Benjamin Wray - Web Application - DTP Yr 12

– Celeste Speedrunning Displayer –

- Video of App Walkthrough

https://github.com/BenjaminWray2008/Celeste_Times_App_Repository

Tools I used for the project:

Database Standard:

- Sqlite Studio
- Lucid Charts

Web Design Standard:

- Visual Studio Code
- Inspect tool Firefox (Debugging and checking on different browsers)
- Inspect tool Chrome (Debugging and checking on different browsers)
- Inspect tool Opera (Debugging and checking on different browsers)
- ChatGPT (used mostly to learn how to code stuff myself + a couple of queries and some css box shadows (I commented where))
- html formatter

Programming Standard:

- Visual Studio Code
- Flask Quickstart
- ChatGPT
- Flake8 (To pep8 my code)

Project Planning Phase (Sprint 0):

Celeste is a 2D platformer game.
The aim is for this project to display, and give users the ability to edit celeste speedrun times, so that they can catalogue their achievements, and goals.

- Spreadsheet example.

The below spreadsheet contains me and my friends times for celeste speedrunning specifically in the Any% (beat the game as fast as possible) category.

Any% Times

For this project, I would like to utilise the information within this spreadsheet, as well as other runners in the world, to create a web application that allows users to view their data, change their data, and compare their data to other runners in a format that is easier to use, and to see than a google spreadsheet. While the spreadsheet attached above only contains times from the any% category, the project end product will contain times for the categories of ARB (all red berries), 100%, True Ending, Bny%, and Cny%.

What is Celeste, and how does speedrunning it work?

Celeste is a game where the main character, Madeline, climbs Mount Celeste while overcoming various obstacles. Within the game there are 9 separate chapters, or sections of the game with different themes. 8 of these are part of the base game, and each contain an A, B, and C side. The A sides are the initial difficulty of the game, while the B sides are harder versions of the A side chapters that are unlocked via Cassette grabs. The C sides are the peak difficulty of the base game with each chapter C side containing 3 very difficult levels. C sides can only be unlocked after all crystal hearts have been collected (collectables hidden in A sides and at the end of B sides). The ninth chapter of the game is a relatively recently released, free to play DLC called Farewell that is a single very long chapter that contains the hardest difficulty in the game.

Speedrunning:
Being a movement based game, Celeste is basically a perfect speedrunning game. There are no RNG moments, only pure movement skill to pass through levels as fast as possible. There are lots of different speedrunning categories, which all represent completing different objectives within the game.

Any%:
The any% category is the main category within Celeste with the objective being to complete the base game as fast as possible. To beat the base game, the runner has

to beat the chapter 7 A side, called the summit. Upon reaching the top, the run is completed. The optimized route for speed in this category is to collect the chapter 5 and 6 cassettes and do 5b and 6b, which will be challenging to implement into my database.

ARB (All Red Berries):
Celeste has a bunch (175) of red berries (collectables) throughout the game. They can be collected by completing various puzzles, difficult rooms, or finding hidden rooms. The objective of an ARB run is to collect all 175 red berries. In terms of my project, this category will be one of the hardest. To collect all 175 berries, the runner has to enter the 8th chapter, Core, as it contains 4 berries. The issue with core is that to enter it, you need to have collected 4 crystal hearts. The optimal route for getting these hearts is to collect chapter 1, 3, and 5 blue crystal hearts, and then the 6a cassette, to complete 6b and get the red crystal heart at the end of that, as 6a doesn't have any berries. Because the runs section of my database is going to contain all checkpoint runs, I might have to implement a 'get crystal heart' checkpoint time into it.

100%:
This category is very simple. Do everything in the game. This will have every single different checkpoint time in it in the database.

True Ending:
Beat Farewell

Bny%:
Bny% is a category where the runner has to beat all the B sides from 1 to 8. They therefore have to collect all the B side cassettes in the A sides, and then beat the B sides. I'm not sure whether I will add a checkpoint called cassette grab, being a singular time per A side for this, or just split up the cassette grab time into the checkpoints that contain it.

Cny%:
Cny% is very simple. Runner activates cheat mode sequence to activate all C sides and completes them as fast as possible.

Requirements and Specifications:

Requirements	Specifications

User management	• Users can enter usernames to access that user's data. • Users can activate accounts to securely keep their times for only them to change.
Celeste speedrun attributes	• Website can support all different speedrun checkpoints or IL • Website can support different speedrun categories in Celeste • Website can store data for each user of their times in all these different checkpoints using a database (SQLite3)
Input Times	• Website can display a users current times in an easy to understand/navigate way • Website can allow users to edit their current times easily, and update the database on submission. This is done with html forms and flask logic.
Display Times	• Website can display any user's times in a logical manner. It can show relevant connections between times and comparisons to other runners
Security	• Websites keeps user's times secure so that only they can change them. Hashing and salt tables will be implemented for passwords, and user passwords will be stored safely in the flask server not in any urls using POST.
Usability	• Website will be easier to manage and will display times in an easier to understand manner so that it will be better than a google spreadsheet.
Technicalities	• Server display done with html/css • Server logic done with flask/jinja • Databasing done with SQLite3 • APIing/JSONing/AJAXing done

	with Javascript

Stakeholders:

Who is my project for?

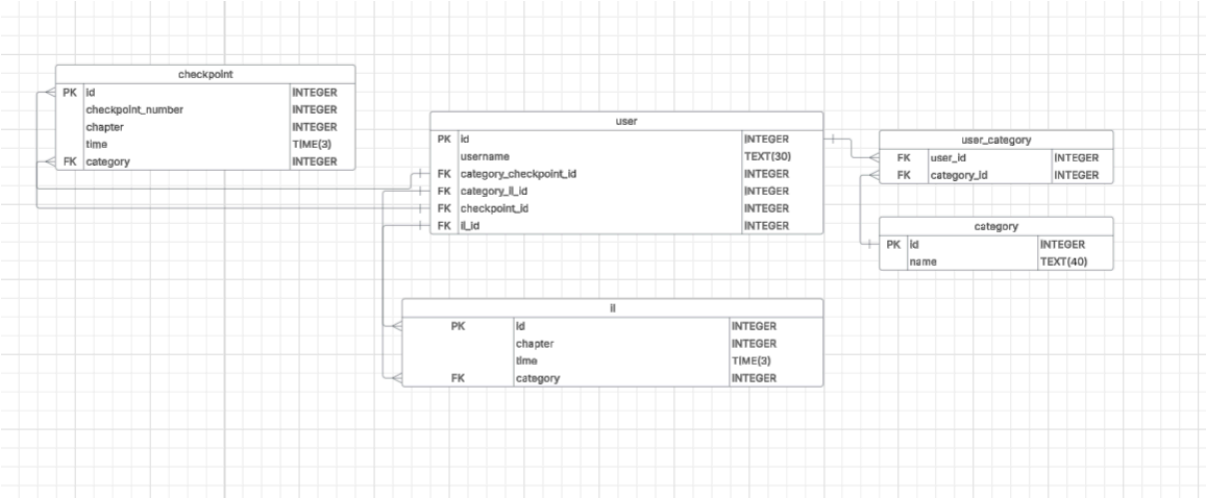
My project is for anyone who speedruns Celeste that is sick of using a google spreadsheet to document their times, and want a better displayed, and easier way of viewing, and changing their times.

Why am I making this project:
I am making this project because I speedrun Celeste and I am sick of using a google spreadsheet to document my times. I want a better display, and easier to change times for my times.

Throughout the creation of this project, I can get Ben Gorman, and Matthew Thoma to provide relevant feedback as they understand the game and speedrun it as well, and can thus help me make it better. Ben is also doing a similar project on Celeste except displaying different data.

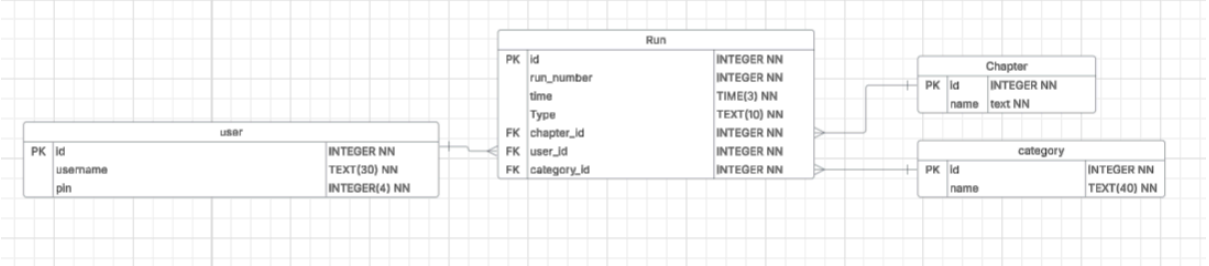
Database Design:

Initial ERD:



Although this ERD has all the functionality it needs, I think there is a better way to design it.

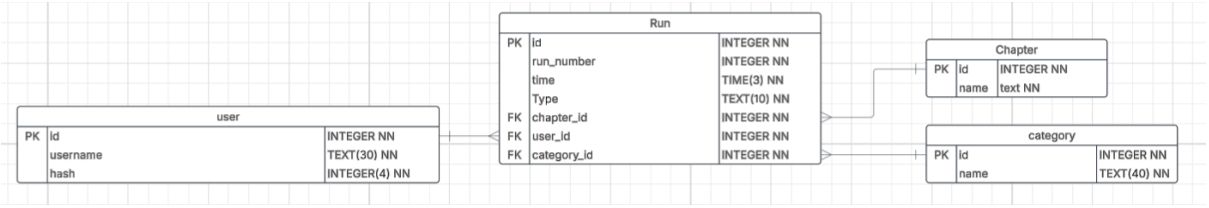
After some contemplation and help from one Ben Gorman, here is the *probably* final ERD product.



Although currently, the chapter table is somewhat unnecessary, as the chapter name could just be stored in the run table, there may be more variables that I need to add in the future. For example, I may need to add the amount of red berries within a chapter into the chapter table.

