what controllers each member of the database), and routes (how they move across the back end). These were all fully completed for the users but the puzzles I was not yet fully sure how its data would be store and used just I was unable to finish the backend functionality for the puzzles.



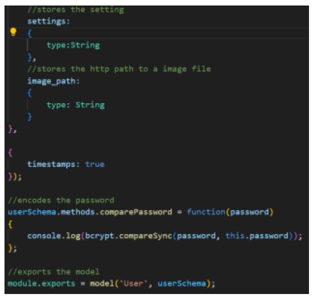
```
routes > JS users.js > ...
//links to the workers need modules
const express = require('express');
const router = express.Router();

//links to the image upload script
const imageUpload = require('../conifg/image_upload.js')

//grabs controllers functions
const
{
    readAll,
    registor,
    login,
    loginRequired,
    readOne,
    updateData,
    deleteData
} = require('../controllers/user.controller.js')

//the controller functions that do not require the user be logged in
router.get('/', readAll);
router.post('/register', imageUpload.single('file'), registor);
router.post('/login', login);

//the controller functions that requires the user be logged in
router.get('/:id', loginRequired, readOne);
router.put('/:id',  loginRequired, imageUpload.single('file'), updateData);
router.delete('/:id', loginRequired, deleteData);
```



```
JS user.model.js ×
models > JS user.model.js > ...
//enables mongoose and bcryptjs modules
const { Schema, model } = require('mongoose');
const bcrypt = require('bcryptjs');

const validateEmail = (email) =>
{
    let regex = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;
    return regex.test(email);
}

const userSchema = new Schema(
{
    rank:
    {
        type:String,
        required:true
    },
    //the users username
    username:
    {
        type:String,
        trim: true,
        required: true
    },
    //the users email
    email:
```

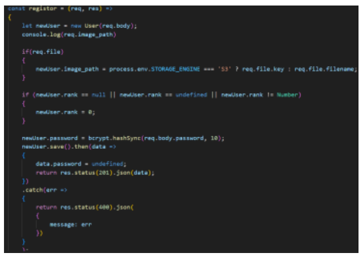


```
email:
{
    type:String,
    unique: true,
    lowercase: true,
    trim: true,
    required: true,
    validate: [validateEmail, 'Please use a valid email'],
    match: [/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/, 'Please use a valid email']
},
//holds any nessary info about the users not include any were else in the database
about:
{
    type:String,
    required: [true, 'Description is required']
},
//the users account password
password:
{
    type:String,
    required: true
},
//stores the setting
settings:
{
    type:String
},
```

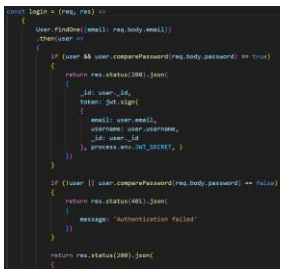


```
    //stores the setting
    settings:
    {
        type:String
    },
    //stores the http path to a image file
    image_path:
    {
        type: String
    }
},

{
    timestamps: true
});

//encodes the password
userSchema.methods.comparePassword = function(password)
{
    console.log(bcrypt.compareSync(password, this.password));
};

//exports the model
module.exports = model('User', userSchema);
```

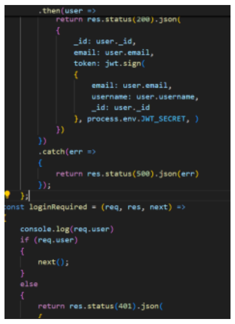
Fig 68 Fig 69 Fig 70 Fig 71



```
const registor = (req, res) =>
{
    let newUser = new User(req.body);
    console.log(req.image_path)

    if(req.file)
    {
        newUser.image_path = process.env.STORAGE_ENGINE === 'S3' ? req.file.key : req.file.filename;
    }

    if (newUser.rank == null || newUser.rank == undefined || newUser.rank != Number)
    {
        newUser.rank = 0;
    }

    newUser.password = bcrypt.hashSync(req.body.password, 10);
    newUser.save().then(data =>
    {
        data.password = undefined;
        return res.status(201).json(data);
    })
    .catch(err =>
    {
        return res.status(400).json(
        {
            message: err
        })
    }
```



```
const login = (req, res) =>
    {
        User.findOne({email: req.body.email})
        .then(user =>
        {
            if (user && user.comparePassword(req.body.password) == true)
            {
                return res.status(200).json(
                {
                    _id: user._id,
                    token: jwt.sign(
                    {
                        email: user.email,
                        username: user.username,
                        _id: user._id
                    }, process.env.JWT_SECRET, )
                })
            }

            if (!user || user.comparePassword(req.body.password) == false)
            {
                return res.status(401).json(
                {
                    message: 'Authentication failed'
                })
            }

            return res.status(200).json(
            {
```

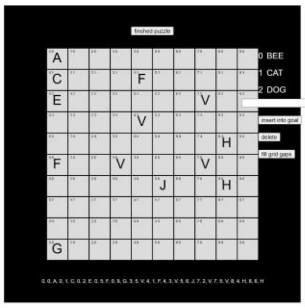


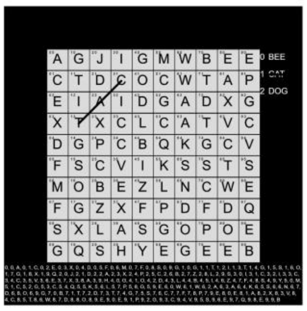
```
        .then(user =>
            return res.status(200).json(
            {
                _id: user._id,
                email: user.email,
                token: jwt.sign(
                {
                    email: user.email,
                    username: user.username,
                    _id: user._id
                }, process.env.JWT_SECRET, )
            })
        })
        .catch(err =>
        {
            return res.status(500).json(err)
        });
    };
const loginRequired = (req, res, next) =>
{
    console.log(req.user)
    if (req.user)
    {
        next();
    }
    else
    {
        return res.status(401).json(
```

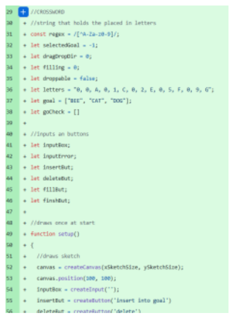
Fig 72 Fig 73 Fig 74

5.3.3 Complete the blueprint for the front-end word searches

The goal of creating a blueprint for how a word search would work within the application, mostly in the form make a grid. Tracking and remembering where each letter are. Setting up a key that holds the puzzle's goal words and allow them to be added and or deleted to. Finaly made the blueprint where the mouse is track and the wordsearch checks if a goal word in the key is crossed out.

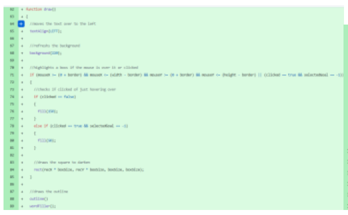




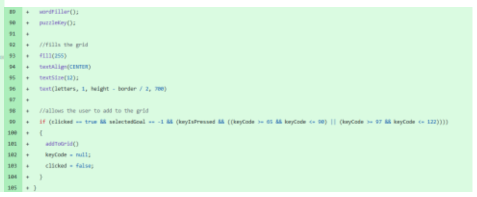


```
+ //CROSSWORD
+ //string that holds the placed in letters
+ const regex = /[^A-Za-z0-9]/;
+ let selectedGoal = -1;
+ let dragDropDir = 0;
+ let filling = 0;
+ let droppable = false;
+ let letters = "0, 0, A, 0, 1, C, 0, 2, E, 0, 5, F, 0, 9, G";
+ let goal = ["BEE", "CAT", "DOG"];
+ let goCheck = []
+
+ //inputs an buttons
+ let inputBox;
+ let inputError;
+ let insertBut;
+ let deleteBut;
+ let fillBut;
+ let finshBut;
+
+ //draws once at start
+ function setup()
+ {
+   //draws sketch
+   canvas = createCanvas(xSketchSize, ySketchSize);
+   canvas.position(100, 100);
+   inputBox = createInput("");
+   insertBut = createButton('insert into goal')
+   deleteBut = createButton('delete')
```

Fig 75 Fig 76 Fig 77



```
+ function draw()
+ {
+  //moves the text over to the left
+  textAlign(LEFT);
+
+  //refreshs the background
+  background(120);
+
+  //highlights a boss if the mouse is over it or clicked
+  if (mouseX >= (0 + border) && mouseX <= (width - border) && mouseY >= (0 + border) && mouseY <= (height - border) || (clicked == true && selectedGoal == -1))
+  {
+   //checks if clicked of just hovering over
+   if (clicked == false)
+   {
+    fill(150);
+   }
+   else if (clicked == true && selectedGoal == -1)
+   {
+    fill(90);
+   }
+
+   //draws the square to darken
+   rect(recX * boxSize, recY * boxSize, boxSize, boxSize);
+  }
+
+  //draws the outline
+  outline()
+  wordFiller();
```

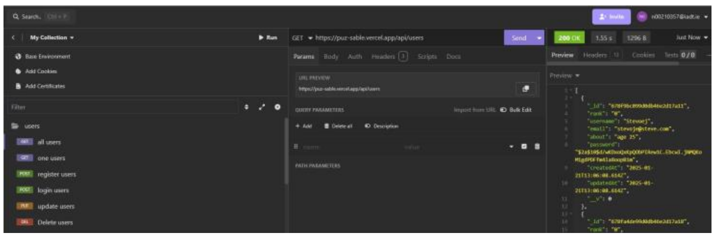


```
+  wordFiller();
+  puzzleKey();
+
+  //fills the grid
+  fill(255)
+  textAlign(CENTER)
+  textSize(12);
+  text(letters, 1, height - border / 2, 700)
+
+  //allows the user to add to the grid
+  if (clicked == true && selectedGoal == -1 && (keyIsPressed && ((keyCode >= 65 && keyCode <= 90) || (keyCode >= 97 && keyCode <= 122))))
+  {
+   addToGrid()
+   keyCode = null;
+   clicked = false;
+  }
+ }
```

Fig 78 Fig 79

5.3.4 Test the backend in insomnia API

The purpose of insomnia was to test the functionality of the backend system. During this sprint I was able to get the login, register, update, delete, find all and one functions working within insomnia API. However, due to the not quite working out how I was going to store the puzzle data, I was unable to test the puzzles backend functionality.



GET https://puz-sable.vercel.app/api/users

```
[
  {
    "_id": "678f9bc099d0db46e2d17a11",
    "rank": "0",
    "username": "Stevoej",
    "email": "stevoje@steve.com",
    "about": "age 25",
    "password":
"$2a$10$d/wKDoxQxKpQObPIRewIC.Ebcwl.jNMQKo
MlgdPDFfm4laBoop8lm",
    "createdAt": "2025-01-
21T13:06:08.614Z",
    "updatedAt": "2025-01-
21T13:06:08.614Z",
    "__v": 0
  },
  {
    "_id": "678fadde99d0db46e2d17a18",
    "rank": "0",
```

Fig 80

5.4 Sprint 2

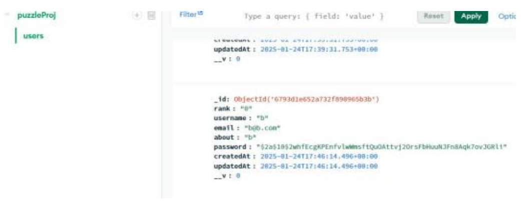
5.4.1 Goals

There were three goals set for the second sprint. These goals were to host the start of the backend, to start the front end and get what I had of the front end hosted.

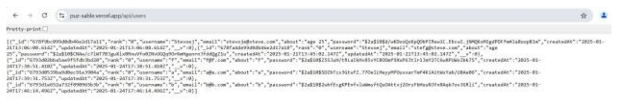
[Sprint 2]

5.4.2 Host what I had of the back end.

This goal was to get the back end hosted on a cloud server and database. So, I used Vercel to host the back-end code and MongoDB as the database. However, as of sprint 2 only the user's functionality of the back end was fully functioning.



```
updatedAt : 2025-01-24T17:39:31.753+00:00
__v : 0

_id: ObjectId('6793d1e652a732f898965b3b')
rank : "0"
username : "b"
email : "b@b.com"
about : "b"
password : "$2a$10$2whfEcgKPEnfvlwWmsftQuOAttvj2OrsFbHuuNJFn8Aqk7ovJGRl1"
createdAt : 2025-01-24T17:46:14.496+00:00
updatedAt : 2025-01-24T17:46:14.496+00:00
__v : 0
```



```
[{"_id":"678f9bc099d0db46e2d17a11","rank":"0","username":"Stevoej","email":"stevoje@steve.com","about":"age 25","password":"$2a$10$d/wKDoxQxKpQObPIRewIC.Ebcwl.jNMQKoMlgdPDFfm4laBoop8lm","createdAt":"2025-01-21T13:06:08.614Z","updatedAt":"2025-01-21T13:06:08.614Z","__v":0},{"_id":"678fadde99d0db46e2d17a18","rank":"0","username":"Stevoej","email":"stef@steve.com","about":"age 25","password":"[illegible]","createdAt":"2025-01-21T13:45:02.147Z","updatedAt":"2025-01-21T13:45:02.147Z","__v":0},
{"_id":"6793d02bba5ae9f5fdb3bd20","rank":"0","username":"f","email":"f@f.com","about":"f","password":"[illegible]","createdAt":"2025-01-24T17:38:51.410Z","updatedAt":"2025-01-24T17:38:51.410Z","__v":0},
{"_id":"6793d0539ba9d0ec91e3904a","rank":"0","username":"a","email":"a@a.com","about":"a","password":"[illegible]","createdAt":"2025-01-24T17:39:31.753Z","updatedAt":"2025-01-24T17:39:31.753Z","__v":0},
{"_id":"6793d1e652a732f898965b3b","rank":"0","username":"b","email":"b@b.com","about":"b","password":"$2a$10$2whfEcgKPEnfvlwWmsftQuOAttvj2OrsFbHuuNJFn8Aqk7ovJGRl1","createdAt":"2025-01-24T17:46:14.496Z","updatedAt":"2025-01-24T17:46:14.496Z","__v":0}]
```

Fig 81 Fig 82

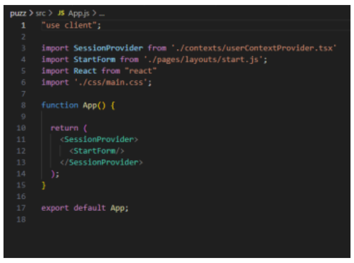
5.4.3 Host what I had of the back end.

This goal was to get the back end hosted on a cloud server and database. So, I used Vercel to host the back-end code and MongoDB as the database. However, as of sprint 2 only the user's functionality of the back end was fully functioning

5.4.4 Start the front end

This front end was started as a react.js application. During this sprint I got three pages fully functioning (the start, register and login), started the home page and the applications routing. While I was able to connect to the back end and use the register and login functions (on their pages) with the front end I had yet to get the context (remember data across the application pages e.g. which user is signed in).

The start page



```
puzz > src > JS App.js > ...
"use client";

import SessionProvider from './contexts/userContextProvider.tsx'
import StartForm from './pages/layouts/start.js';
import React from "react"
import './css/main.css';

function App() {

  return (
    <SessionProvider>
      <StartForm/>
    </SessionProvider>
  );
}

export default App;
```

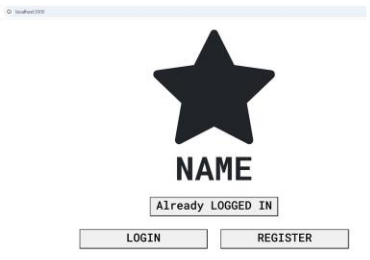
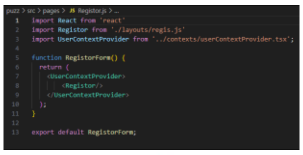
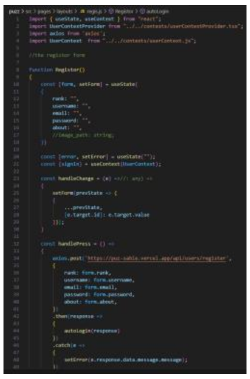


Fig 83 Fig 84 Fig 85 Fig 86

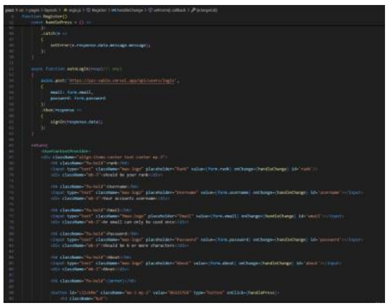
The register page



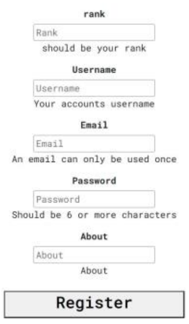
```
puzz > src > pages > JS Registor.js > ...
import React from 'react'
import Registor from './layouts/regis.js'
import UserContextProvider from '../contexts/userContextProvider.tsx';

function RegistorForm() {
  return (
    <UserContextProvider>
        <Registor/>
    </UserContextProvider>
  );
}

export default RegistorForm;
```



```
puzz > src > pages > layouts > JS regis.js > ⊘ Registor > ⊘ autoLogin
import { useState, useContext } from "react";
import UserContextProvider from "../../contexts/userContextProvider.tsx";
import axios from 'axios';
import UserContext  from "../../contexts/userContext.js";

//the register form

function Registor()
{
    const [form, setForm] = useState(
    {
        rank: "",
        username: "",
        email: "",
        password: "",
        about: "",
        //image_path: string,
    })

    const [error, setError] = useState("");
    const {signIn} = useContext(UserContext);

    const handleChange = (e) =>//: any) =>
    {
        setForm(prevState => (
        {
            ...prevState,
            [e.target.id]: e.target.value
        }));
    }

    const handlePress = () =>
    {
        axios.post('https://puz-sable.vercel.app/api/users/register',
        {
            rank: form.rank,
            username: form.username,
            email: form.email,
            password: form.password,
            about: form.about,
        })
        .then(response =>
        {
            autoLogin(response)
        })
        .catch(e =>
        {
            setError(e.response.data.message.message);
        })
```



```
puzz > src > pages > layouts > JS regis.js > ⊘ Registor > handleChange > ⊘ setForm() callback > [e.target.id]
function Registor()
    const handlePress = () =>
        })
        .catch(e =>
        {
            setError(e.response.data.message.message);
        })
    }

    async function autoLogin(resp)//: any)
    {
        axios.post('https://puz-sable.vercel.app/api/users/login',
        {
            email: form.email,
            password: form.password
        })
        .then(response =>
        {
            signIn(response.data);
        })
    }

    return(
        <UserContextProvider>
        <div className="align-items-center text-center mx-3">
            <h4 className="fw-bold">rank</h4>
            <input type="text" className="max-logo" placeholder="Rank" value={form.rank} onChange={handleChange} id="rank"/>
            <div className="mb-3">should be your rank</div>

            <h4 className="fw-bold">Username</h4>
            <input type="text" className="max-logo" placeholder="Username" value={form.username} onChange={handleChange} id="username"></input>
            <div className="mb-3">Your accounts username</div>

            <h4 className="fw-bold">Email</h4>
            <input type="text" className="max-logo" placeholder="Email" value={form.email} onChange={handleChange} id="email"></input>
            <div className="mb-3">An email can only be used once</div>

            <h4 className="fw-bold">Password</h4>
            <input type="text" className="max-logo" placeholder="Password" value={form.password} onChange={handleChange} id="password"></input>
            <div className="mb-3">Should be 6 or more characters</div>

            <h4 className="fw-bold">About</h4>
            <input type="text" className="max-logo" placeholder="About" value={form.about} onChange={handleChange} id="about"></input>
            <div className="mb-3">About</div>

            <h4 className="fw-bold">{error}</h4>

            <button id="clickMe" className="mx-3 my-2" value="[illegible]" type="button" onClick={handlePress}>
                <h3 className="[illegible]">
```



rank

Rank

should be your rank

Username

Username

Your accounts username

Email

Email

An email can only be used once

Password

Password

Should be 6 or more characters

About

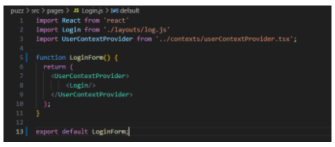
About

About

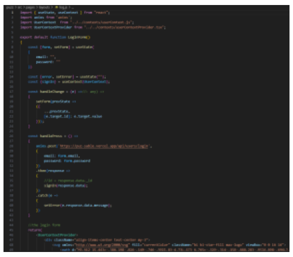
Register

Fig 87 Fig 88 Fig 89 Fig 90

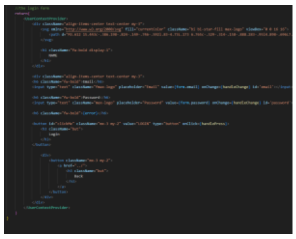
The login page



```
puzz > src > pages > Login.js > default
import React from 'react'
import Login from './layouts/log.js'
import UserContextProvider from '../contexts/userContextProvider.tsx';

function LoginForm() {
  return (
    <UserContextProvider>
      <Login/>
    </UserContextProvider>
  );
}

export default LoginForm;
```



```
puzz > src > pages > layouts > log.js > ...
import { useState, useContext } from "react";
import axios from 'axios';
import UserContext  from "../../contexts/userContext.js";
import UserContextProvider from "../../contexts/userContextProvider.tsx";

export default function LoginForm()
{
    const [form, setForm] = useState(
    {
        email: "",
        password: ""
    });

    const [error, setError] = useState("");
    const {signIn} = useContext(UserContext);

    const handleChange = (e) =>//: any) =>
    {
        setForm(prevState =>
        ({
            ...prevState,
            [e.target.id]: e.target.value
        }));
    }

    const handlePress = () =>
    {
        axios.post('https://puz-sable.vercel.app/api/users/login',
        {
            email: form.email,
            password: form.password
        })
        .then(response =>
        {
            //id = response.data._id
            signIn(response.data);
        })
        .catch(e =>
        {
            setError(e.response.data.message);
        })
    }

    //the login form
    return(
        <UserContextProvider>
            <div className="align-items-center text-center my-3">
                <svg xmlns="http://www.w3.org/2000/svg" fill="currentColor" className="bi bi-star-fill max-logo" viewBox="0 0 16 16">
                    <path d="M3.612 15.443c-.386.198-.824-.149-.746-.592l.83-4.73L.173 6.765c-.329-.314-.158-.888.283-.95l4.898-.696L7.538.792c [illegible]
```



```
    //the login form
    return(
        <UserContextProvider>
            <div className="align-items-center text-center my-3">
                <svg xmlns="http://www.w3.org/2000/svg" fill="currentColor" className="bi bi-star-fill max-logo" viewBox="0 0 16 16">
                    <path d="M3.612 15.443c-.386.198-.824-.149-.746-.592l.83-4.73L.173 6.765c-.329-.314-.158-.888.283-.95l4.898-.696L7.538.792c [illegible]
                </svg>

                <h1 className="fw-bold display-1">
                    PUZZ
                </h1>
            </div>

            <div className="align-items-center text-center my-3">
            <h6 className="fw-bold">Email</h6>
            <input type="text" className="max-logo" placeholder="Email" value={form.email} onChange={handleChange} id='email'></input>

            <h6 className="fw-bold">Password</h6>
            <input type="text" className="max-logo" placeholder="Password" value={form.password} onChange={handleChange} id='password' [illegible]

            <h6 className="fw-bold">{error}</h6>

            <button id="clickMe" className="mx-3 my-2" value="LOGIN" type="button" onClick={handlePress}>
                <h3 className="but">
                    Login
                </h3>
            </button>

                <div>
                    <button className="mx-3 my-2">
                        <a href="../">
                            <h3 className="but">
                                Back
                            </h3>
                        </a>
                    </button>
                </div>
            </div>
        </UserContextProvider>
    )
}
```

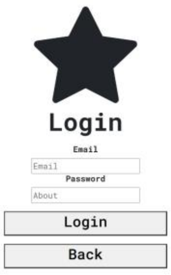
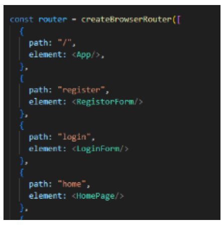


Fig 91 Fig 92 Fig 93

Fig 94

The applications routing



```
const router = createBrowserRouter([
  {
    path: "/",
    element: <App/>,
  },
  {
    path: "register",
    element: <RegistorForm/>
  },
  {
    path: "login",
    element: <LoginForm/>
  },
  {
    path: "home",
    element: <HomePage/>
  },
```

Fig 95

5.4.5 Host the front end

Due to not getting the context set up while starting the front end I decided to delay hosting the application just leaving this goal uncompleted during this sprint.

5.5 Sprint 3

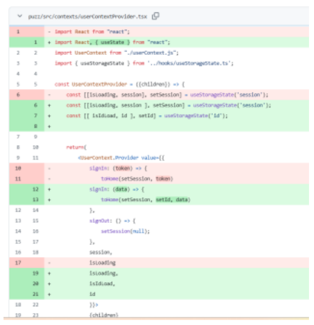
5.5.1 Goals

The goals that I had set out to achieve for sprint three were to finish the fronts context functionality, make the home page and the account page.

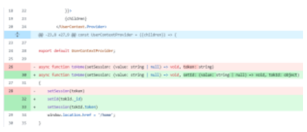
Sprint 3

5.5.2 The context

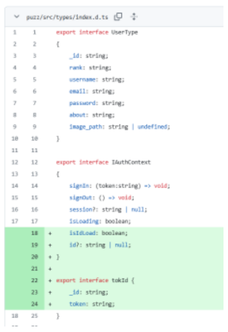
The context or user context was that applications ability to remember data across different pages. This means it's necessary for features such as knowing who is the signed, what are permission they have, their sign in and sign out functions and to tell the front end if the data was has been received yet (the loading variables). I had quite a bit of problems getting context to function with react, this was most likely because of a misunderstanding of how reacts built in create context function and how it was meant to be called within the return functions.



```
puzz/src/contexts/userContextProvider.tsx
- import React from "react";
+ import React, { useState } from "react";
  import UserContext from "./userContext.js";
  import { useStorageState } from '../hooks/useStorageState.ts';

  const UserContextProvider = ({children}) => {
-     const [[isLoading, session], setSession] = useStorageState('session');
+     const [[isLoading, session ], setSession] = useStorageState('session');
+     const [[ isIdLoad, id ], setId] = useStorageState('id');
+

      return(
          <UserContext.Provider value={{
-             signIn: (token) => {
-                 toHome(setSession, token)
+             signIn: (data) => {
+                 toHome(setSession, setId, data)
              },
              signOut: () => {
                  setSession(null);
              },
              session,
-             isLoading
+             isLoading,
+             isIdLoad,
+             id
          }}>
          {children}
```



```
          }}>
          {children}
      </UserContext.Provider>
@@ -23,8 +27,9 @@ const UserContextProvider = ({children}) => {

export default UserContextProvider;

- async function toHome(setSession: (value: string | null) => void, token: string)
+ async function toHome(setSession: (value: string | null) => void, setId: (value: string | null) => void, tokId: object)
  {
-     setSession(token)
+     setId(tokId._id)
+     setSession(tokId.token)
      window.location.href = '/home';
  }
```

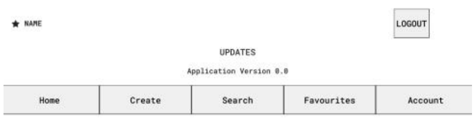


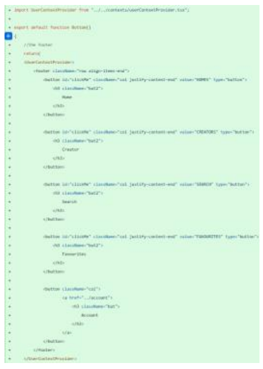
```
puzz/src/types/index.d.ts
  export interface UserType
  {
      _id: string;
      rank: string;
      username: string;
      email: string;
      password: string;
      about: string;
      image_path: string | undefined;
  }

  export interface IAuthContext
  {
      signIn: (token:string) => void;
      signOut: () => void;
      session?: string | null;
      isLoading: boolean;
+     isIdLoad: boolean;
+     id?: string | null;
+ }
+
+ export interface tokId {
+     _id: string;
+     token: string;
  }
```

Fig 96 Fig 97 Fig 98

5.5.3 The Home page

The home page is the page of the application that the user was sent to after they are signed in. With its construction I also had to make the first draft header and footer that will be used on nearly every other page that will be in the front end of the application.