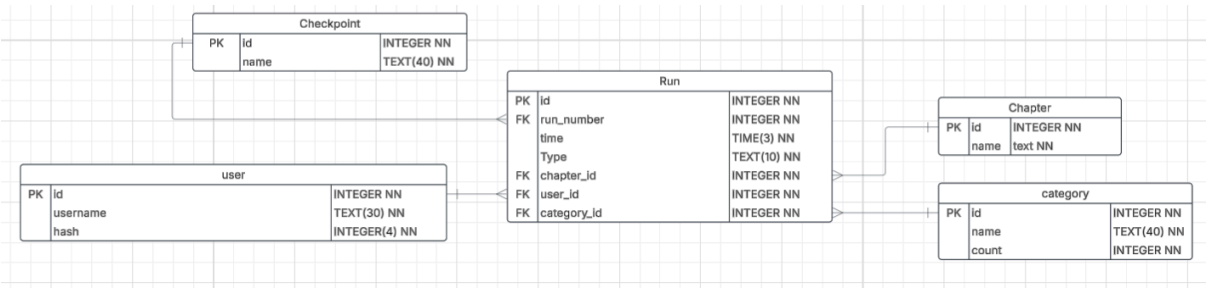
The many to many relationship within the table is the category to user relationship. A user can run multiple categories, and a category can be run by multiple users. The run table therefore, is the bridging table that holds every user's runs from all the different categories.

Small Update:



Changed ‘pin’ to ‘hash’ in the user table after I learnt about hashing passwords for security.

Bigger Update:



Added a checkpoint table and added a column in the category table to show how many different checkpoints in a specific category as this value is constant.

What queries am I going to need to use:

Queries + Expected results:

- Get user_id from an inputted username.
    - SELECT id FROM user WHERE username = ‘{username}’;

results=[id]

- Get all runs from an inputted username.
    - SELECT id,run_number,type,chapter_id,category_id FROM Run WHERE user_id = (SELECT id FROM user WHERE username={‘username’};
- Get all runs from an inputted username for a specific category.
    - SELECT id,run_number,type,chapter_id FROM Run WHERE user_id = (SELECT id FROM user WHERE username={username}) AND category_id = (SELECT id FROM category WHERE name = {inputted_category});

results=[id,run_number,time,type,chapter_id,category_id]

- Get all categories a user has times in.
    - SELECT name FROM category WHERE id IN (SELECT category_id FROM Run WHERE user_id = (SELECT id FROM user WHERE username = {username}));
- Get chapter name from chapter table.
    - SELECT name FROM Chapter WHERE id = chapter_id;

results=[name]

- Get category name from category table.
    - SELECT name FROM category WHERE id = category_id;

results=[name]

- Get a user pin for inputted username so you can check if it's correct.
    - SELECT hash FROM user WHERE username = {‘username’};

results=[pin]

- Get all times from all users for a specific run type.
    - SELECT time FROM Run WHERE Type=’{checkpoint/IL}’ and category_id={whatever category you looking for eg: 1 (any%)}

results=[1.354676,2353.25353,5332.345643,12.12345]

- Update Times from inputted times.
    - UPDATE Runs SET time = {inputted time} WHERE user_id={inputted user} AND category_id={inputted category} AND checkpoint_id = {checkpoint box that user inputted data into}

results=The database is now different. Yippee!

- When adding a new user, create every different time possible for that user.
    - INSERT INTO Runs (run_number,time,Type,chapter_id,user_id,category_id) VALUES ({inputted number, time, type, chapter_id, user_id, category_id);

results=The database is now different. Yippee!

- Add a new user
    - INSERT INTO user (name,hash) VALUES ({name}, {hashed_password});

results=The database is now different. Yippee!

These are all the main body queries that will be used to get data, but many other smaller queries that only get single columns of data will be used as well for efficiency.

Sample data:

Run table entry:
Run_number column denotes the checkpoint id Run_number=1 means that it's the first checkpoint in the game

id	run_number	time	type	chapter_id	user_id	category_id
1	1	15.003	‘checkpoint’	1	1	1

Corresponding chapter, user, and category tables:

Category table:

id	name
1	‘Any%’

User table:

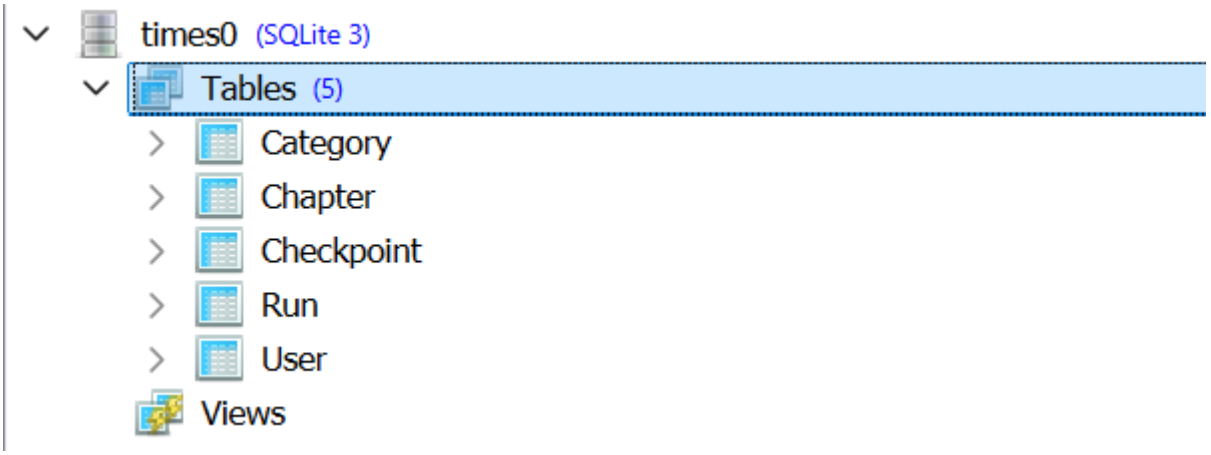
id	username	hash
1	‘John Smith’	EGIJJF#$%^iGIENF54kkefif$44

Chapter Table:

id	name
1	‘Forsaken City’

id	name

1	StartA



Yay! Database has been created!

Layout Design:

My Plan for this website is to have a simple, easy to navigate design that follows HCI heuristics. I don't want users to feel trapped, or find it difficult to locate data, so I'm going to focus on making the design look modern, and aesthetically pleasing.

Relevant Conventions I will use on my site:

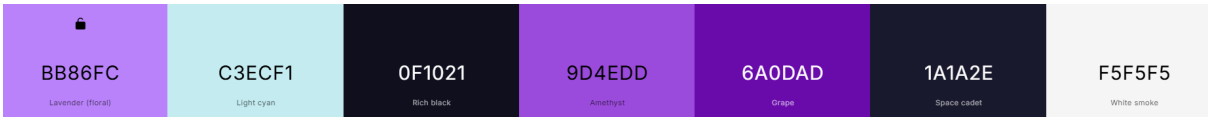
One thing I am going to do on my site is utilise flex design. This will follow the heuristic of Aesthetics and minimalist design as it will help me to layout the design simply and effectively. An example of how I will do this will be on my data entry times page. Here I will layout the input boxes simply so that the site is aesthetically pleasing for the user and is easy for them to find stuff they want.

Another heuristic I will follow will be consistency and standards, specifically external consistency. Most real websites will have their nav bar at the top of the page, and it will contain things like a signup / login button, a search bar, or an about page. They also have a footer that provides contact information etc. I will do this on my website too so that users can easily work out how to use the site, as it is consistent with other sites that have often.

Colour Scheme:

#1A1A2E - Dark Purple
#F5F5F5 - Smoky White
#C3ECF1 - Cyan
#6A0DAD - Purple
#9D4EDD - Purple
#BB86FC - Purple
#0F1021 - Dark Blue

https://coolors.co/generate/bb86fc-c3ecf1-0f1021-9d4edd-6a0dad-1a1a2e-f5f5f5



I think this selection of colours fits the celeste theme really well. I tried to make the colours similar to the colours in this image as they are what feel like represent the game to me.

Fonts:

https://fonts.google.com/specimen/Poppins
https://fonts.google.com/specimen/Nunito
https://fonts.google.com/specimen/VT323
https://fonts.google.com/specimen/Press+Start+2P
https://www.fontmagic.com/pixel-operator.font

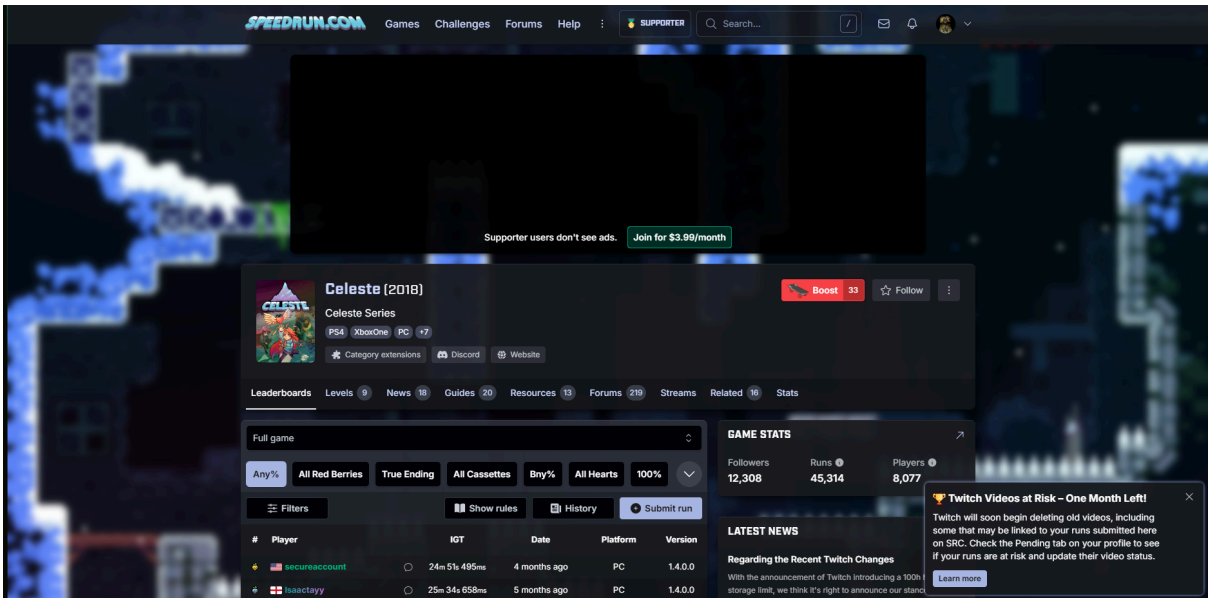
I think these fonts will be good. I can use the pixel-like fonts as headings, and the smoother fonts for text as they are more readable.

```
from flask import Flask, render_template
import sqlite3

with sqlite3.connect("times.db",check_same_thread=False) as database: #Connecting the database
    db=database.cursor()
    app=Flask(__name__)


    @app.route('/')
    def home():
        return render_template('home.html',title='Home')


if __name__ == '__main__':
    app.run(debug=True)
```

Initialisation of the application. Yippee!

Home Page:

I want the homepage of this website to really fit the Celeste theme.



Above is a picture of the speedrun.com page for Celeste.

Flask route and function signature:

```python
@app.route('/')
def home():
    return render_template('home.html', title='Home')

@app.route('/search', methods=['GET'])
def search():
    user_id='the id of the inputted username'
    return redirect(url_for('input_times',user_id=user_id, category_id=1))
```

This route runs when the user opens the application, hence the '/' file path.
The function is called home, because it runs when the home page is run. Upon the user searching their username, the form will submit with an action of running the /search route. This will then use the method of GET to redirect the user to the inputs times page, which contains arguments of the user_id, and a default category id to start on; 1.

Queries for the home page:
I will need to get a specific user's id from an inputted username in the /search route. This is because I don't want the user's name in the url; I want the user's id.

I will also need to check whether a user's password, after being hashed, is the same as the hashed password in the database.

After some thinking, I realised that I needed to edit this route a bit. Firstly, I forgot that this route also needed to check whether the password was correct, so I added that. This means that I don't want a GET request, otherwise the entered password would be displayed in the url. This would be bad, so I am now using POST. It was also very hard to read code commented stuff on the processes of this route, so I just wrote all the code instead.

The code gets the results of the form in the home page. It then gets the details of the user of the entered username. Then it checks if passwords match. Then it sends the user to input_times page if valid entries, profile page if only a username, or home page if nothing is correct.

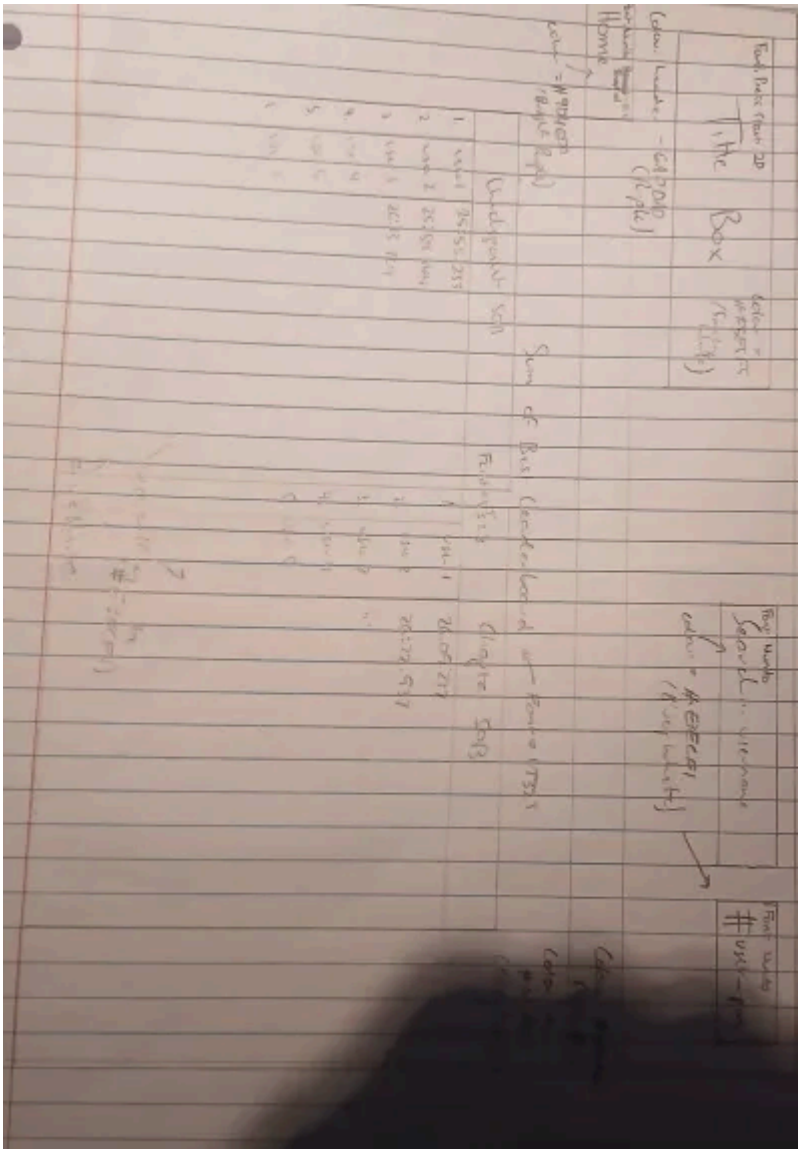
```python
@app.route('/search', methods=['POST'])
def search():
    username = request.form.get('username')
    password = request.form.get('password')

    db.execute(f'SELECT id, hash FROM user WHERE name = {username};')
    results=db.fetchone()
    if results:

        user_id, hash = results

        if check_password_hash(hash, password):
            return redirect(url_for('input_times',user_id=user_id, category_id=1))

        return render_template('profile.html', user_id = user_id, category_id = 1)

    print('No user found.')
    return render_template('home.html')
```

The only query that will need to be used here is (‘SELECT id, hash, from user WHERE name = ?;’, (username,) This will return a result of an id and a hash in a tuple, like so: (id, hash).

In the home page, the html form that will send this data will look like this.

```
{% extends 'layout.html' %}
{% block content %}
<form action="{{url_for('search')}}" method='POST'>
    <label for="username">Enter Username:</label>
    <input type="text" name="username" required>

    <label for="password">Enter Password:</label>
    <input type="text" name="password">

    <button type="submit">Submit!</button>
</form>
{% endblock %}
```

Initial Sketch:



Data Entry Page:

This page will hold all the forms that allow the user to edit their times. It will need to be compact, as there are a lot of different times.

After some research, I have found that the best way to do this is using an editable table. This is essentially an html structure that will have all the times for all the checkpoint in all the chapters for each category. Users will be able to simply click on the table, and then edit their time. To save data, there will be a save button, which will be more effective than saving after each edit as users may want to change a lot of different times at once.

There will then be a menu on a new page that opens after saving that allows users to select which category table they want to view, whether that be any%, ARB, etc.

For the overall layout and aesthetics of this page, I'm not going to try to make it look good, but be functional, as users don't want to be confused when on this page. Therefore, I'm not going to worry too much about the css for now.

Flask Route and Function Signatures:

As far as I understand, this will need two different app routes. This is because the application will first have to get all the times from the user of a specific category when the page is opened, and then when the save button is pressed, the application will run a new page that will post the updated data back to the database.

```
@app.route('/get_times/<int:user_id>/<int:category_id>', methods=['GET','POST'])
def input_times(user_id,category_id):

    category_id='the id of the category the user selected // Default is any%'
    #If the user has select a category change: Run this return statement
    return redirect(url_for('input_times',user_id=user_id, category_id=category_id))
    #Else, if the user has selected the submit button: Run this return statement
    times=#First collect all the times that are in the page ready to submit
    return redirect(url_for('update_times', user_id=user_id, category_id=category_id, times=times))
    #if neither of these conditions:
    return render_template('input_times.html', user_id=user_id, category_id=category_id)


@app.route('update_times/<int:user_id>/<int:category_id>', methods=['POST'])
def update_times(user_id,category_id):
    #get the times from the url
    #update the database
    return render_template('profile.html',user_id=user_id,category_id=category_id)
```

The first app route is called upon the user entering a username. It checks whether the user has submitted a category change, which would then redirect the user to the same page with a different category_id in the url. If the user has instead clicked on the submit button, they will be redirected to the route with the update times function, which will then update the database, and render the profile.html page, where the user can view their times. If neither of these conditions are met, the input_times page will be rendered for the user to enter time changes.

After some thinking, I decided to just code the data display page, because looking at a bunch of comments, inline return statements, and no html, I was becoming very confused.

```python
@app.route('/get_times/<int:user_id>/<int:category_id>', methods=['GET','POST'])
def input_times(user_id,category_id):
    db.execute('SELECT time FROM Runs WHERE id = ? and category_id = ?', (user_id, category_id))
    times = [row[0] for row in db.fetchall()]

    if request.method == 'POST':

        categories = []
        categories_list = ['Any%', 'ARB', '100%', 'True Ending', 'Bny%', 'Cny%', 'All Hearts']
        categories.append(request.form.get('Any%'))
        categories.append(request.form.get('ARB'))
        categories.append(request.form.get('100%'))
        categories.append(request.form.get('True_Ending'))
        categories.append(request.form.get('Bny%'))
        categories.append(request.form.get('Cny%'))
        categories.append(request.form.get('All_Hearts'))
        for i, j in enumerate(categories):
            if j:
                db.execute(f'SELECT id FROM category WHERE name = "{categories_list[i]}"')
                results=db.fetchone()
                category_id = results[0]
                db.execute(f'SELECT COUNT(*) FROM Runs WHERE category_id = {category_id} AND id = {user_id}')
                count = db.fetchone()
                session['current_count'] = count[0]
                return redirect(url_for('input_times',user_id=user_id, category_id=category_id))
        current_count = session.get('current_count')
    return render_template('get_times.html', user_id=user_id, category_id=category_id, current_count=current_count, times=times)
```

When the user runs this route, the server will get all of the user's times from the database, it will then get all the results from all of the category forms. If a form hasn't been clicked, they will each return None, and if they have been clicked then the name of that category will be added to the list. The server then loops through the list (categories) of all the results, and if it finds a value that isn't None, then it queries the database to find out how many checkpoints within that category, and then adds the value to the session. This session is helpful because it means I can store this data in the server, instead of the url. The server then redirects to the current page with a different category and count variable to display the amount of different checkpoint times to change on the actual page.

If all the values of the category forms are None, a user hasn't clicked on any of them. This means that the server can now render the template, as the current category_id must be the one the user wants.

```html
<form action="{{url_for('get_times', user_id=user_id, category_id=category_id)}}" method="GET">
    <label for="any%">any%</label>
    <input name="any%" type="submit">

    <label for="ARB">ARB</label>
    <input name="ARB" type="submit">

</form>

<form action="{{url_for('update_times', user_id=user_id, category_id=category_id')}}" method="POST">
    {{% for i in range(current_count) %}}
    <input type="text" name="checkpoints[]" placeholder="Enter Checkpoint Time:">
    {{% endfor %}}
    <button type="submit" name="submit_button">Submit</button>
</form>
```

Here is the html for the page. When the user clicks on an item in the category form, it runs the action of the app route 'get_data'. This then does the stuff mentioned

above. Otherwise, If the user clicks submit, it will run the update_times action, which will then update the database.