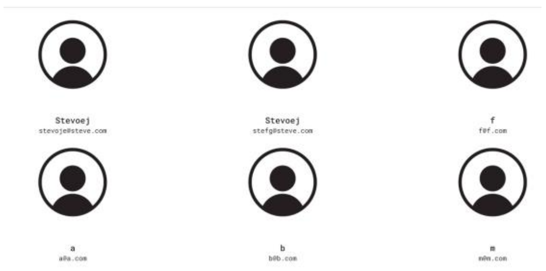
```
+ import UserContextProvider from "../../contexts/userContextProvider.tsx";
+
+ export default function Bottom()
+ {
+     //the footer
+     return(
+     <UserContextProvider>
+         <footer className="row align-items-end">
+             <button id="clickMe" className="col justify-content-end" value="HOMEY" type="button">
+                 <h3 className="but2">
+                     Home
+                 </h3>
+             </button>
+
+             <button id="clickMe" className="col justify-content-end" value="CREATORS" type="button">
+                 <h3 className="but2">
+                     Creator
+                 </h3>
+             </button>
+
+             <button id="clickMe" className="col justify-content-end" value="SEARCH" type="button">
+                 <h3 className="but2">
+                     Search
+                 </h3>
+             </button>
+
+             <button id="clickMe" className="col justify-content-end" value="FAVOURITES" type="button">
+                 <h3 className="but2">
+                     Favourites
+                 </h3>
+             </button>
+
+             <button className="col">
+                 <a href="../account">
+                     <h3 className="but">
+                         Account
+                     </h3>
+                 </a>
+             </button>
+         </footer>
+     </UserContextProvider>
```

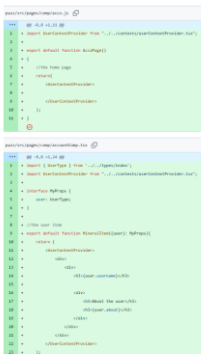
```
puzz/src/pages/comp/hom.js
+         <nav className="row align-items-center text-center p-2">
+             <div className="col-3">
+                 <ul className="nav justify-content-start align-items-center text-center">
+                     <li className="nav-item px-2">
+                         <svg xmlns="http://www.w3.org/2000/svg" fill="currentColor" className="bi bi-star-fill [illegible]" viewBox="0 0 16 16">
+                             <path d="[illegible]"/>
+                         </svg>
+                     </li>
+
+                     <li className="nav-item px-2">
+                         <h4 className="fw-bold align-items-center text-center">
+                             NAME
+                         </h4>
+                     </li>
+                 </ul>
+             </div>
+
+             <div className="col-6">
+
+             </div>
+
+             <div className="col-3">
+                 <button id="clickMe" className="justify-content-end" value="LOGOUT" type="button" onClick={handlePress}>
+                     <h3 className="but2">
+                         LOGOUT
+                     </h3>
+                 </button>
+             </div>
+         </nav>
+
+         <div className="align-items-center text-center">
+             <h3>
+                 UPDATES
+             </h3>
+
+             <h4>
+                 Application Version 0.0
+             </h4>
+         </div>
+     </div>
+     </UserContextProvider>
```

Fig 99 Fig 100 Fig 101

5.5.4 The Account page

The Account page was only start as I spent far too much time getting the contexts to work during this Sprint, I had only managed to get the account page to call all the users and render their information.





```
puzz/src/pages/comp/acco.js
@@ -0,0 +1,11 @@
+ import UserContextProvider from "../../contexts/userContextProvider.tsx";
+
+ export default function AccoPage()
+ {
+     //the home page
+     return(
+         <UserContextProvider>
+
+         </UserContextProvider>
+     );
+ }
```

```
puzz/src/pages/comp/accountComp.tsx
@@ -0,0 +1,24 @@
+ import { UserType } from '../../types/index';
+ import UserContextProvider from "../../contexts/userContextProvider.tsx";
+
+ interface MyProps {
+     user: UserType;
+ }
+
+ //the user item
+ export default function MineralItem({user}: MyProps){
+     return (
+         <UserContextProvider>
+             <div>
+                 <div>
+                     <h3>{user.username}</h3>
+
+                     <div>
+                         <h3>About the user</h3>
+                         <h3>{user.about}</h3>
+                     </div>
+                 </div>
+             </div>
+         </UserContextProvider>
+     );
```

Fig 102 Fig 103

5.6 Sprint 4

5.6.1 Goals

For Sprint for I the goals were to get the full functionality for the accounts page. This means that the goals for the fourth sprint are the account edit page, the account delete functionality, the individual account page and I while not a goal I did some work on the puzzle create page.

Sprint 4

5.6.2 The individual Account page

This page started the creation of the user component that was used to display the user or users' information. The layout for the account page was starts by getting all users (this was done under the idea that the data of all users would eventually need to be sorted when adding the ability to message each other) and then using that data to get the user that the page is about. The page will then check if the user has an image path and if they don't will set their image to a placeholder.

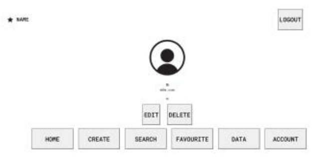
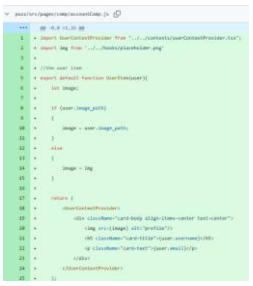
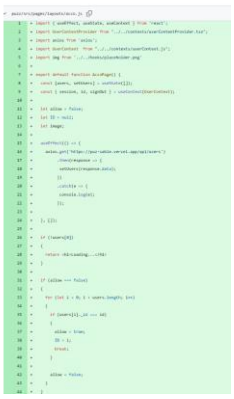


Fig 104



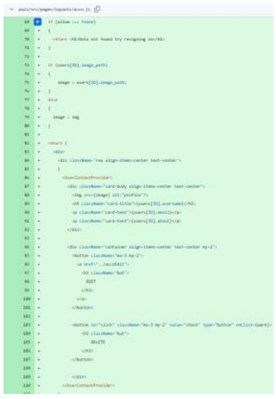
```
puzz/src/pages/comp/accountComp.js
@@ -0,0 +1,25 @@
+ import UserContextProvider from "../../contexts/userContextProvider.tsx";
+ import img from '../../hooks/placeholder.png'
+
+ //the user item
+ export default function UserItem(user){
+     let image;
+
+     if (user.image_path)
+     {
+         image = user.image_path;
+     }
+     else
+     {
+         image = img
+     }
+
+     return (
+         <UserContextProvider>
+             <div className="card-body align-items-center text-center">
+                 <img src={image} alt="profile"/>
+                 <h5 className="card-title">{user.username}</h5>
+                 <p className="card-text">{user.email}</p>
+             </div>
+         </UserContextProvider>
+     );
```

Fig 105



```
puzz/src/pages/layouts/acco.js
+ import { useEffect, useState, useContext } from 'react';
+ import UserContextProvider from "../../contexts/userContextProvider.tsx";
+ import axios from 'axios';
+ import UserContext  from "../../contexts/userContext.js";
+ import img from '../../hooks/placeholder.png'
+
+ export default function AccoPage() {
+   const [users, setUsers] = useState([]);
+   const { session, id, signOut } = useContext(UserContext);
+
+   let allow = false;
+   let ID = null;
+   let image;
+
+   useEffect(() => {
+    axios.get('https://puz-sable.vercel.app/api/users')
+         .then(response => {
+          setUsers(response.data);
+         })
+         .catch(e => {
+          console.log(e);
+         });
+
+   }, []);
+
+   if (!users[0])
+   {
+    return <h1>Loading...</h1>
+   }
+
+   if (allow === false)
+   {
+    for (let i = 0; i < users.length; i++)
+    {
+     if (users[i]._id === id)
+     {
+      allow = true;
+      ID = i;
+      break;
+     }
+
+     allow = false;
+    }
+   }
```

Fig 106

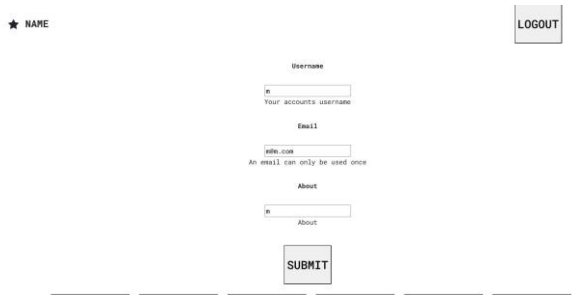


```
puzz/src/pages/layouts/acco.js
   if (allow === false)
+   {
+    return <h1>Data not found try resigning in</h1>
+   }
+
+   if (users[ID].image_path)
+   {
+     image = users[ID].image_path;
+   }
+   else
+   {
+    image = img
+   }
+
+   return (
+    <div>
+     <div className="row align-items-center text-center">
+      {
+       <UserContextProvider>
+        <div className="card-body align-items-center text-center">
+         <img src={image} alt="profile"/>
+         <h5 className="card-title">{users[ID].username}</h5>
+         <p className="card-text">{users[ID].email}</p>
+         <p className="card-text">{users[ID].about}</p>
+        </div>
+
+        <div className="container align-items-center text-center my-2">
+         <button className="mx-3 my-2">
+          <a href="../accoEdit">
+           <h3 className="but">
+            EDIT
+           </h3>
+          </a>
+         </button>
+
+         <button id="click" className="mx-3 my-2" value="check" type="button" onClick={users[
+           <h3 className="but">
+             DELETE
+           </h3>
+         </button>
+
+         </div>
+       </UserContextProvider>
```

Fig 107

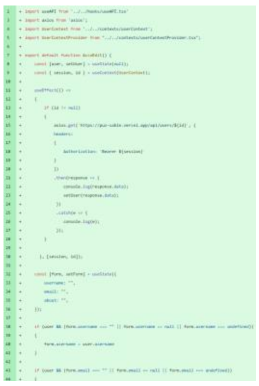
5.6.3 The edit Account page

This page lets the user edit their account info with exception for their password (and at the time their image). It was also be used as a blueprint for all future edit pages.



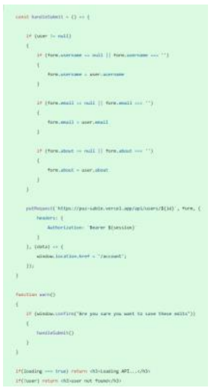
Username
m
Your accounts username
Email
m@m.com
An email can only be used once
About
m
About
SUBMIT

Fig 108



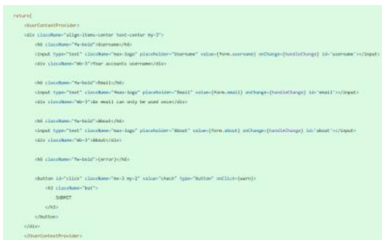
```
+ import useAPI from '../../hooks/useAPI.tsx'
+ import axios from 'axios';
+ import UserContext from '../../contexts/userContext';
+ import UserContextProvider from "../../contexts/userContextProvider.tsx";
+
+ export default function AccoEdit() {
+     const [user, setUser] = useState(null);
+     const { session, id } = useContext(UserContext);
+
+     useEffect(() =>
+     {
+         if (id != null)
+         {
+             axios.get(`https://puz-sable.vercel.app/api/users/${id}`, {
+             headers:
+             {
+                 Authorization: `Bearer ${session}`
+             }
+             })
+             .then(response => {
+                 console.log(response.data);
+                 setUser(response.data);
+             })
+             .catch(e => {
+                 console.log(e);
+             });
+         }
+
+       }, [session, id]);
+
+     const [form, setForm] = useState({
+         username: "",
+         email: "",
+         about: "",
+     });
+
+     if (user && (form.username === "" || form.username == null || form.username === undefined)){
+     {
+         form.username = user.username
+     }
+
+     if (user && (form.email === "" || form.email == null || form.email === undefined))
+     {
```

Fig 109



```
const handleSubmit = () => {

    if (user != null)
    {
        if (form.username == null || form.username === '')
        {
            form.username = user.username
        }

        if (form.email == null || form.email === '')
        {
            form.email = user.email
        }

        if (form.about == null || form.about === '')
        {
            form.about = user.about
        }
    }

    putRequest(`https://puz-sable.vercel.app/api/users/${id}`, form, {
        headers: {
            Authorization: `Bearer ${session}`
        }
    }, (data) => {
        window.location.href = '/account';
    });
}

function save()
{
    if (window.confirm("Are you sure you want to save these edits"))
    {
        handleSubmit()
    }
}

if(loading === true) return <h3>Loading API...</h3>
if(!user) return <h3>user not found</h3>
```

Fig 110

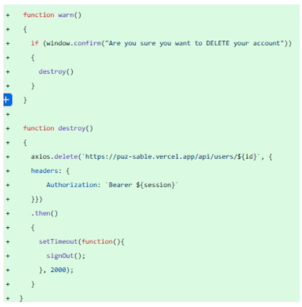


```
return(
    <UserContextProvider>
    <div className="align-items-center text-center my-3">
        <h6 className="fw-bold">Username</h6>
        <input type="text" className="max-logo" placeholder="Username" value={form.username} onChange={handleChange} id='username'></input>
        <div className="mb-3">Your accounts username</div>

        <h6 className="fw-bold">Email</h6>
        <input type="text" className="max-logo" placeholder="Email" value={form.email} onChange={handleChange} id='email'></input>
        <div className="mb-3">An email can only be used once</div>

        <h6 className="fw-bold">About</h6>
        <input type="text" className="max-logo" placeholder="About" value={form.about} onChange={handleChange} id='about'></input>
        <div className="mb-3">About</div>

        <h6 className="fw-bold">{error}</h6>

        <button id="click" className="mx-3 my-2" value="check" type="button" onClick={warn}>
            <h3 className="but">
                SUBMIT
            </h3>
        </button>
    </div>
    </UserContextProvider>
```

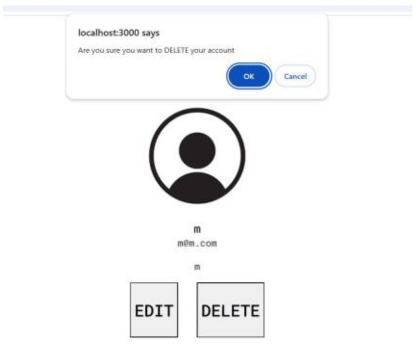
Fig 111

5.6.4 The delete Account function

The delete function is on the account page on is comprised of only twenty lines of code and its activator button and was only treated as a full goal due to the fact that like the edit page it will be reused across the entire project.



```
+   function warn()
+   {
+     if (window.confirm("Are you sure you want to DELETE your account"))
+     {
+       destroy()
+     }
+   }
+
+   function destroy()
+   {
+     axios.delete(`https://puz-sable.vercel.app/api/users/${id}`, {
+     headers: {
+         Authorization: `Bearer ${session}`
+     }})
+     .then()
+     {
+       setTimeout(function(){
+         signOut();
+       }, 2000);
+     }
+ }
```



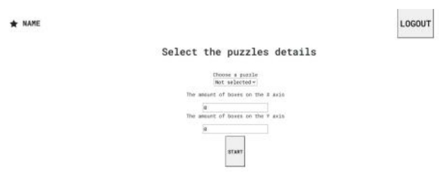
localhost:3000 says

Are you sure you want to DELETE your account

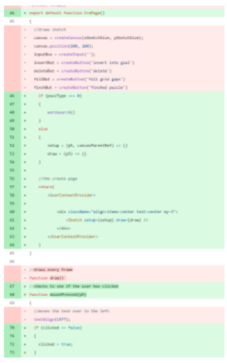
Fig 112 Fig 113

5.6.5 The start of the create puzzle page

The work that was done on the create page at during the fourth sprint was connecting the create wordsearch blueprint into react structure.



NAME
LOGOUT
Select the puzzles details
Choose a puzzle
Not selected
The amount of boxes on the X axis
0
The amount of boxes on the Y axis
0
START



```
+ export default function CrePage()
  {
-   //draws sketch
-   canvas = createCanvas(xSketchSize, ySketchSize);
-   canvas.position(300, 300);
-   inputBox = createInput('');
-   insertBut = createButton('insert into goal')
-   deleteBut = createButton('delete')
-   fillBut = createButton('fill grid gaps')
-   finshBut = createButton('finshed puzzle')
+     if (puzzType === 0)
+     {
+         wordsearch()
+     }
+     else
+     {
+         setup = (p5, canvasParentRef) => {}
+         draw = (p5) => {}
+     }
+
+     //the create page
+     return(
+         <UserContextProvider>
+
+             <div className="align-items-center text-center my-3">
+                 <Sketch setup={setup} draw={draw} />
+             </div>
+         </UserContextProvider>
+     )
  }

- //draws every frame
- function draw()
+ //checks to see if the user has clicked
+ function mousePressed(p5)
  {
-   //moves the text over to the left
-   textAlign(LEFT);
+   if (clicked == false)
+   {
+     clicked = true;
+   }
```

Fig 114 Fig 115

5.7 Sprint 5

5.7.1 Goals

The were a lot of goals that I had list for this sprint. These goals were finish the puzzles back end, the comment back end, get the create page to a point of functioning (for a word search), get the puzzle page to a point of functioning (for a word search) and allow puzzles to be edited and deleted.

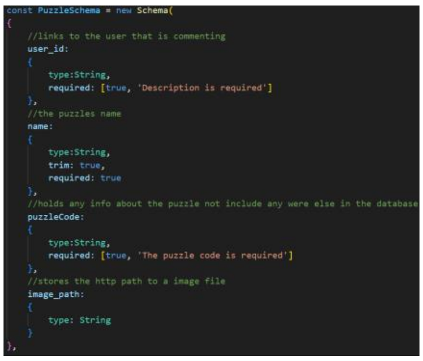
Sprint 5

5.7.2 Finish the puzzles back end

The back end was started back during the first sprint. The only reason it was not completed then was because at the time I was still trying to come up with a method of storing and reading the puzzles data. I ultimately ended up deciding to use a string.

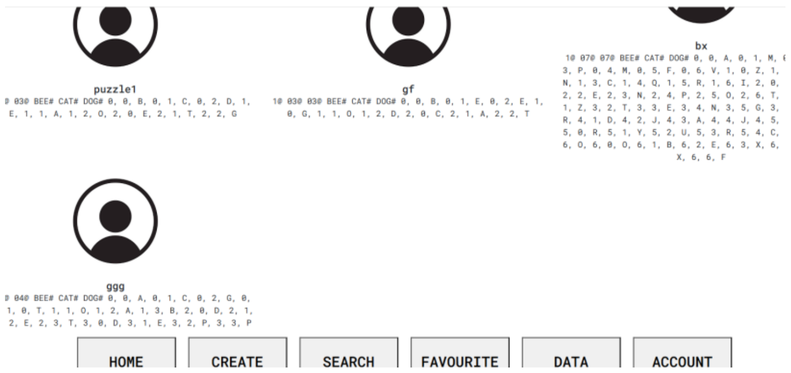
That string is split into four categories, the first category is the puzzle type (in the final forms of the application this can edit be a 1 for wordsearch or 2 for crossword), the next category is the grid size (the grid size is two number variable split from the puzzle type variable and each other by @ to tell the puzzle reader to move on to the next object).

The third category start after the third @ in the string, this category contains the goals of the puzzles (these are the object of each puzzle separated by #). Finally, after the last # there is the data that fills out the grid (e.g. the letters for wordsearch). This data is stored as its x grid position then y grid position and the data that is in that position.



```
const PuzzleSchema = new Schema(
{
    //links to the user that is commenting
    user_id:
    {
        type:String,
        required: [true, 'Description is required']
    },
    //the puzzles name
    name:
    {
        type:String,
        trim: true,
        required: true
    },
    //holds any info about the puzzle not include any were else in the database
    puzzleCode:
    {
        type:String,
        required: [true, 'The puzzle code is required']
    },
    //stores the http path to a image file
    image_path:
    {
        type: String
    }
},
```

Fig 116

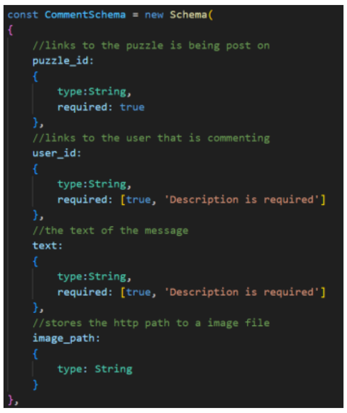


```
puzzle1
@ 03@ BEE# CAT# DOG# 0, 0, B, 0, 1, C, 0, 2, D, 1,
E, 1, 1, A, 1, 2, O, 2, 0, E, 2, 1, T, 2, 2, G

gf
1@ 03@ 03@ BEE# CAT# DOG# 0, 0, B, 0, 1, E, 0, 2, E, 1,
0, G, 1, 1, O, 1, 2, D, 2, 0, C, 2, 1, A, 2, 2, T

bx
1@ 07@ 07@ BEE# CAT# DOG# 0, 0, A, 0, 1, M,
3, P, 0, 4, M, 0, 5, F, 0, 6, V, 1, 0, Z, 1,
N, 1, 3, C, 1, 4, Q, 1, 5, R, 1, 6, I, 2, 0,
2, 2, E, 2, 3, N, 2, 4, P, 2, 5, O, 2, 6, T,
1, Z, 3, 2, T, 3, 3, E, 3, 4, N, 3, 5, G, 3,
R, 4, 1, D, 4, 2, J, 4, 3, A, 4, 4, J, 4, 5,
5, 0, R, 5, 1, Y, 5, 2, U, 5, 3, R, 5, 4, C,
6, O, 6, 0, O, 6, 1, B, 6, 2, E, 6, 3, X, 6,
X, 6, 6, F

ggg
@ 04@ BEE# CAT# DOG# 0, 0, A, 0, 1, C, 0, 2, G, 0,
1, 0, T, 1, 1, O, 1, 2, A, 1, 3, B, 2, 0, D, 2, 1,
2, E, 2, 3, T, 3, 0, D, 3, 1, E, 3, 2, P, 3, 3, P
```

Fig 117

5.7.3 Finish the comments back end

The comments back end was made originally under the idea that messages, and replies would be under their own branch of the database. During a later sprint however, I noticed that all three (comments, reply and messages) all had the same schema (functionally) and just I decided to just use comments for all of them (using puzzle_id as a receiver and user_id as a sender).

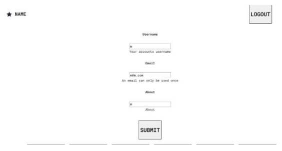


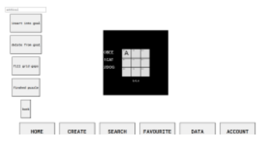
```
const CommentSchema = new Schema(
{
    //links to the puzzle is being post on
    puzzle_id:
    {
        type:String,
        required: true
    },
    //links to the user that is commenting
    user_id:
    {
        type:String,
        required: [true, 'Description is required']
    },
    //the text of the message
    text:
    {
        type:String,
        required: [true, 'Description is required']
    },
    //stores the http path to a image file
    image_path:
    {
        type: String
    }
},
```

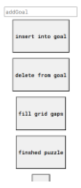
Fig 118

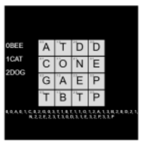
5.7.4 The create page

The create page is split into two page the first where the puzzles details are added and the second where the puzzle is named and given an image (added during a later sprint).









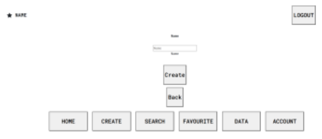
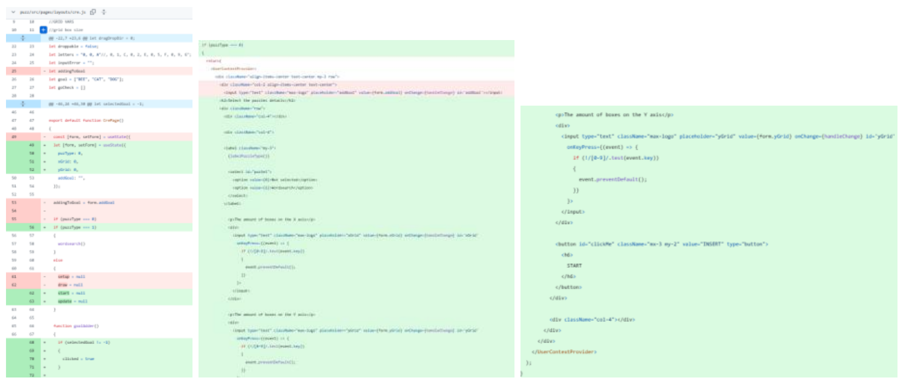


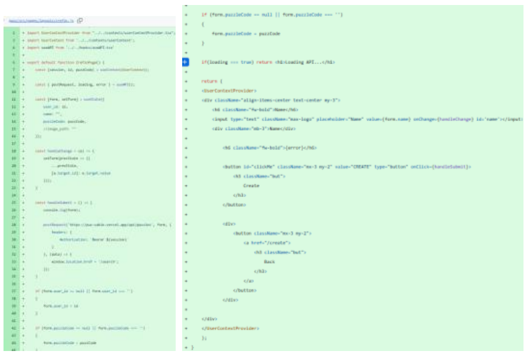
Fig 119 Fig 120 Fig 121 Fig 122

The detail page starts with asking the user what type of puzzle (during this sprint it was only word searches) do they want to make and what do they want the grid size to be. When it comes to creating a word search after this sprint was finished the grid would start with a “A” in the 0, 0 grid and no goals.

To the left of the grid there is a list of buttons. At the top of this list is an input text bar with add to goal inside. This text bar is how the user creates new goals (simply put in the goal that you would have wanted and hit the insert into goal button (the one right below it)). Having inserted a goal, it should then have appeared within the left border of the wordsearch.

Once a goal was inside the border it can be selected. A selected goal could have been place within the grid or deleted by pressing the delete from goal button. The third button is labelled fill grid gaps this button fills in any blank spaces that are within the grid with random letters. The button after called finish puzzle checks that all goals are in the grid and that all grid spaces are filled and if they are takes you to the second create page.