After even more thinking, I realise that this code is pretty terrible and isn't correct. Here is the revised code.

```python
@app.route('/get_times/<int:user_id>/<int:category_id>', methods=['GET','POST'])
def input_times(user_id,category_id):
    checkpoints = []
    db.execute('SELECT time, run_number FROM Runs WHERE id = ? AND category_id = ?', (user_id, category_id))
    times = [row[0] for row in db.fetchall()]
    checkpoint_id = [row[1] for row in db.fetchall()]
    for checkpoint in checkpoint_id:
        db.execute('SELECT name FROM checkpoint WHERE id = ?', (checkpoint,))
        checkpoints.append(db.fetchall())
    db.execute('SELECT name, count FROM categoty WHERE id = ?', (category_id,))
    name, count = db.fetchall()
    return render_template('get_times.html', user_id = user_id, category_id = category_id, times = times, checkpoints = checkpoints, count = count, name = name)
```

The main change here is that on the second revision of this page, I made the category selector a form. I realise now that this is stupid, and i can instead just make them links that send to the same page with different category id. This majorly simplifies everything.

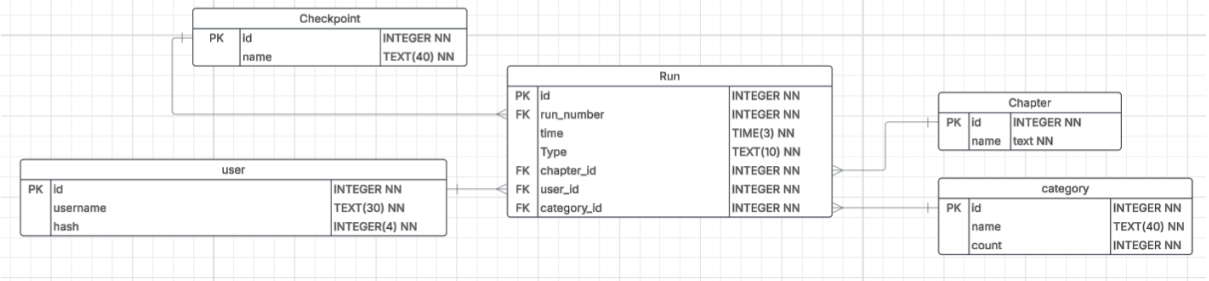
When the route runs, the program pretty much just does all the SQL queries required to get the data to display the page. The program runs the SQL queries of:
('SELECT time, run_number FROM Runs WHERE id = ? AND category_id = ?;', (user_id, category_id))
This query gets all the runs of a specific user of a specific category. Sample result will look like [(time, run_number), (time, run_number)], or with actual values, something like [(23456.3456, 43), (354.32353, 63)]
('SELECT name FROM checkpoint WHERE id = ?;', (checkpoint,))
This query gets all the names of all the checkpoints from all the checkpoints that were collected in the last query. Results will look like a single checkpoint name in a tuple. This is in a for loop, so it will collect all of them. Once appended to a list it will look something like this. [start1A, crossing1A, chasm1A, etc..]
('SELECT name, count FROM category WHERE id = ?;', (category_id,))
This query gets the name of the category to display, and the amount of data fields, called count, to create, so that the page can contain all the users time. Results will be a single tuple with data like (ARB, 65).

Then it renders it. Here is the html that shows this.

```
<a href="/get_times/{{user_id}}/1">any%</a>
<a href="/get_times/{{user_id}}/2">ARB</a>
<a href="/get_times/{{user_id}}/3">100%</a>

<form action="{{url_for('update_times', user_id=user_id, category_id=category_id')}}" method="POST">
    {{% for i in range(count) %}}
    <input type="text" name="checkpoints[]" placeholder="Enter Checkpoint Time:">
    <input type="hidden" name="any%" id="any%">
    {{% endfor %}}
    <button type="submit" name="submit_button">Submit</button>
</form>
```

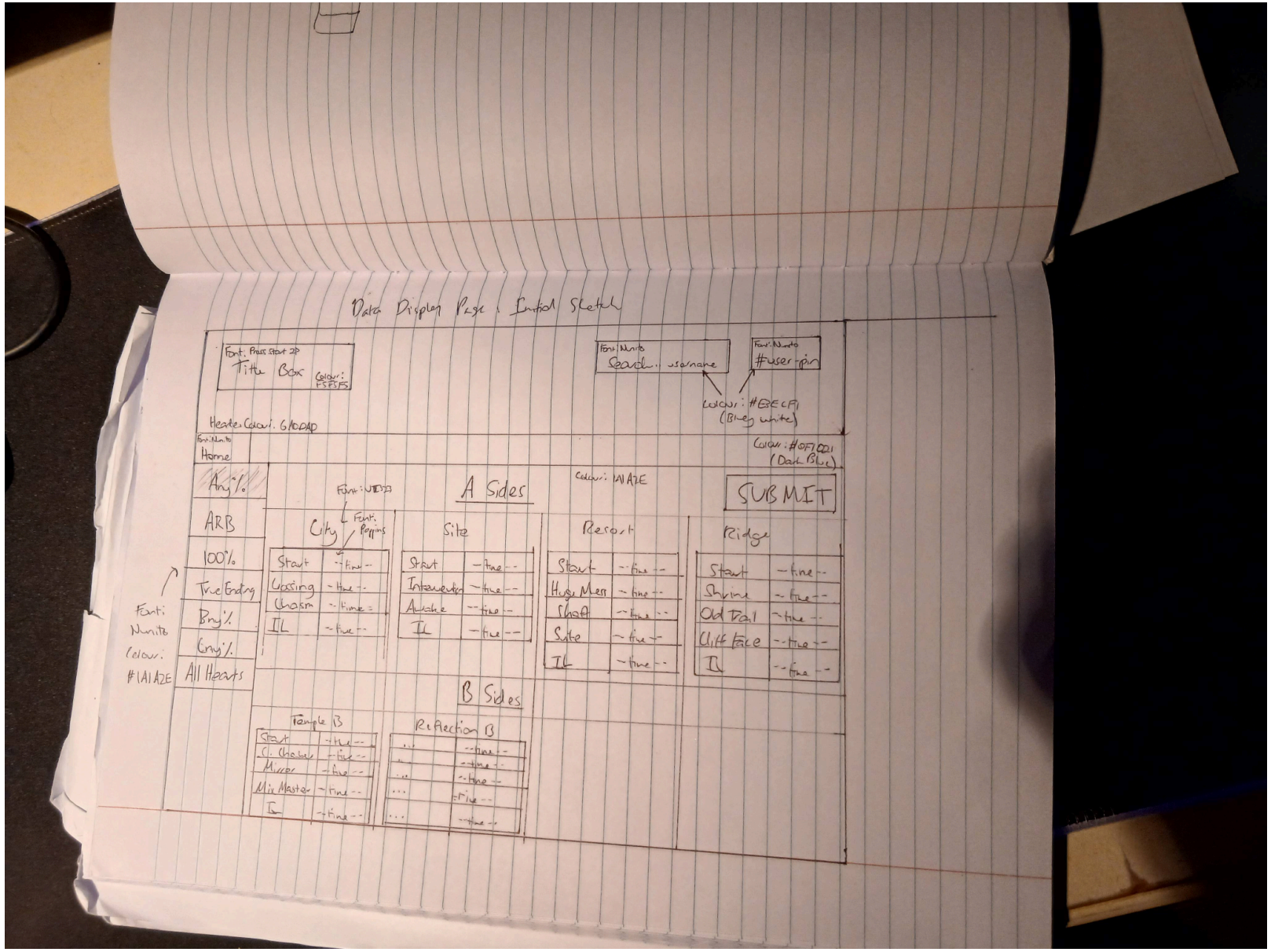
The anchor tags at the top are links to change the category you are on. The form below loops through how many different checkpoints in a certain category, then creates an input for each of these. It names them the name of that checkpoint, and then lets the user submit the form to get the results, and go to the update_times page.

These changes come with revisions with the database. Here is the revised database.



I have added a checkpoints table to get the name of each checkpoint, and also added a column to category to show how many different checkpoints in a category.

Initial Sketch:


Data Display Page : Initial Sketch
Font: Press Start 2P
Title Box
Colour: #5F5F5F
Header Colour: 6A0DAD
Font: Nunito
Home
Font: Nunito
Search.. username
Font: Nunito
#user#pin
Colour: #E3ECF1
(Blueish white)
Colour: #0F1021
(Dark Blue)
Any%
ARB
100%
True Ending
Any%
Any%
All Hearts
Font: Nunito
Colour: #1A1A2E
Font: VT323
A Sides
Colour: 1A1A2E
SUBMIT
City
Font: Pepsia
Site
Resort
Ridge
Start --time--
Crossing --time--
Chasm --time--
IL --time--
Start --time--
Intersection --time--
Autobahn --time--
IL --time--
Start --time--
Huge Mess --time--
Shaft --time--
Suite --time--
IL --time--
Start --time--
Shrine --time--
Old Trail --time--
Cliff Face --time--
IL --time--
B Sides
Temple B
Start --time--
Ch. Chaser --time--
Mixer --time--
Mix Master --time--
IL --time--
Reflection B
... --time--
... --time--
... --time--
... --time--
... --time--

Data Display Page:

This page will display all the data that the user submitted in the previous page. The aesthetic for this page needs to be very tidy, and easy to read, as it will contain different sections depending on which categories the user has submitted times in.

‘SELECT name FROM category WHERE id=(
SELECT category_id FROM Run WHERE user_id={current_user})’

This will return a list of the category names that a user runs in, which can then be formatted into sections of the data display page.

There is also the issue of how to layout this page. Within a specific category, there will be checkpoint times for each chapter, full chapter times, and I will need to get the program to calculate the sum of best times for all the checkpoints, and all the chapters. Since I don’t know how many checkpoints are going to be on a section of the page, as different categories have differing amounts of checkpoints, I’m going to have to make some kind of recursive design that can expand based on size.

As far as I can see, there are two options for how to do this.

Firstly, I could have an in-between page, where after clicking submit on the form, users will be sent to a page that lets them choose between which category they would like to see. This would be done with the GET method, as after clicking on, for example, the any% link, the application would send them to /profile/<str:user_id>/<int:category_id>. This would then give the id of the category to the profile page, which could be used to find all the users times for that category, and then display them.