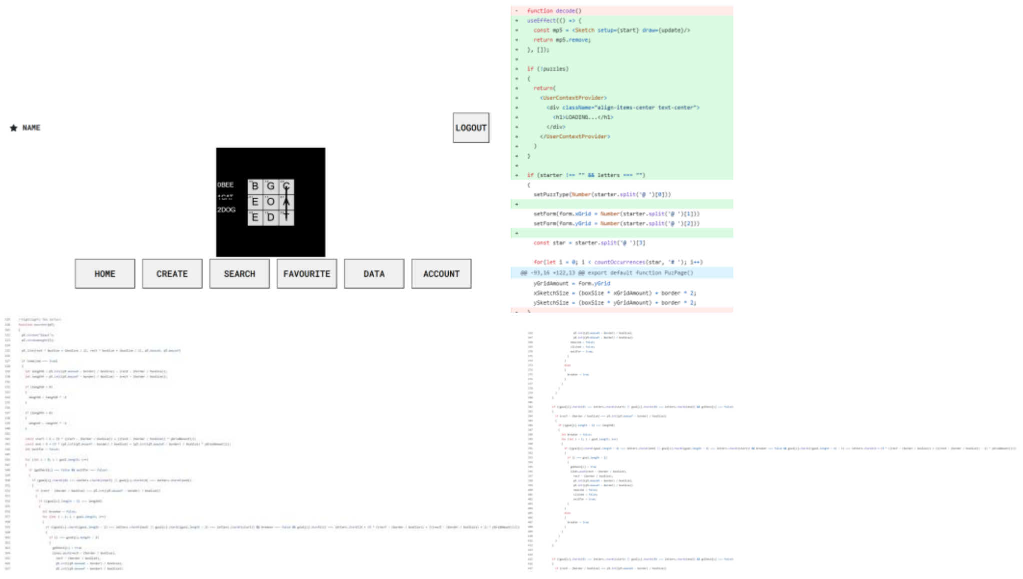


```
[illegible]
```

```
if (form.puzzleCode == null || form.puzzleCode === "")
{
    form.puzzleCode = puzzCode
}

if(loading === true) return <h1>Loading API...</h1>

return (
<UserContextProvider>
<div className="align-items-center text-center my-3">
    <h6 className="fw-bold">Name</h6>
    <input type="text" className="max-logo" placeholder="Name" value={form.name} onChange={handleChange} id='name'></input>
    <div className="mb-3">Name</div>

        <h6 className="fw-bold">{error}</h6>

        <button id="clickMe" className="mx-3 my-2" value="CREATE" type="button" onClick={handleSubmit}>
            <h3 className="but">
                Create
            </h3>
        </button>

        <div>
            <button className="mx-3 my-2">
                <a href="/create">
                    <h3 className="but">
                        Back
                    </h3>
                </a>
            </button>
        </div>

</div>
</UserContextProvider>
);
}
```

5.7.5 The puzzle page

The puzzles page starts by decoding the puzzles data then using it to construct the puzzle (during this sprint only a word search). The only part of the puzzle rendering code that is not copied from the create page was the crossing out of the goals and grid spaces when clicked. Even the detection code was copied from the grid check from the create page.



```
- function decode()
+ useEffect(() => {
+   const mp5 = <Sketch setup={start} draw={update}/>
+   return mp5.remove;
+ }, []);
+
+ if (!puzzles)
+ {
+   return(
+     <UserContextProvider>
+       <div className="align-items-center text-center">
+         <h1>LOADING...</h1>
+       </div>
+     </UserContextProvider>
+   )
+ }
+
+ if (starter !== "" && letters === "")
  {
    setPuzzType(Number(starter.split('@ ')[0]))
+
    setForm(form.xGrid = Number(starter.split('@ ')[1]))
    setForm(form.yGrid = Number(starter.split('@ ')[2]))
+
    const star = starter.split('@ ')[3]

    for(let i = 0; i < countOccurrences(star, '# '); i++)
@@ -93,16 +122,13 @@ export default function PuzPage()
    yGridAmount = form.yGrid
    xSketchSize = (boxSize * xGridAmount) + border * 2;
    ySketchSize = (boxSize * yGridAmount) + border * 2;
- }
```

```
[illegible]
```

5.7.6 Allow puzzles to be edited and deleted

I ran out of time during this sprint to finish the edit puzzle a page show I did not finish this goal.

5.8 Sprint 6

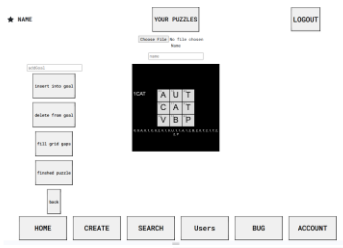
5.8.1 Goals

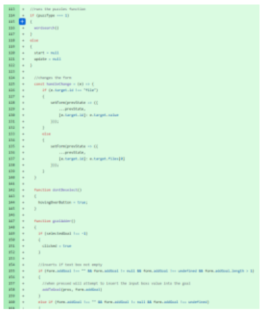
For Sprint 6 the goals were to implement the edit and delete functions for puzzles, the ability to comment on puzzles and made the bug report, message and reply back-end code.

Sprint 6

5.8.2 Allow puzzles to be edited and deleted

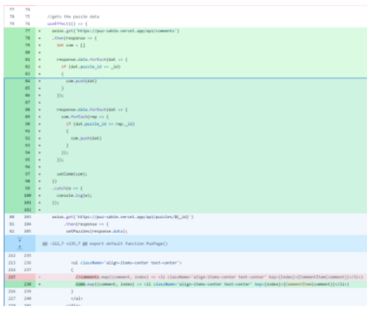
The delete puzzle feature was just a copy of the delete user code found on the account page while the edit puzzle page was created by combining both create pages together and calling the puzzle from the database to be edited.

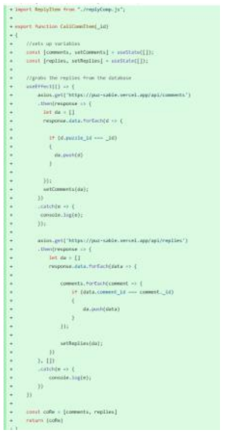




5.8.3 Comment on puzzles

Comments are texts and image that are posted onto each individual puzzle. Comments can be posted by any user doing the puzzle.

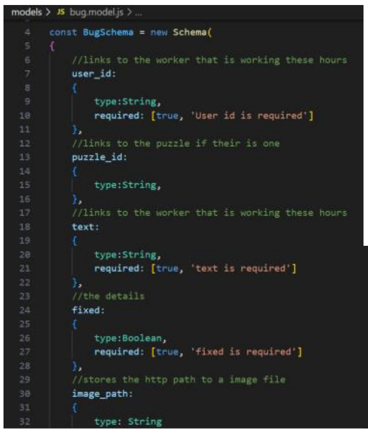




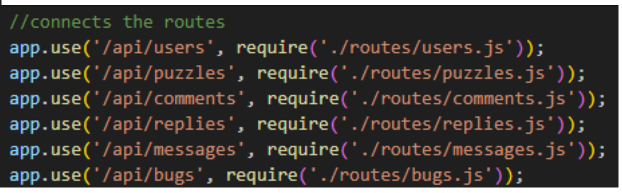
```
+ import ReplyItem from "./replyComp.js";
+
+ export function CallCommItem(_id)
+ {
+     //sets up variables
+     const [comments, setComments] = useState([]);
+     const [replies, setReplies] = useState([]);
+
+     //grabs the replies from the database
+     useEffect(() => {
+         axios.get('https://puz-sable.vercel.app/api/comments')
+         .then(response => {
+           let da = []
+           response.data.forEach(d => {
+
+             if (d.puzzle_id === _id)
+             {
+               da.push(d)
+             }
+
+           });
+           setComments(da);
+         })
+         .catch(e => {
+          console.log(e);
+         });
+
+         axios.get('https://puz-sable.vercel.app/api/replies')
+         .then(response => {
+             let da = []
+             response.data.forEach(data => {
+
+                 comments.forEach(comment => {
+                     if (data.comment_id === comment._id)
+                     {
+                         da.push(data)
+                     }
+                 });
+
+                 setReplies(da);
+             })
+         }, [])
+         .catch(e => {
+             console.log(e);
+         })
+     })
+
+     const coRe = [comments, replies]
+     return (coRe)
+ }
```

5.8.4 Bug report, message and reply back-end code

As mentioned during a previous sprint message and replay have the same model as comment and that later, I would go on to delete them and have all their feature run through comments. As for bug reports, they have also had mostly the same data but with the addition of a Boleen to see if the bugs been fixed or not.



```
models > JS bug.model.js > ...
const BugSchema = new Schema(
{
    //links to the worker that is working these hours
    user_id:
    {
        type:String,
        required: [true, 'User id is required']
    },
    //links to the puzzle if their is one
    puzzle_id:
    {
        type:String,
    },
    //links to the worker that is working these hours
    text:
    {
        type:String,
        required: [true, 'text is required']
    },
    //the details
    fixed:
    {
        type:Boolean,
        required: [true, 'fixed is required']
    },
    //stores the http path to a image file
    image_path:
    {
        type: String
```



```
//connects the routes
app.use('/api/users', require('./routes/users.js'));
app.use('/api/puzzles', require('./routes/puzzles.js'));
app.use('/api/comments', require('./routes/comments.js'));
app.use('/api/replies', require('./routes/replies.js'));
app.use('/api/messages', require('./routes/messages.js'));
app.use('/api/bugs', require('./routes/bugs.js'));
```

5.9 Sprint 7

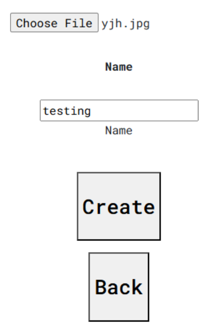
5.9.1 Goals

There was only one goal for sprint seven and that was to get images integrated with the front end. After images was integrated into the front end, I spent the rest of this sprint going through the code and add comments and fixing bugs.

[Sprint 7]

5.9.2 Images integrated with the front end

The register (give new user an image), account edit puzzle (edit current users image), create puzzle page (give a puzzle an image), edit puzzle page (edit a puzzles image) and puzzle page (create and edit a comments image) all require (during this sprint, later pages like bug report will need it too) the means of uploading an image to the front end then to send its data to the back end.

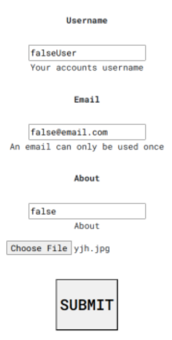


Choose File yjh.jpg

Name

testing

Name



Username

falseUser

Your accounts username

Email

false@email.com

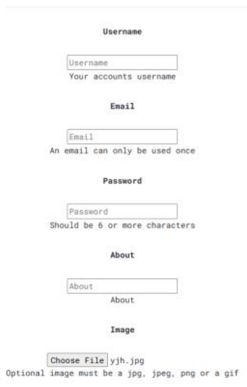
An email can only be used once

About

false

About

Choose File yjh.jpg



Username

Username

Your accounts username

Email

Email

An email can only be used once

Password

Password

Should be 6 or more characters

About

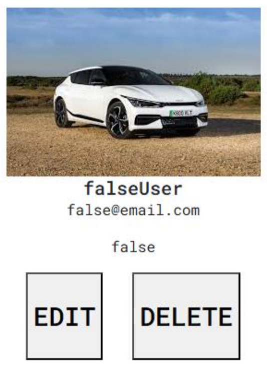
About

About

Image

Choose File yjh.jpg

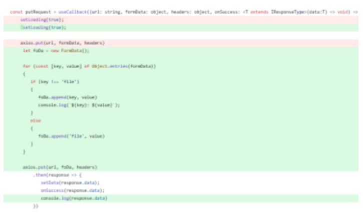
Optional image must be a jpg, jpeg, png or a gif



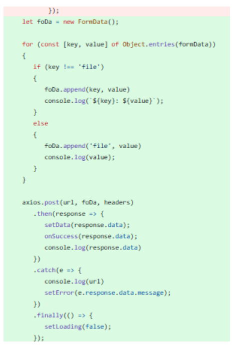
falseUser

false@email.com

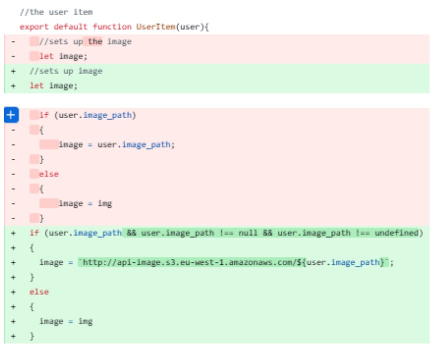
false



```
const putRequest = useCallback((url: string, formData: object, headers: object, onSuccess: <T extends IResponseType>(data:T) => void) =>
-    setLoading(true);
+    setLoading(true);

-    axios.put(url, formData, headers)
+    let foDa = new FormData();

     for (const [key, value] of Object.entries(formData))
     {
        if (key !== 'file')
        {
           foDa.append(key, value)
           console.log(`${key}: ${value}`);
        }
        else
        {
           foDa.append('file', value)
        }
     }

     axios.put(url, foDa, headers)
        .then(response => {
           setData(response.data);
           onSuccess(response.data);
           console.log(response.data)
        })
```



```
            });
    let foDa = new FormData();

    for (const [key, value] of Object.entries(formData))
    {
       if (key !== 'file')
       {
          foDa.append(key, value)
          console.log(`${key}: ${value}`);
       }
       else
       {
          foDa.append('file', value)
          console.log(value);
       }
    }

    axios.post(url, foDa, headers)
       .then(response => {
          setData(response.data);
          onSuccess(response.data);
          console.log(response.data)
       })
       .catch(e => {
          console.log(url)
          setError(e.response.data.message);
       })
       .finally(() => {
          setLoading(false);
       });
```



```
  //the user item
  export default function UserItem(user){
-    //sets up the image
-    let image;
+    //sets up image
+    let image;

-    if (user.image_path)
-    {
-       image = user.image_path;
-    }
-    else
-    {
-       image = img
-    }
+    if (user.image_path && user.image_path !== null && user.image_path !== undefined)
+    {
+       image = `http://api-image.s3.eu-west-1.amazonaws.com/${user.image_path}`;
+    }
+    else
+    {
+       image = img
+    }
```

5.10 Sprint 8

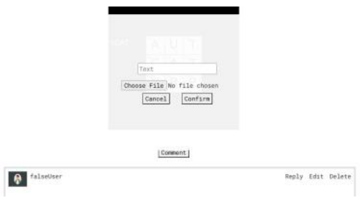
5.10.1 Goals

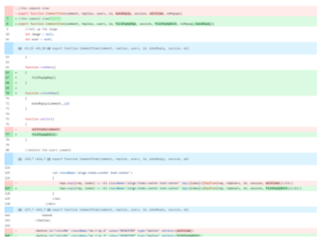
For the eighth Sprint the goals were comments (edit and delete), messages, users page and user page.

Sprint 8

5.10.2 Comments

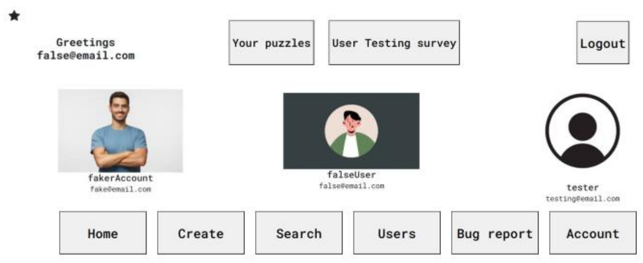
This was just finishing up the comment's functionality at the end of the puzzle page as messages are a near exact copy and paste of them. I also set up the ability for user to reply to other comments which is just the regular comment but with the comment id instead of the puzzles id.

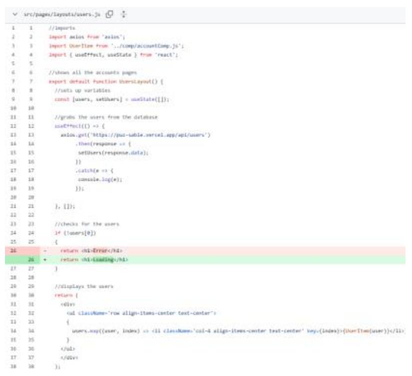




5.10.3 The users page

The users page is a copy of the puzzles search page but instead of getting all the puzzles it gets all the users.

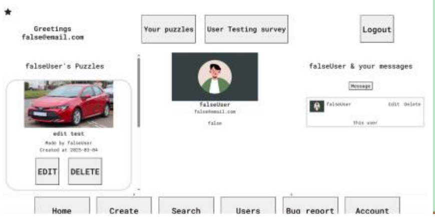


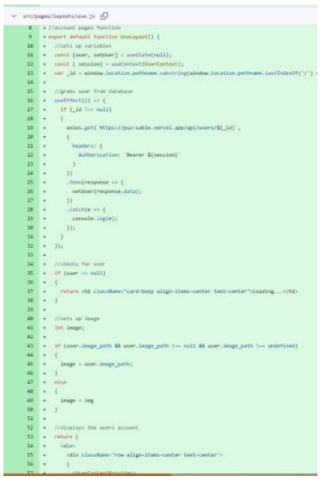


```
src/pages/layouts/users.js
  //imports
  import axios from 'axios';
  import UserItem from '../comp/accountComp.js';
  import { useEffect, useState } from 'react';

  //shows all the accounts pages
  export default function UsersLayout() {
    //sets up variables
    const [users, setUsers] = useState([]);

    //grabs the users from the database
    useEffect(() => {
      axios.get('https://puz-sable.vercel.app/api/users')
          .then(response => {
           setUsers(response.data);
          })
          .catch(e => {
           console.log(e);
          });

    }, []);

    //checks for the users
    if (!users[0])
    {
-     return <h1>Error</h1>
+     return <h1>Loading</h1>
    }

    //displays the users
    return (
      <div>
        <ul className='row align-items-center text-center'>
        {
          users.map((user, index) => <li className='col-4 align-items-center text-center' key={index}>{UserItem(user)}</li>)
        }
        </ul>
      </div>
    );
```

5.10.4 User page

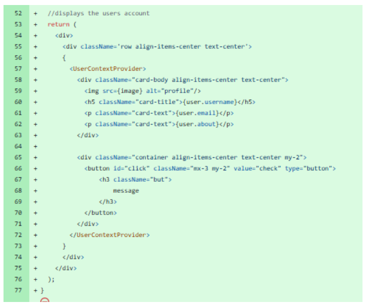
The user page started by being a copy of the previous sprint's account page with the edit and delete features removed. What the user page contains a scroll section that shows the users puzzles on the left of the page and on the right of the page there is another scroll section that shows would show your messages with that user. Both of these features would be added to the account page, but the message scroll would also have a user selector to pick which user you are messaging.





src/pages/layouts/use.js

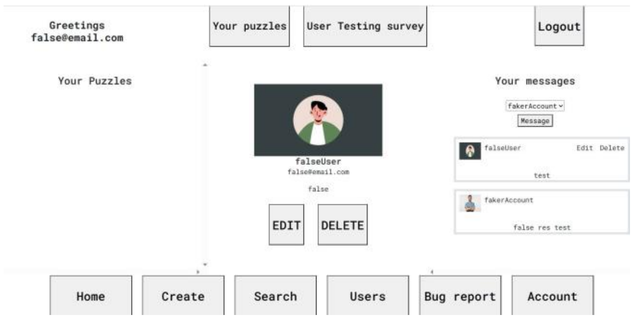
```
+ //account pages function
+ export default function UseLayout() {
+   //sets up variables
+   const [user, setUser] = useState(null);
+   const { session} = useContext(UserContext);
+   var _id = window.location.pathname.substring(window.location.pathname.lastIndexOf('/') +
+
+   //grabs user from database
+   useEffect(() => {
+     if (_id !== null)
+     {
+       axios.get(`https://puz-sable.vercel.app/api/users/${_id}`,
+       {
+         headers: {
+           Authorization: `Bearer ${session}`
+         }
+       })
+       .then(response => {
+         setUser(response.data);
+       })
+       .catch(e => {
+         console.log(e);
+       });
+     }
+   });
+
+   //checks for user
+   if (user == null)
+   {
+     return <h1 className="card-body align-items-center text-center">Loading...</h1>
+   }
+
+   //sets up image
+   let image;
+
+   if (user.image_path && user.image_path !== null && user.image_path !== undefined)
+   {
+     image = user.image_path;
+   }
+   else
+   {
+     image = img
+   }
+
+   //displays the users account
+   return (
+     <div>
+       <div className='row align-items-center text-center'>
+       {
```



```
+   //displays the users account
+   return (
+     <div>
+       <div className='row align-items-center text-center'>
+       {
+         <UserContextProvider>
+           <div className="card-body align-items-center text-center">
+             <img src={image} alt="profile"/>
+             <h5 className="card-title">{user.username}</h5>
+             <p className="card-text">{user.email}</p>
+             <p className="card-text">{user.about}</p>
+           </div>
+
+           <div className="container align-items-center text-center my-2">
+             <button id="click" className="mx-3 my-2" value="check" type="button">
+                 <h3 className="but">
+                     message
+                 </h3>
+             </button>
+           </div>
+         </UserContextProvider>
+       }
+       </div>
+     </div>
+   );
+ }
```

5.10.5 Messages

As just stated in the comments goal messages are nearly a copy of puzzle comments with the puzzle id being replace with another user's id and appears only in the use and account pages.



5.11 Sprint 9

5.11.1 Goals

The goals are for sprint 9 are bug reports, host the front end and a user testing survey.

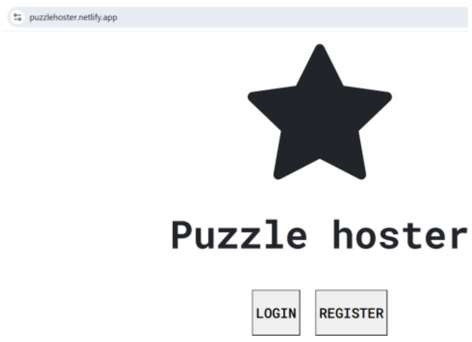
Sprint 9

5.11.2 Bug report

The bug reports page is a copy of the user's page but for the bug reports. The bug reports are a copy of the comments but with check to see if it's being fixed or not (it will say solved or unsolved at its top center of the bug report).

5.11.3 Host the front end

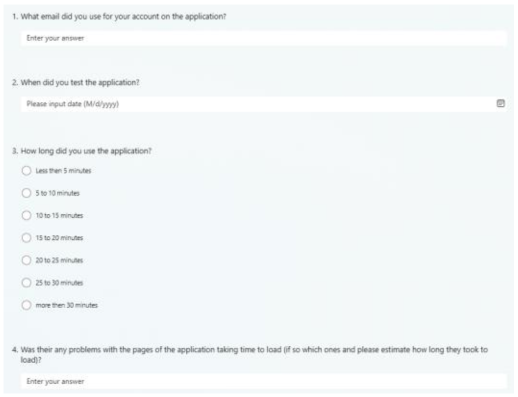
The front end was hosted on Netlify.



5.11.4 The survey

This survey was designed to discover problems with the application as it existed during this sprint and were I testers would be best put my effort in improving the application. The question for the project were 1 asking what email they made their account with (to find out about any bugs they discovered), 2 when did they test the application (encase they went to the application at a later date), 3 how long did they use the application for (to see how well it kept their attention).

Question 4 asked if they encounter any loading or latency problems (so I would know if there was any were that would require improvement in efficiency), 5 Asked if they had witnessed any bugs, 6 was about where they believe future development was needed and 7 was about what more specific features do they think should be focused on.



1. What email did you use for your account on the application?

Enter your answer

2. When did you test the application?

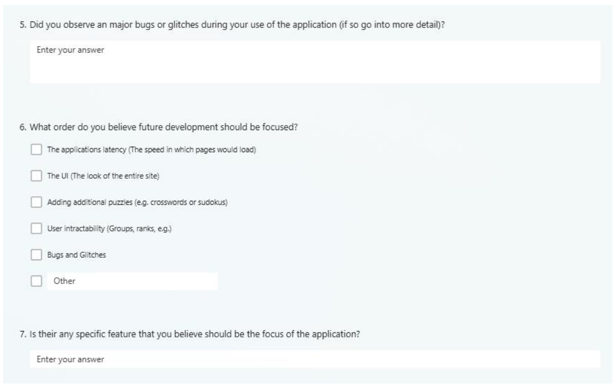
Please input date (M/d/yyyy)

3. How long did you use the application?

- ○ Less then 5 minutes
- ○ 5 to 10 minutes
- ○ 10 to 15 minutes
- ○ 15 to 20 minutes
- ○ 20 to 25 minutes
- ○ 25 to 30 minutes
- ○ more then 30 minutes

4. Was their any problems with the pages of the application taking time to load (if so which ones and please estimate how long they took to load)?

Enter your answer



5. Did you observe an major bugs or glitches during your use of the application (if so go into more detail)?

Enter your answer

6. What order do you believe future development should be focused?

- ☐ The applications latency (The speed in which pages would load)
- ☐ The UI (The look of the entire site)
- ☐ Adding additional puzzles (e.g. crosswords or sudokus)
- ☐ User intractability (Groups, ranks, e.g.)
- ☐ Bugs and Glitches
- ☐ Other

7. Is their any specific feature that you believe should be the focus of the application?

Enter your answer

5.12 Sprint 10

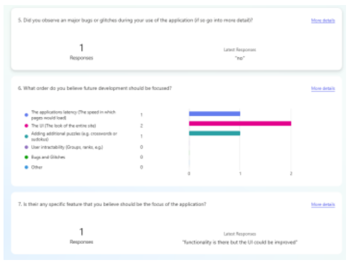
5.12.1 Goals

Sprint tens goals was the first stage of user testing and working on the UI.

Sprint 10

5.12.2 User testing

This user testing showed many problems that the application had the biggest and most likely least fixable was that it was very easy for multiple users to use up MongoDB's threshold limit, then theirs was a problem of testers not understanding what to do (e.g. one tester was sure what the pop up after clicking finish puzzle meant) and another problem was despite getting over 10 testers only two bothered to use the survey (though some of the testers could not log in because of the threshold problem).



5. Did you observe an major bugs or glitches during your use of the application (if so go into more detail)? More details

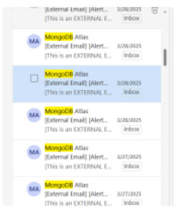
1 Responses — Latest Responses "no"

6. What order do you believe future development should be focused? More details

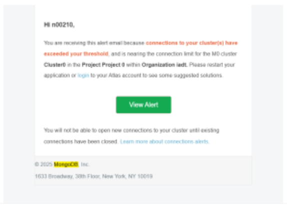
The applications latency (The speed in which pages would load)	1
The UI (The look of the entire site)	2
Adding additional puzzles (e.g. crosswords or sudokus)	1
User intractability (Groups, ranks, e.g.)	0
Bugs and Glitches	0
Other	0

7. Is their any specific feature that you believe should be the focus of the application? More details

1 Responses — Latest Responses "functionality is there but the UI could be improved"



```
[External Email] [Alert...   3/28/2025
[This is an EXTERNAL E...    Inbox
MongoDB Atlas
[External Email] [Alert...   3/28/2025
[This is an EXTERNAL E...    Inbox
MongoDB Atlas
[External Email] [Alert...   3/28/2025
[This is an EXTERNAL E...    Inbox
MongoDB Atlas
[External Email] [Alert...   3/28/2025
[This is an EXTERNAL E...    Inbox
MongoDB Atlas
[External Email] [Alert...   3/27/2025
[This is an EXTERNAL E...    Inbox
MongoDB Atlas
[External Email] [Alert...   3/27/2025
[This is an EXTERNAL E...    Inbox
```

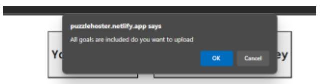


Hi n00210,

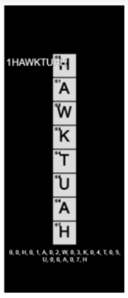
You are receiving this alert email because **connections to your cluster(s) have exceeded your threshold**, and is nearing the connection limit for the M0 cluster **Cluster0** in the **Project Project 0** within **Organization iadt.** Please restart your application or login to your Atlas account to see some suggested solutions.

You will not be able to open new connections to your cluster until existing connections have been closed. Learn more about connections alerts.