Secondly, after clicking submit, the application could send users straight to the times page, defaulting to the alphabetically first category that they have times in. Within this page, there could be some kind of navigation bar, or dropdown, that would allow the user to select a new category to view. These buttons would be forms, so upon submitting, the application could send them to a different version of the page with a variable in the url that defines the category. E.g. /profile/<str:user_id>/<int:category_id>.

Although both solutions are essentially the same, as they utilise the same approach to use the url to display the data, I think option two would be better for the user, which should always be the end goal, yet might be tricky to implement.

After further thinking, I have concluded that this page is pretty much exactly the same as the data inputting page in functionality. It will contain times, and the user will be able to click on a different category to view that category of times.

Therefore, I can use exactly the same code for the routing without the bit about submitting times.

Flask Route, and Function Signature:

```
@app.route('update_times/<int:user_id>/<int:category_id>', methods=['POST'])
def update_times(user_id,category_id):
    #get the times from the url
    #update the database
    return render_template('profile.html',user_id=user_id,category_id=category_id)

@app.route('/profile/<int:user_id>/<int:category_id>', methods=['GET','POST'])
def profile(user_id, category_id):
    #If user selects a category change run this return statement:
    return redirect(url_for('profile', user_id=user_id, category_id=category_id))
    #If above condition is not true; render the template profile.html
    return render_template('profile.html',user_id=user_id,category_id=category_id)
```

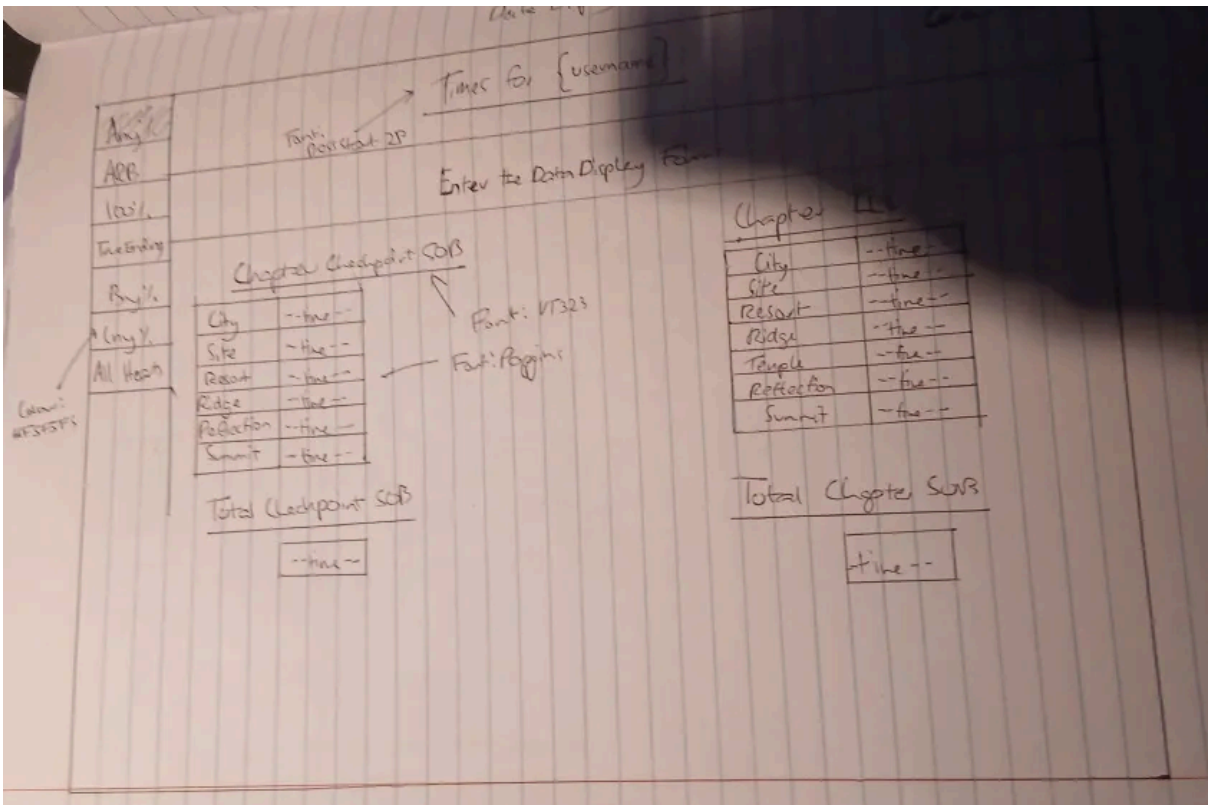
The url of this page will be the current user's id, followed by the html page called 'profile'. It will need to be able to post results of the form submission to the server, so that is why methods=['POST'] is included. When in the profile page, If a user selects a different category to view, the profile function will redirect them to the same function, but with a different category_id, allowing the program to change the displayed data.

Changes:
Here is the new code for the profile page.

```
@app.route('/update_times/<int:user_id>/<int:category_id>', methods=['POST'])
def update_times(user_id,category_id):

    if request.method == 'POST':
        checkpoint_times=request.form.getlist('checkpoints[]')
    #get the times from the url
    #update the database
    return render_template('profile.html',user_id=user_id,category_id=category_id)
 (variable) app: Flask
@app.route('/profile/<int:user_id>/<int:category_id>', methods=['GET','POST'])
def profile(user_id, category_id):
    checkpoints = []
    db.execute('SELECT time, run_number FROM Runs WHERE id = ? AND category_id = ?', (user_id, category_id))
    times = [row[0] for row in db.fetchall()]
    checkpoint_id = [row[1] for row in db.fetchall()]
    for checkpoint in checkpoint_id:
        db.execute('SELECT name FROM checkpoint WHERE id = ?', (checkpoint,))
        checkpoints.append(db.fetchall())
    db.execute('SELECT name, count FROM category WHERE id = ?', (category_id,))
    name, count = db.fetchall()
    return render_template('profile.html', user_id = user_id, category_id = category_id, times = times, checkpoints = checkpoints, count = count, name = name)
```

After the user submits their updated times, the update_times route will get the results of the form, and then update the database. It then renders the profile.html page. For now profile.html has the same code as get_times.html as it does the same stuff. I will eventually add changes but planning the basis for this is the same. This also means all the sql queries are the same.

Initial Sketch:



Login/Signup Page:

Since as a user, you don't want other people to be able to change your times, this application is going to need to contain a password system. If a user already has an account, the program will just check to see whether the pin is the same as the one in the database, but when the user doesn't already have an account, they will need to create one.

In this signup page, the user will simply enter a username, this must be unique, and a password, which doesn't have to be unique. The program will then hash this password using a hashing algorithm, and insert it into the database.

Flask route and function signature:

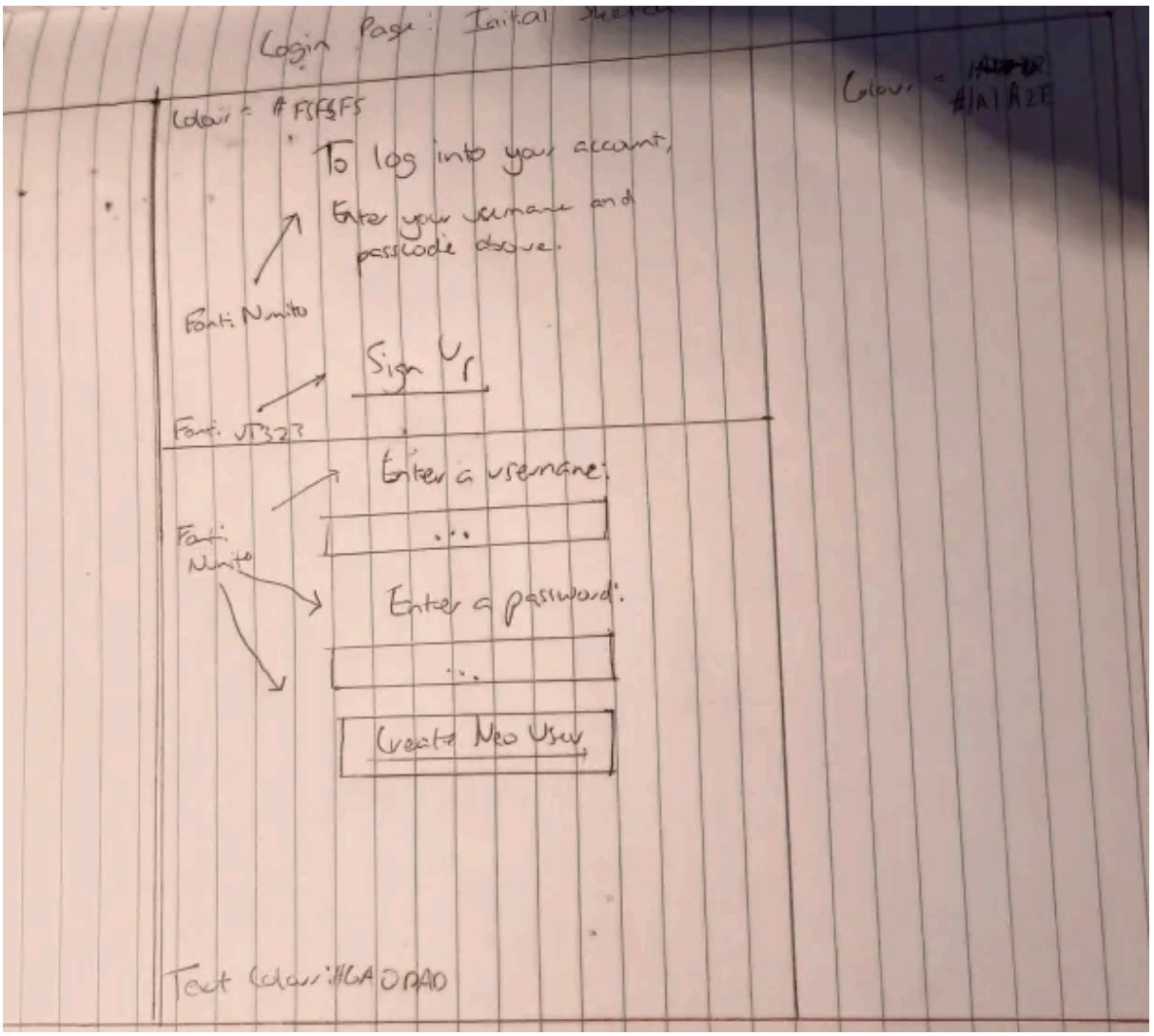
```python
@app.route('/signup')
def signup():
    return render_template('signup.html')

@app.route('/new_user', methods=['POST'])
def new_user():
    user_name='thing requested from the form'
    user_password='other thing requested from the form'
    #Hash the user password
    #Insert the new username and the hashed password into the database
    return render_template('home.html')
```

Upon the user entering the signup page, the page will simply render. This will contain the form that the user can input their details in. After clicking submit, the application will run the /new_user route. This will get the name the user submitted, get the password, and then insert the new data into the database. It will then run the home page again with a new user in the database now.