© 2025 MongoDB, Inc.
1633 Broadway, 38th Floor, New York, NY 10019



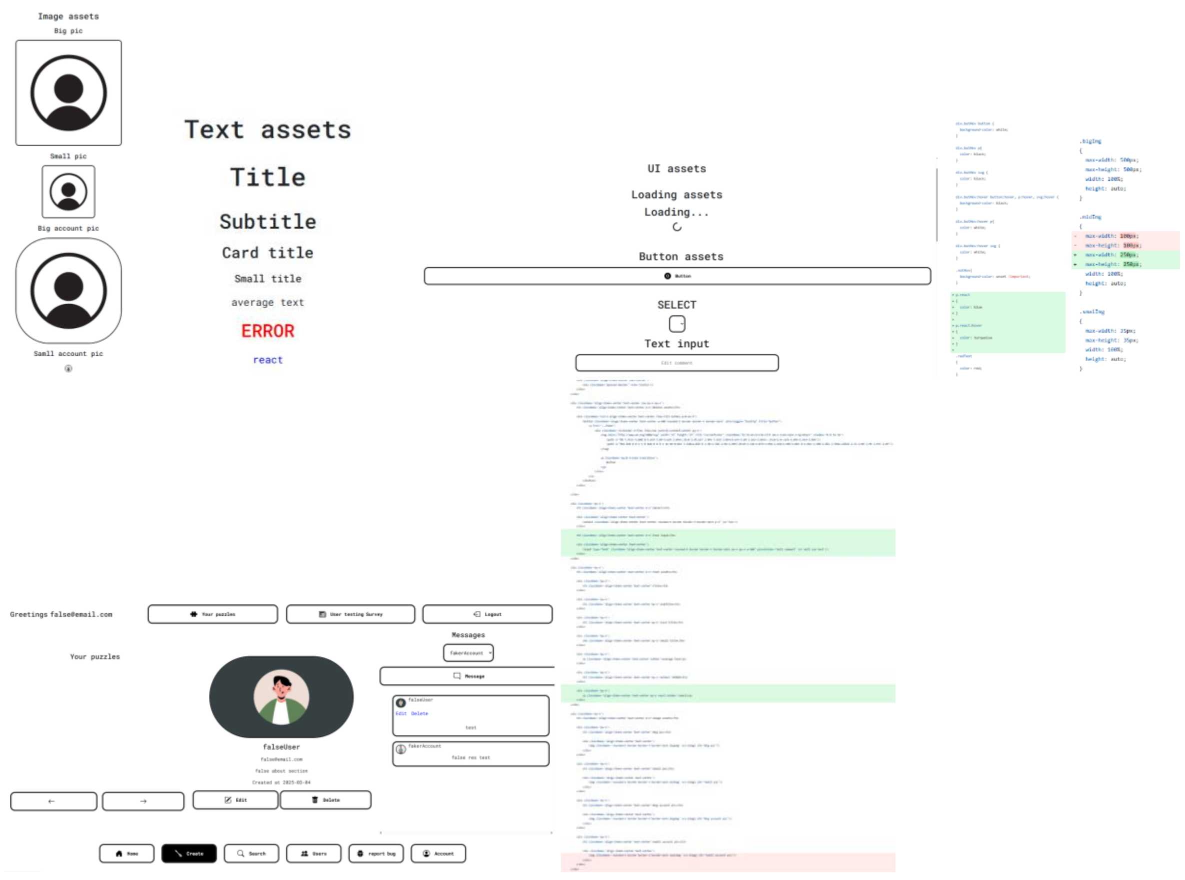
puzzlehoster.netlify.app says
All goals are included do you want to upload



0, 0, H, 0, 1, A, 0, 2, W, 0, 3, K, 0, 4, T, 0, 5, U, 0, 6, A, 0, 7, H

5.12.3 UI

The UI was design in this application to be basic, simple and minimalistic. This was started by making an assets page that would hold every single asset that the application has. Once these assets were created it was a matter of copying and replacing them over the old purely functional UI. However, on this sprint I had been unable to finish the UI for the create and puzzle pages.



5.13 Sprint 11

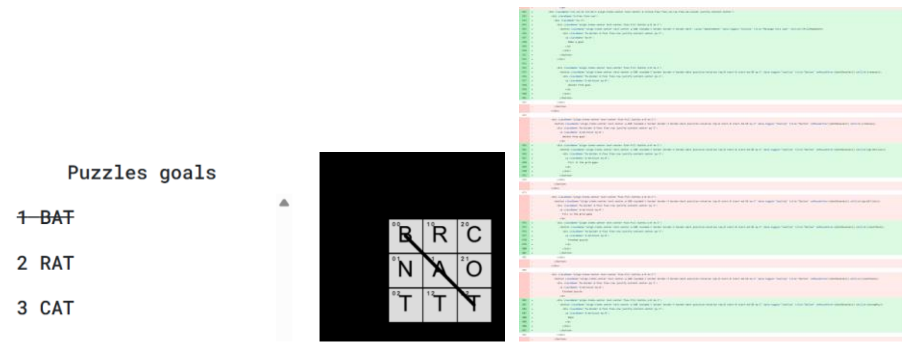
5.13.1 Goals

The eleventh sprint goals were to finish the UI and add the crosswords functionality.

[Sprint 11]

5.13.2 UI

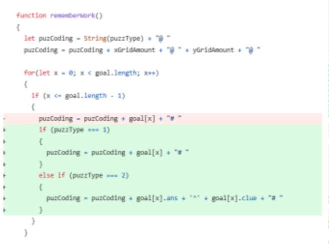
The UI change is continued from the previous sprint.



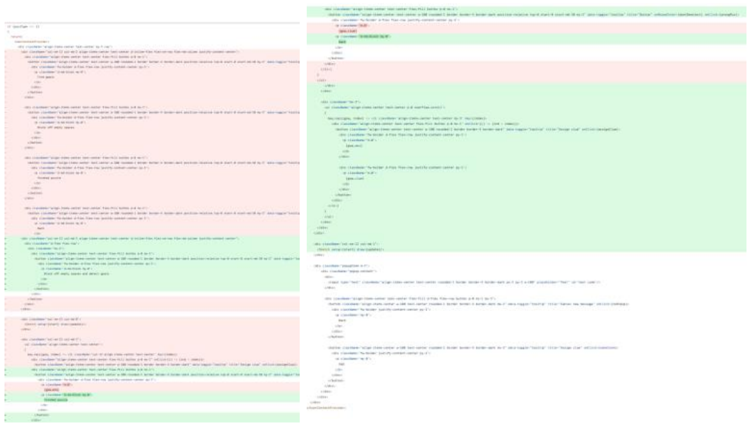
5.13.3 Crosswords

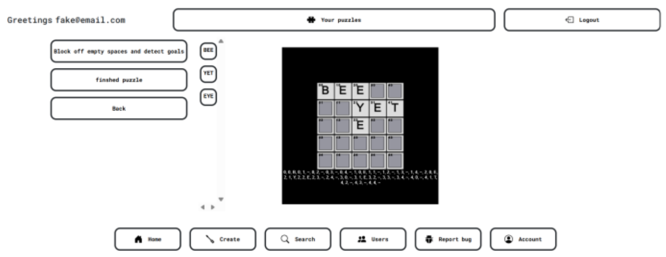
The crosswords systems are built on top of the word search code. The goals for example were altered from being a single string to hold the word searches goals to an object that holds to strings (answer and a clue). The buttons have been replaced with block off spaces and detect goal (which was when clicked will replace any black grid space with a block and converts every word in the crossword into new goals answer).

These goals will then need to select in order to input that words clue. Once all the clues are in the goal then the finish puzzle button can be pressed to check all the elements within the crossword and if they are fine will move on to the finish create page.



```
function rememberWork()
{
  let puzCoding = String(puzzType) + "@ "
  puzCoding = puzCoding + xGridAmount + "@ " + yGridAmount + "@ "

  for(let x = 0; x < goal.length; x++)
  {
    if (x <= goal.length - 1)
    {
-     puzCoding = puzCoding + goal[x] + "# "
+     if (puzzType === 1)
+     {
+       puzCoding = puzCoding + goal[x] + "# "
+     }
+     else if (puzzType === 2)
+     {
+       puzCoding = puzCoding + goal[x].ans + '^' + goal[x].clue + "# "
+     }
    }
  }
```





5.14 Sprint 12

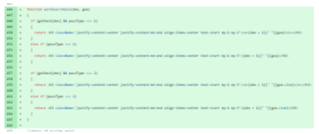
5.14.1 Goals

The goals of sprint 12 was to finish the crossword functionality and clean up the application.

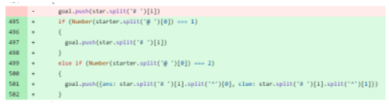
Sprint 12

5.14.2 Finish crossword functionality

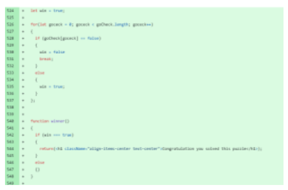
The way the crossword works in the puzzle page is that it shows the user the clue and every grid space that has an answer in it is just a black box that the user (using the letter input function from the create page) can try and guess the word. If they are right the clue will be crossed out.



```
function wordSearchGoal(des, goa)
{
  if (goCheck[des] && puzzType === 1)
  {
    return <h3 className='justify-content-center justify-content-md-end align-items-center text-start my-1 mp-3'><s>{des + 1}{" "}{goa}</s></h3>
  }
  else if (puzzType === 1)
  {
    return <h3 className='justify-content-center justify-content-md-end align-items-center text-start my-1 mp-3'>{des + 1}{" "}{goa}</h3>
  }

  if (goCheck[des] && puzzType === 2)
  {
    return <h3 className='justify-content-center justify-content-md-end align-items-center text-start my-1 mp-3'><s>{des + 1}{" "}{goa.clue}</s></h3>
  }
  else if (puzzType === 2)
  {
    return <h3 className='justify-content-center justify-content-md-end align-items-center text-start my-1 mp-3'>{des + 1}{" "}{goa.clue}</h3>
  }
}
```

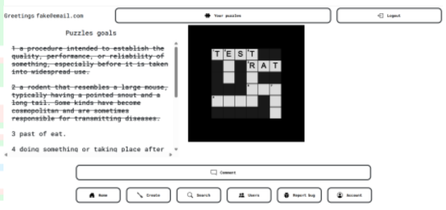


```
-       goal.push(star.split('# ')[i])
+       if (Number(starter.split('@ ')[0]) === 1)
+       {
+         goal.push(star.split('# ')[i])
+       }
+       else if (Number(starter.split('@ ')[0]) === 2)
+       {
+         goal.push({ans: star.split('# ')[i].split('^')[0], clue: star.split('# ')[i].split('^')[1]})
+       }
```



```
+ let win = true;
+
+ for(let goceck = 0; goceck < goCheck.length; goceck++)
+ {
+   if (goCheck[gocecк] == false)
+   {
+     win = false
+     break;
+   }
+   else
+   {
+     win = true;
+   }
+ };
+
+
+ function winner()
+ {
+   if (win === true)
+   {
+     return(<h1 className="align-items-center text-center">Congratulation you solved this puzzle</h1>);
+   }
+   else
+   {}
+ }
+
```

```
-  //the puzzle page
-  return(
+  //the puzzle page
+  return(
     <UserContextProvider>
       <div className="align-items-center text-center">
         <div className="row">
-          <div className="col-2">
+          <div className="col-sm-11 col-md-5">
+            <div className="mx-3">
+              <h3 className='align-items-center text-center my-3'>Puzzles goals</h3>

+              <ul className='align-items-center text-center overflow-scroll'>
+              {
+                goal.map((goa, index) => <li className='col-12 align-items-center text-center' key={index}>
+                  <div className='align-items-start text-start flex-fill p-0 ms-2'>
+                    <div className='fw-bolder d-flex flex-row'>
+                      {wordSearchGoal(index, goa)}
```



Greetings fake@email.com

Puzzles goals

~~1 a procedure intended to establish the quality, performance, or reliability of something, especially before it is taken into widespread use.~~

~~2 a rodent that resembles a large mouse, typically having a pointed snout and a long tail. Some kinds have become cosmopolitan and are sometimes responsible for transmitting diseases.~~

3 past of eat.

4 doing something or taking place after

5.15 Conclusion

The implementation of the application project went mostly smoothly such as the UI elements for the non-puzzle functionality parts being easily swapped in, the hosting to both Vercel and Netlify having pretty much no problems that couldn't be solve quickly and interaction between the back end and front end.

Areas of the application that were probably the biggest problems during implementation are the context taking far longer to develop then it should have MongoDB's threshold limit and wasting time trying to make replies and messages their own elements within the database only to write everything I did with them to use comments instead.

There were however quite a few features planned for the application that I just never got around to implementing. These features were ratings (the ability for users to rate puzzles), sudokus (the ability for users to make sudokus puzzles), favourites (user were going to have the power to favourite a puzzles) and settings (I had intended to grant users the ability to alter the size of the puzzles font, borders and grid boxes based on their like). The biggest reason for me not getting around to implementing these features is a combination of running out of time and spend too much of it on other areas of the project (e.g. wasting time on the messages and replies features and taking far too long getting the contexts to work).

6 Testing

6.1 Introduction

The two types of testing methods that this sections outlines (functional and user) both were carried out in different waves. Functional testing was performed as the application was coded and only on occasion or near the end was going back to retest functionality done. While with user testing there was about three main waves of user testing. The first held mostly only and the two after major implementation was done with only bug fixing in between them.

6.2 Functional Testing

Functional testing was performed in some form at every stage of the project. For the back end this was performed most with the aid of insomnia API. This allows for checking if a feature require a user's token or not, checking to see if a view all, view one, create, update and delete functions works as intended. As for the front-end testing all functionality testing was performed within the application.

6.2.1 Navigation

Test No	Description of test case	Input	Expected Output	Actual Output	Comment
1	Footer test	Press all buttons in the footer	All footer buttons take to their selected pages	All footer buttons take to their selected pages	Works
2	Create puzzle test	Create a puzzle, and solve it	Will got from create to create fin to puzzles, to puzzle	Went from create to create fin to create page	Not works
3	Create puzzle test 2	Create a puzzle, and solve it	Will got from create to create fin to puzzles, to puzzle	Will got from create to create fin to puzzles, to puzzle	Works
4	Edit puzzle test 1	Edit a puzzle	Go from your puzzles, pick a puzzle, edit it returned to your puzzles	Go from your puzzles, pick a puzzle, edit it returned to puzzles	Not worked

5	Edit puzzle test 2	Edit a puzzle	Go from your puzzles, pick a puzzle, edit it returned to your puzzles	Go from your puzzles, pick a puzzle, edit it returned to your puzzles	Works

6.2.2 Calculation

Test No	Description of test case	Input	Expected Output	Actual Output	Comment
1	Word search Puz code create	Create a Word search and solve it	Wordsearch is made and then is playable	Wordsearch is made and then is playable	Works
2	Word search Puz code edit	Edit a Word search and solve it	Wordsearch is edited and then is playable	Wordsearch is edited and then is playable	Works
3	Crossword Puz code create	Create a Crossword and solve it	Crossword is made and then is playable	Crossword is made and then is playable	Works
4	Crossword Puz code edit	Edit a Crossword and solve it	Crossword is edited and then is playable	Crossword is edited and then is playable	Works

6.2.3 CRUD

Test No	Description of test case	Input	Expected Output	Actual Output	Comment
1	Test user's functions in insomnia API	Try and use all functions within insomnia API	All functions run.	All functions run.	Works
2	Test puzzle's functions in insomnia API	Try and use all functions within insomnia API	All functions run.	All functions run.	Works
3	Test comment's functions in insomnia API	Try and use all functions within insomnia API	All functions run.	All functions run.	Works
4	Test bug's functions in insomnia API	Try and use all functions within insomnia API	All functions run.	All functions run.	Works
5	Test user's functions in the application	Try and use all functions within the front end	All functions run.	All functions run.	Works
6	Test puzzle's functions in the application	Try and use all functions within the front end	All functions run.	All functions run.	Works
7	Test comment's functions in the application	Try and use all functions within the front end	All functions run.	The message comments timed out the database	Not Works
8	Test comment's functions in the application	Try and use all functions within the front end	All functions run.	All functions run.	Works

9	Test bug's functions in the application	Try and use all functions within the front end	All functions run.	All functions run.	Works

6.2.4 Discussion of Functional Testing Results

The functional testing results were pretty good with the only ones needing to be retest being Create puzzle (sent the user back to the create page as appose to the puzzles page) and edit puzzle (brought users to the general puzzles search page instead of the user specific page) for Navigation and the message form of the comments function was being refreshed to frequently and was using up too much resources was discovered during the CRUD testing.

6.3 User Testing

As previous state user testing was split into three main waves. The first of these waves was mostly online with users trying to user the Netlify hosted front end and the survey to respond. The biggest problem that occurred during this testing wave was that the free MongoDB database I had linked to the application has a low threshold limit for its clusters. When this limit is reached the database shuts down for about half an hour and shuts down the entire site. This cause nearly half of all testers to struggle to even be able to sign in.

The next problem that was uncovered during this wave of testing was that a lot of users simply did not understand my descriptions. This to me meant that I should first reword all the descriptions and second not perform any more user testing until the UI is finalised. Another what I believe to be a decently big problem was that only two of the tests both to finish the survey.

Then came the second wave of user testing. This wave was done in person to watch how user navigated the site, to not have to rely on an online survey for people's opinions and to prevent everyone trying to join the application at once just not risking the database’s threshold. The main problem that was found during this wave testing was that users were still getting confused by the applications descriptions on the puzzle create page.

Other problems encountered during this wave of user testing was multiple UI errors such text turn white when the mouse is hovering in-between two buttons and so button icons disappearing during screen size scaling when they are not meant to. Other than that, most testers were able to easy find their way around the site.

So, after attempting to fix these problems I tried to once again test new testers for the final wave of testing. Like the previous wave of testers, they had little problems moving around the application however there was still some instances of tester pause and asking for confirmation on what so of the descriptions meant and also one last big bug was discovered were a test managed to put a letter into the blacked out grid space of a crossword puzzle making it impossible for them to complete the puzzle without resetting all their progress.

6.4 Conclusion

The testing of the application was successful in find problems the need to be fixed. Most commonly of which it was the minor UI bugs the mostly made by the buttons (hovering in between two buttons turning each both their text white and some buttons that are just icons with no text still having that icon turn invisible at the applications middle sizing). Then the next most common issue that was discovered about the application that need to be fix was the lack of understanding testers had for some of the descriptions.

However, most of those were found during user testing which leads me to believe that If I had time I probably would have been best to use it not implementing features I did not have time to include but instead to do another round to user testing as I am still not completely confident in the understandability of the create page.

All the while I don’t believe that there is any were near a risk of the functionality side have such issue that it’s testing would discover. Just I would not go back to do more testing for it if I had the time.

7 Project Management

7.1 Introduction

The main tools I used to keep track of this projects work was Miro and GitHub. GitHub was used solely for the code (along with moving the code around machines and linking up to the cloud sites). While Miro was used primarily for general / all work done on the project. This included but not is not limited to what happened during each sprint, what is in the backlog and in progress, the list of links that are associated with the project and the personas.

7.2 Project Phases

7.2.1 Proposal

There are four drafts of the project proposal (Draft1, Draft2, Draft3 and Draft4)). The proposals needed to be edited so many times because of the likes of bad grammar and poor explanations that was mostly removed over the multiple drafts.

7.2.2 Requirements

The requirements of this project for the project when broken down to what I believe is the simplest explanation is the ability to create, share, solve, edit and delete on multiple different puzzles. Whilst also allowing users to communicate with each other both privately and openly in the puzzles. So, the rest of the requirements phase became about trying to blueprint how to achieve them.

7.2.3 Design

A lot of the design phase was spent in Figma (figmaPage) where the first most basic draft of the applications front end UI was designed. Then the design phase was spent trying to get the application into a somewhat function state in which to try a further get an idea for what would be a more complete look for the finalized UI.

7.2.4 Implementation

The implementation phase came next, during which the second and third design framework for its UI was created. This was because as the application filled up, I got a better picture of were each piece of it was going to go exactly. As for the future development of the applications functionality during this phase was going from back end to front end (user back end then users front end). Until the back end was finished then focused fully on the front end.

7.2.5 Testing

The Testing phase of this project was three user testing waves. Wave one online relied on a survey only two of the testers used and showed major flaws with descriptions and the database threshold limit. This was then followed by quite a bit of development to try and fix these issues (the ones that could). The second wave of user testing was in person and was focus on how they moved around the application.

While the testers did navigate the site well, they encountered several bugs while doing so. So, after some more bug fixing the was the final wave of user testing while there was less bugs the was still confusion and some new bugs. As for functional testing this was pretty much done throughout the whole project as appose to limiting it to the testing phase.

7.3 SCRUM Methodology

There were 12 sprints involved in the project and I feel like that they went well with only few hiccups occurring during all of them (the user contexts taking longer to set up and wasting time making messages and replies in the back end only to end up using the comments for them instead).

Despite not having finished all that I wanted to get done for the project I do believe that I have manage to just about meet the requirements I had proposed.

There were six items that was still left in the backlog that I was unable to finish. The first three of them being to do with adding Sudoku as a third puzzle. Then the was two pages (settings and favourite) and a ratings system.

progress board

7.4 Project Management Tools

7.4.1 Miro

Miro is a design board made with the intended purpose of being used to aid in project management

For this project the Miro board kind of just became the storage space for everything that this project is linked to. Including the Figma board, the GitHub link form both the front and back-end, the research links and the projects sprints

7.4.2 GitHub

GitHub is a cloud-based platform design for the storing, transferring, sharing, managing and documenting of software. It was used as both a source of documenting the project code but also to transferer the code between computers and to their hosting site. GitHub function pretty much as I expected it to.

7.5 Reflection

7.5.1 Your views on the project

I think that the project wasn't handle badly but that I could have handled it a lot more efficiently at certain times with how there was several times I did things slower than I felt I should have been doing them (e.g. the user contexts) and that other times I seemingly went so much fast such as the words search blueprint. However overall, I think the project turned out pretty well.

7.5.2 Completing a large software development project

What I learned from the point of view of completing a large software development project is the importance of being able to accurately explain what you did and mean.

7.5.3 Working with a supervisor

I found that working with a supervisor to go pretty much as I suspected and overall was pretty much not surprised with how it turned out.

7.5.4 Technical skills

I learned from a technical skill-based viewpoint how to better manage a project that has multiple git repositories (back end and front end), several word documents (extended outline, the proposals, research report and the thesis report) and two design boards (Miro for project management and Figma for UI)

7.5.5 Further competencies and skills

I think I could do with learning better communication and describing tools so that others can better understand what I am trying to explain.

7.6 Conclusion

The project was about making a social media like application that is focus on the sharing of puzzles much like the puzzle you would find in a newspaper. Due to this the management of the project early on was focused on the functionality of the application with it's look and UI being one of the last things I started to implement into it.

This led to the project for most of the development looking like the bare minimum to function and as such when I got to testing the project I had little to no problems with the functionality testing but a ton with the users testing. As users the UI was for the more complicated features (the create puzzle) page probably need more work.

Ultimately, I believe that I managed this project is pretty ok, not the best but also not that poorly.

8 Conclusion

In conclusion, the project started out with the intended goals of being social media like platform for puzzles that you would most likely find within a newspaper. Such as world searches, crosswords and sudokus. Another goal should be that users should be able to comment on these puzzles and reply to these comments.
The technology I then tried to use this to achieve these projects was react.js for the front end and node.js for the basis of the back end. Both these code site would be stored, transferred and documented of GitHub. MongoDB was chosen as the database because it was always online that would link to the back end with mongoose. The back end then used express.js to link to the front end. The sprints were then be documented within Miro.

Researching for the project was done with a combination of partial comparison applications (a wordsearch creator to see how other site allow users to create word searches) and a research report (Research report) that looks into the other research papers on topics such as puzzle design, social media influence and encouraging and user generated content.

When it came to the based design of this project, I based a lot of it after other social media sites (mostly reddit) and started doing the base design in Figma (figma page). After the first draft was completed however, I then proceeded to go onto implementing the features of the application and only return to designing when I saw a clearer picture of what the final application was going to look like.

That implementation started off by switching between back end and front end. In the form of getting back-end user functionality then focusses on getting all the front-end user functionality. This was most how I went about things till I finished off the back end. From there I focus on sectioning of the front end (user page or comments) and focusing on that. The only time I would break this cycle was to test what was working.

When it comes to testing this project, I found that most of what I did was very functional with any mistake tending to be because I had inputted the wrong web page link or had forgotten to remove some styling code when copy and paste similar components to one another. While for the understanding of the site there was quite a problem of users having needed to ask what a button meant to do.

Overall, I think that the project turned out ok not as good as originally planned due to running out of time but also still functional. I think this that my management while not perfect it was more than enough to get most of the work done with little to no major bugs or glitches.

When it comes to what could be done to further the development of this project, the simplest would be to finish the back log (add sudokus, user rating system, a favourite and setting pages). along with Just more puzzles to bulk up the content the application has to offer. Also upgrading the database to one with a high threshold limit as well.

References

Miro Links

The whole miro board

Links stored in miro

Research links stored on miro

Resource report references

User generation content references
https://eprints.qut.edu.au/21206/

Bruns, A., & Bahnisch, M. (2009). Social media: Tools for user-generated content: Social drivers behind growing consumer participation in user-led content generation, Volume 1-State of the art.

Last accessed 05/12/2024

(Axel Bruns and Mark Bahnisch)

http://www.ladamic.com/papers/SecondLife/SocialInfluenceEC.pdf

Bakshy, E., Karrer, B., & Adamic, L. A. (2009, July). Social influence and the diffusion of user-created content. In *Proceedings of the 10th ACM conference on Electronic commerce* (pp. 325-334).

Last accessed 29/11/2024

(Eytan Bakshy, et al, 2009)

https://papers.ssrn.com/sol3/Delivery.cfm?abstractid=2549198

Luca, M. (2015). User-generated content and social media. In Handbook of media Economics (Vol. 1, pp. 563-592). North-Holland.

Last accessed 29/11/2024

(Micheal Luca, 2015)

Exploring consumer motivations for creating user-generated content

Daugherty, T., Eastin, M. S., & Bright, L. (2008). Exploring consumer motivations for creating user-generated content. *Journal of interactive advertising*, *8*(2), 16-25.

Last accessed 29/11/2024

(Terry Daugherty, et al, 2008)

https://www.tandfonline.com/doi/full/10.1080/09669582.2017.1372442

Han, W., McCabe, S., Wang, Y., & Chong, A. Y. L. (2018). Evaluating user-generated content in social media: an effective approach to encourage greater pro-environmental behavior in tourism?. *Journal of Sustainable Tourism*, *26*(4), 600-614.

Last accessed 04/11/2024

(Wei Han, et al, 2018)

Puzzles references
https://www.sciencedirect.com/science/article/abs/pii/S0747563214002672

Oei, A. C., & Patterson, M. D. (2014). Playing a puzzle video game with changing requirements improves executive functions. *Computers in Human Behavior*, *37*, 216-228.

Last accessed 05/12/2024

(Adam C. Oei and Michael D. Patterson 2014)

https://www.ericbutler.net/assets/papers/fdg2013_shortcuts.pdf

Smith, A. M., Butler, E., & Popovic, Z. (2013, May). Quantifying over play: Constraining undesirable solutions in puzzle design. In *FDG* (pp. 221-228).

Last accessed 05/12/2024

(Adam M. Smith, et al 2013)

Learning curves: analysing pace and challenge in four successful **puzzle** games

Linehan, C., Bellord, G., Kirman, B., Morford, Z. H., & Roche, B. (2014, October). Learning curves: analysing pace and challenge in four successful puzzle games. In *Proceedings of the first ACM SIGCHI annual symposium on Computer-human interaction in play* (pp. 181-190).

Last accessed 05/12/2024

(Connor Linehan, et al 2014)

https://link.springer.com/chapter/10.1007/978-3-642-34347-6_12

Danado, J., & Paternò, F. (2012). Puzzle: a visual-based environment for end user development in touch-based mobile phones. In *Human-Centered Software Engineering: 4th International Conference, HCSE 2012, Toulouse, France, October 29-31, 2012. Proceedings 4* (pp. 199-216). Springer Berlin Heidelberg.

Last accessed 05/12/2024

(Jose Danado and Fabio Paternò 2012)

https://link.springer.com/article/10.1007/s10639-015-9433-1

Xinogalos, S., Satratzemi, M., & Malliarakis, C. (2017). Microworlds, games, animations, mobile apps, puzzle editors and more: What is important for an introductory programming environment?. *Education and Information Technologies*, *22*, 145-176.

Last accessed 05/12/2024

(Stelios Xinogalos, et al 2017)

https://www.tandfonline.com/doi/abs/10.1080/00219266.2003.9655856

Franklin, S., Peat, M., & Lewis, A. (2003). Non-traditional interventions to stimulate discussion: the use of games and puzzles. *Journal of biological Education*, *37*(2), 79-84.

Last accessed 05/12/2024

(Sue Franklin, et al 2003)