SQL queries for this page:
('INSERT INTO User (name, hash) VALUES (?, ?);', (username, (hashed_password)))
This inserts a new user into the database with the username they entered, and also assigns a hashed password that corresponds to the password they entered into the database as well.
The other sql query I am going to have to do for this page is to create every single different checkpoint time for all different categories for the new user. This is not going to be very fun, so I am not going to do it in the planning. That will require pretty much hard coding each of the values of the checkpoints within each category for a user and inserting them. Not very fun.

The result of these queries will be that the database will have changed.

Initial Sketch:

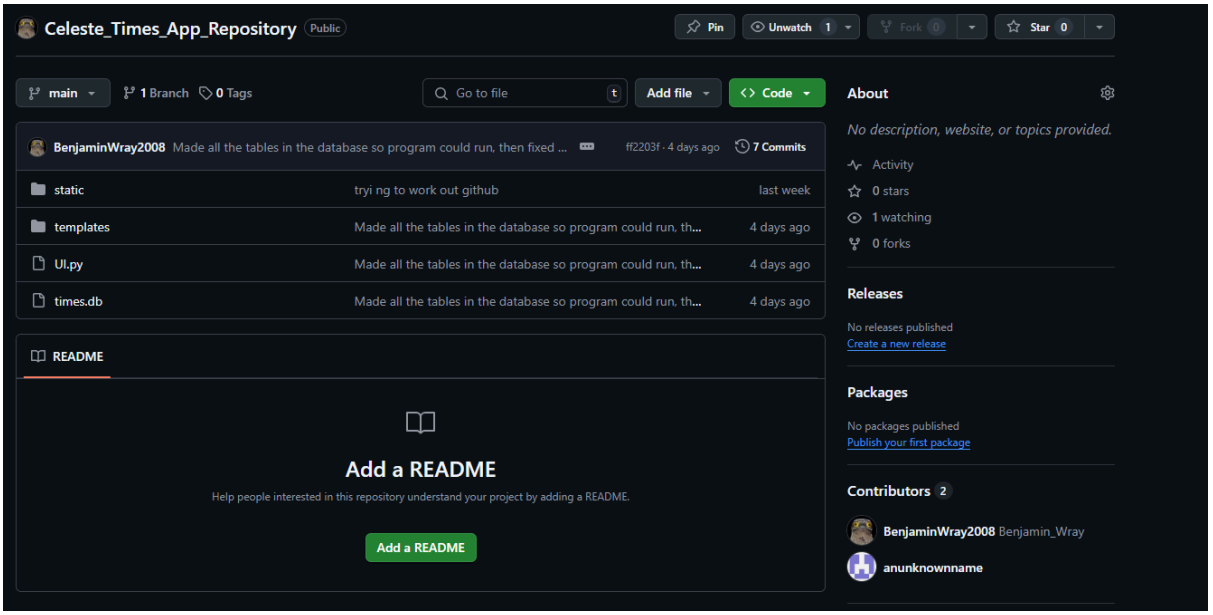


These are all of the pages that I am going to make for the application, although if I have spare time at the end of the project, I will come up with other things to add.

Github!

Here is my initiated github repository. This was difficult to make because apparently I have two different github accounts with the same name, so it kept bugging out when I tried to push the changes as it wasn't sure which account to use. This was annoying to fix as I didn't know what the issue was. Anyways it is fixed now. Here is the repo link and an image.

https://github.com/BenjaminWray2008/Celeste_Times_App_Repository



Development Phase:

This project will be completed in phases with testing and feedback stops between. The plan for the first section is to create all base aspects of the website with functionality. Styling not required.

Testing/Feedback Stop 1: Beginning 21st April - Ended 10th May

Accomlishments:

I have reached the point where my website does all of the things involving functionality that I outlined in the plan.
This includes:

- Adding users
- Logging into to a user account
- Entering data for that user (correct format is required)
- Displaying times for that user
- Updating database when required

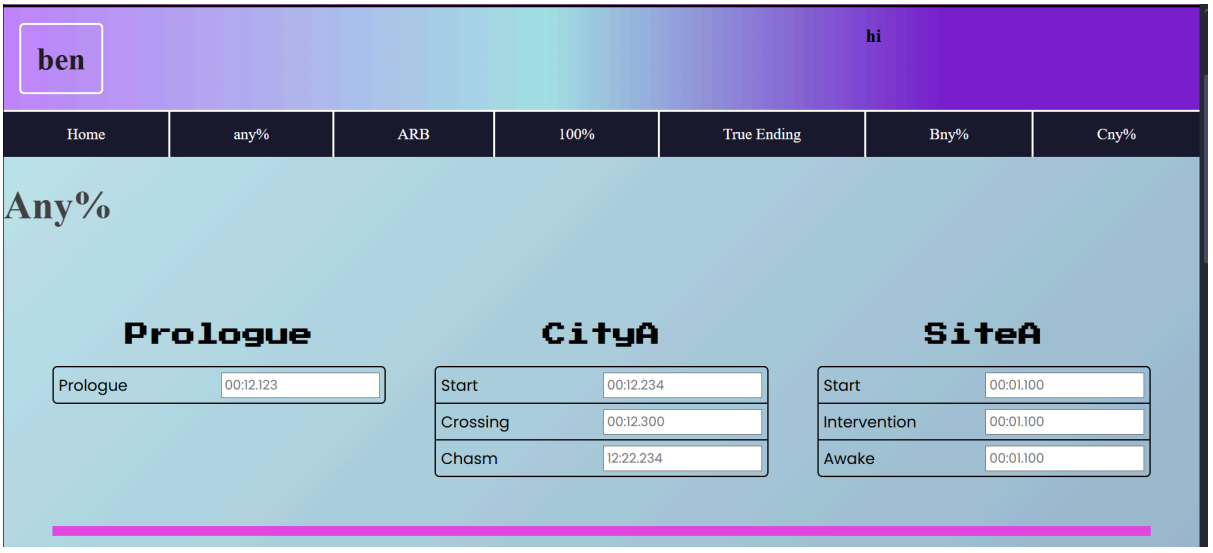
- Sum of Best leaderboard home page

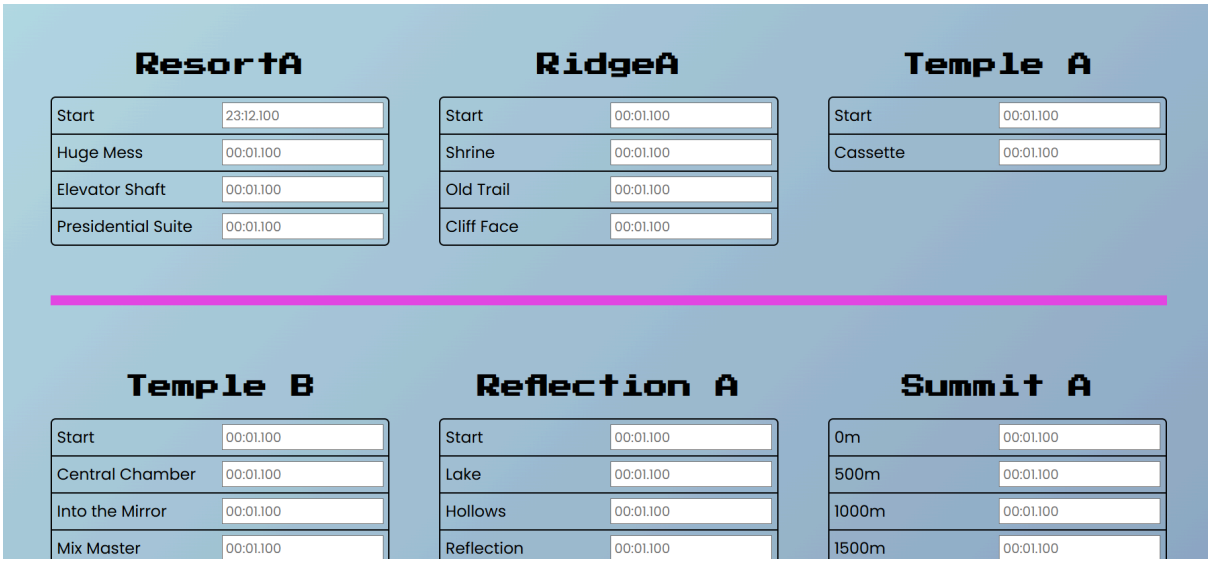
I haven't done much CSS so it doesn't look good, but here are screenshots displaying these components.

Logging into User account:

Editing Data for a User:

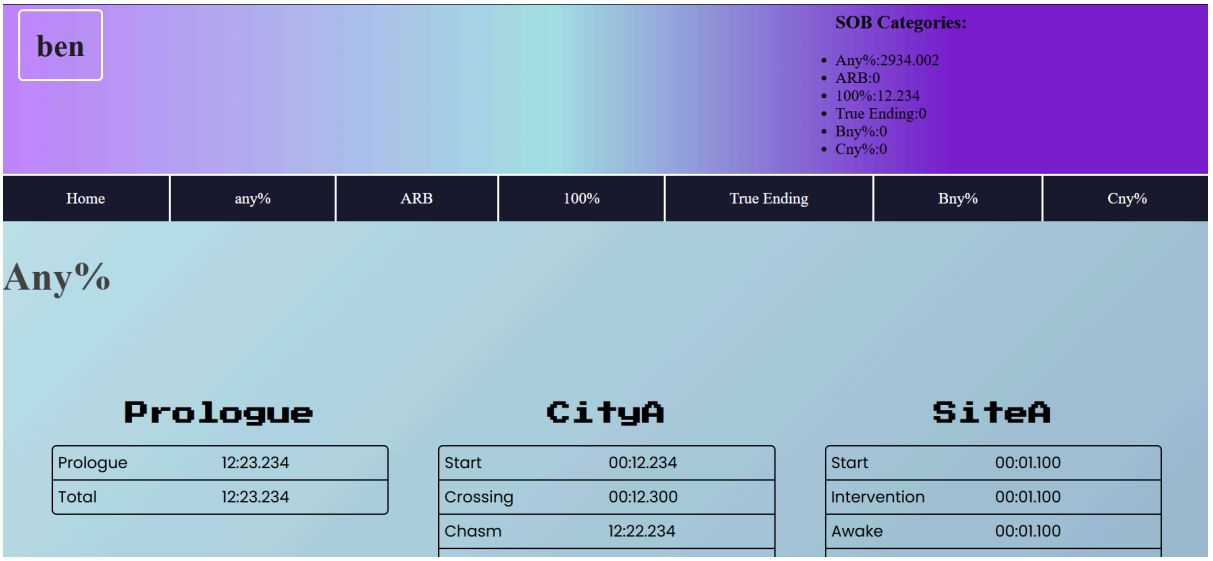
User can input times into the input boxes and app will update database with those times



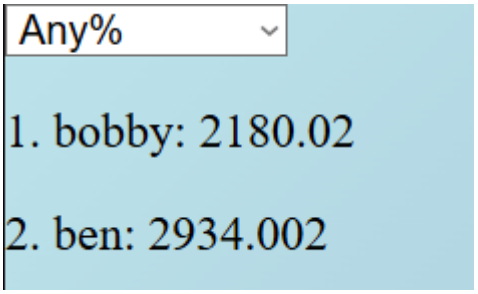


User Profile:

The profile page is exactly the same as the input times page except you can't input times; all the times are just p tags.



Home Page Leaderboard:



Those are the sum of best times that are valid, ranked.

Feedback for Sprint 1:

Ben Gorman: (Software student) Ben's feedback was that he didn't think it looked good for the profile to be the full width of the screen, as it looks like two headers, and also will have too much empty space. I also thought about this. He suggested making the profile be just an item on the screen that still had a reasonably large width. This would make it clear that it was just a profile, and now another header, and would also make the website have a more minimal and clean aesthetic. He also wanted me to make my code more readable as he found it very hard to work out anything when trying to debug some CSS.
Riki Feedback: (Software Student) Riki's feedback was to make the font system consistent throughout, and to make the colour scheme more simple with less focus on gradient and more on making colours fit together. He also suggested that instead of using 1 input box for searching users and logging in, you could instead have a search users box, and then a separate log in box. This would make the app more user friendly.
Mr. Rodkiss: (Non-human) The hover color on the navbar item should extend to reach the full size of the link. Use solid colours instead of gradients.
Matthew: (Software Student) Matthew's feedback was to make the home page feel cleaner by making the leaderboard seem more professional, and by styling all the hard colour line changes to make the pages feel smoother. He also wanted me to make the various aspects have a feature where if there are no results it says so. For instance, on creating a new user, it says nothing in the sum of the best profile header. This could instead say something like 'no run entries'.
Mr Dunford: (Non-human) Mr Dunford's feedback was to make it clear what each thing means for people entering the site who don't know the jargon. For example, he didn't know what any% means; although every actual speedrunner would, it's still important to make the site accessible to everyone. He also thought the gradients - especially the double gradient - were bad.

Response to feedback:

Ben's feedback:

I am going to apply Ben's feedback on the profile header in a later sprint. It looks pretty terrible and needs to be updated. I'm not sure how I'm going to do this. Ben suggested 1 big box but I may do otherwise. I will also make my code more readable with comments at some point.
Riki's Feedback: Riki's feedback about the colour schema is very valid, and I think that over the next few sprints I will work to fix it. He also thought it was confusing that the login box and search box were the same thing, and wanted me to change it. When I'm doing the login system properly I think I will add a separate login page, which will solve this issue.
Mr Rodkiss's: I have implemented already
Matthew feedback: Matthew wanted the leaderboard to be better, and obviously I will do this; I'm not leaving it as an unstyled list of times and ranks. He also wanted me to add a feature where getting no results will say no results. I will do this.
Mr Dunford feedback: I will remove the gradients :(everyone hates them. I will also make an about page at some point in the future for his reasons.

Goals for Sprint 2:

- Apply the Feedback Given

Home page stuff:

- Style the leaderboard
- Make different selections with the leaderboard, i.e category, type, user, etc.
- Redo the header - I don't like it.

Testing/Feedback Stop 2: Beginning 10th May - Ended 4th June

Accomplishments:

During this sprint I focused a lot on improving the homepage. I think I did a good job of this; it feels a lot more professional, but I didn't do a lot of the feedback about other pages due to this. I will do them next sprint. I did however add another search box (Riki's feedback), get rid of the bad gradients (Pretty much everyone's feedback), and massively improve the leaderboard (Matthew's feedback).

Although these changes took like a month, I feel as though I have improved a lot at CSS as I have had to redesign the home page a lot. I now find it a lot easier to work out which styles to use to do what I want.

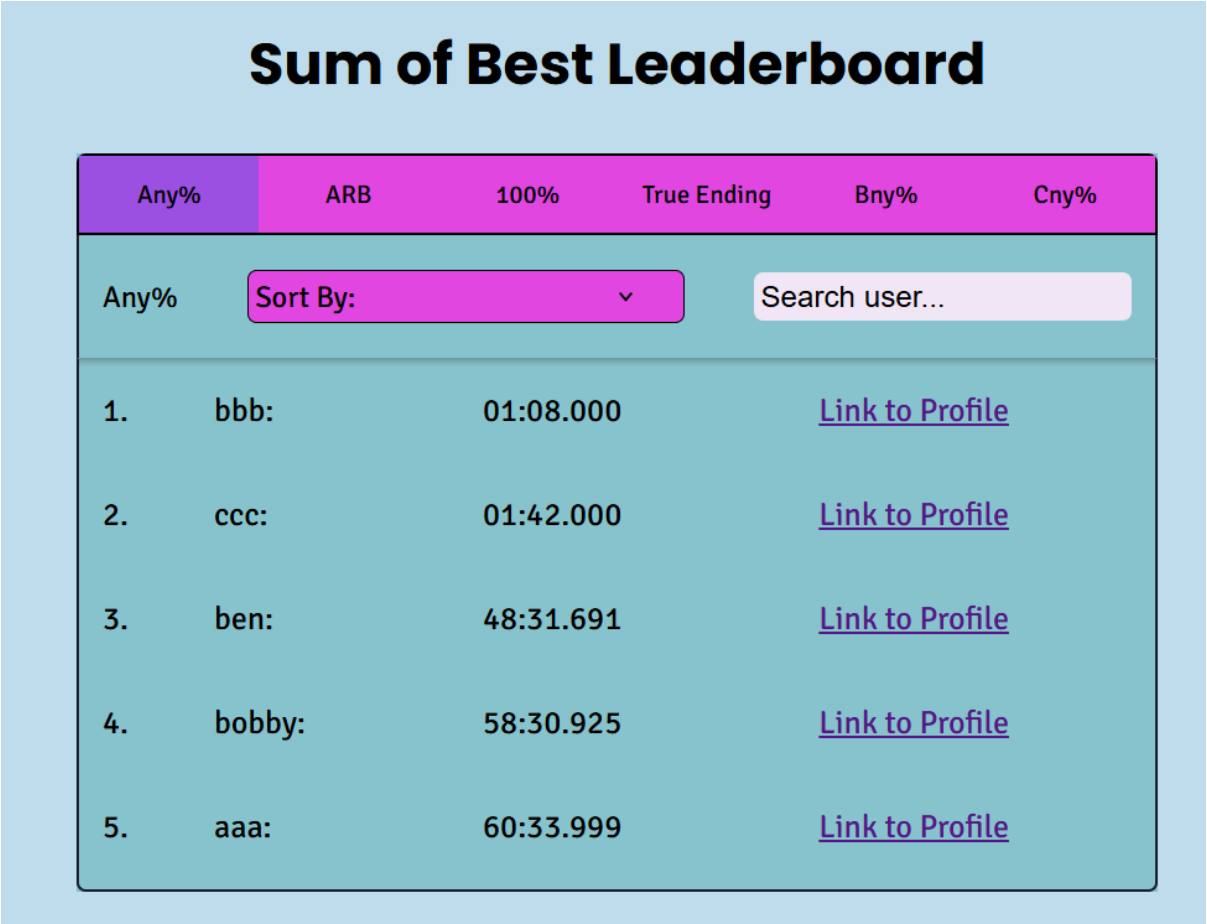
What I have done:

- Leaderboard:
    - Working, provides correct leaderboard data for each category
    - Category buttons with JSON, no page reload on user input. They work.
    - Sort by dropdown also with JSON, currently can sort by alphabetical or by time but it will be easy to add more options.
    - Search box. You can search for a user in the leaderboard, and it will snap down to that user's position.
    - Styling to make it look good.
- Header
    - The header name is on 1 line along with an image, and the search boxes. Feels cleaner
    - All the search box issues found from testing last sprint have been fixed.
    - Nav bar with various links
    - Colour Schema completely changed
- Stuff on Home template
    - Some basic site information
    - A dynamic display showing current amount of accounts and completed category SOBs.

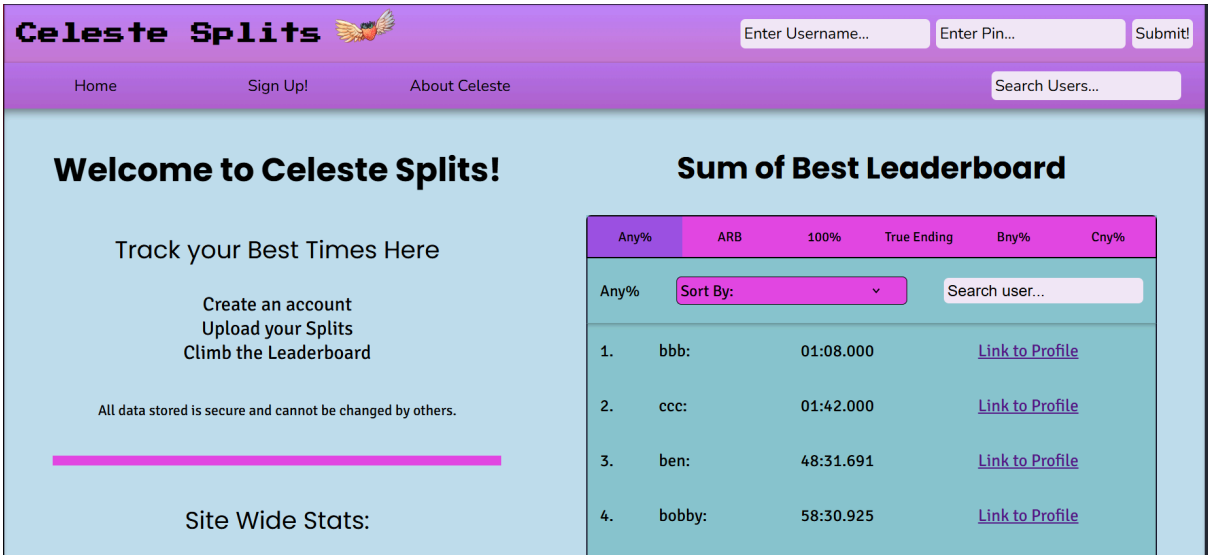
Here are some images showing the newly improved home page:

Header:

Leaderboard:



Full home page:



I think this is starting to look really good. There are still some changes I want to make, like adding more site stats, more select options for the leaderboard, and potentially adding a feature profile, but for now this is good.

Feedback for Sprint 2:

Hayden: (Regular Human) Hayden's feedback was for the leaderboard search. He said when you snap to a position on the leaderboard (after a search), highlight the corresponding item. I think this is a cool idea and I will apply it.
Oliver: (Regular Human) Get rid of the link to profile user row on the leaderboard, and just put the link on the username. Add a scroll on the leaderboard so it doesn't just infinitely go down.

Response to Feedback:

Hayden's feedback: I think it's a cool idea to highlight the position on the leaderboard you snap to so I will do this.
Oliver's feedback: The fact that every leaderboard position has a link to profile doesn't look professional, and I think putting the link just on the username is a good idea. I will do this. I will also make a scroll on the leaderboard as this is just a good idea.

Goals for Sprint 3:

In sprint 3 I want to make the get_times and profile page look better and be more functional. I will also apply the feedback given in sprint 2 and the feedback I haven't yet utilised from sprint 1.

This includes:

- Making links to different categories
- Making the profile summary header
- Adding slide down to specific checkpoints/chapters through dropdown and search bar
- Potentially redesigning the layout

This is a stop partially through sprint 3.
I've decided that I want the profile header to contain the following:

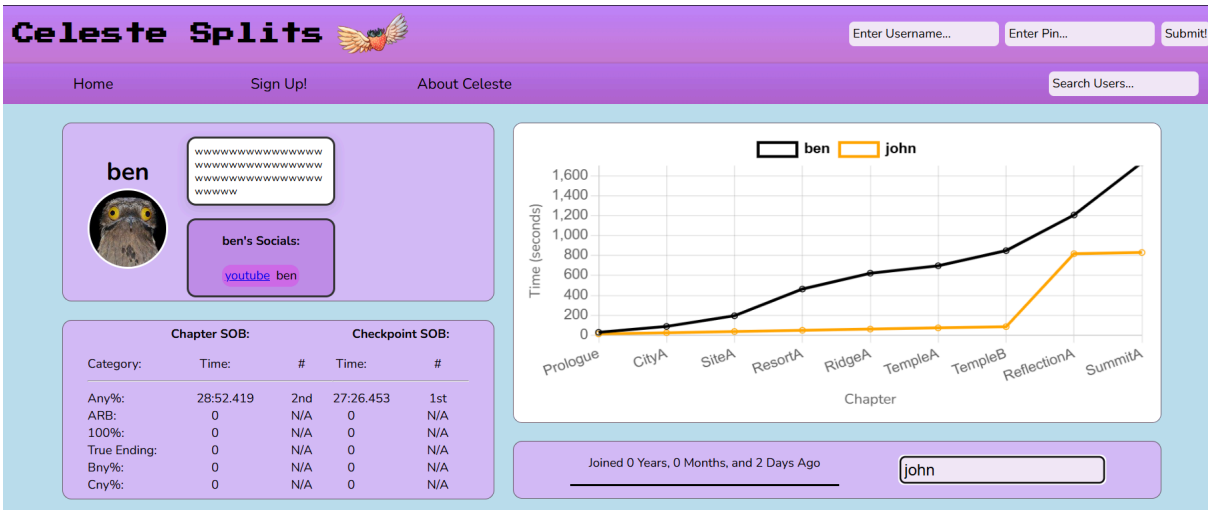
- Username

- User description
- Profile picture
- Socials links
- Sum of best stats
- How long ago they joined

Testing/Feedback Stop 3: Beginning 8th June, Ending 16th July

Accomplishments:

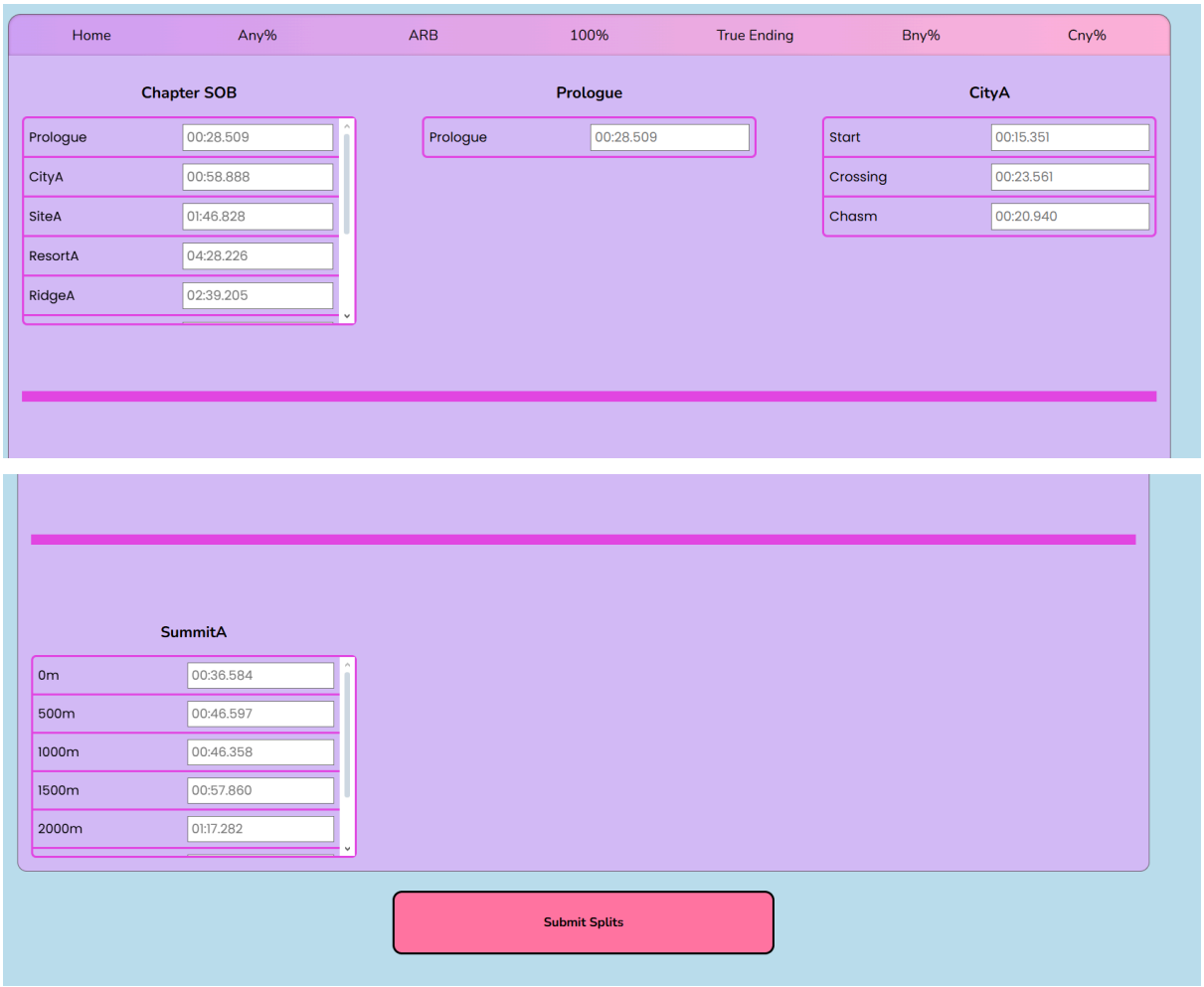
Sprint 3 is now finished. Everything is not quite how I would like it to be, but I want to move on. I have added all the features that I said for the profile header, which I spent the most time working on. The actual times layout I have kept mostly the same, as I think the new profile header compliments them well. Here are images of the product.



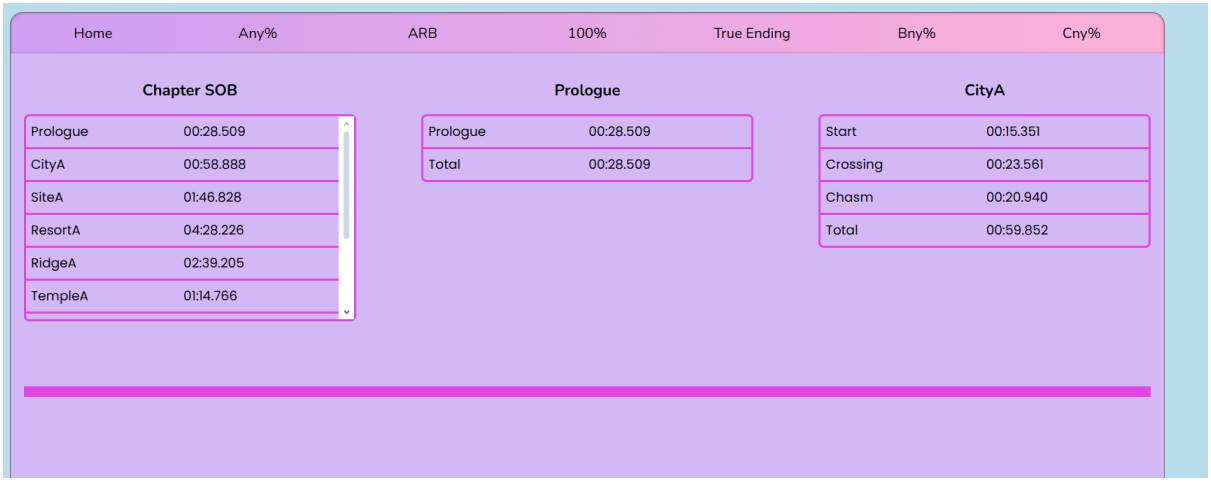
Above is an image of what the profile looks like from another perspective.

Above is the user's profile that they can edit. The user can change their profile picture, description, add, remove and edit social links, and can view their sum of best for checkpoints. The profile version (top image) has all of these same features, but obviously people can't change them, only view existing information.

Here is the times section.



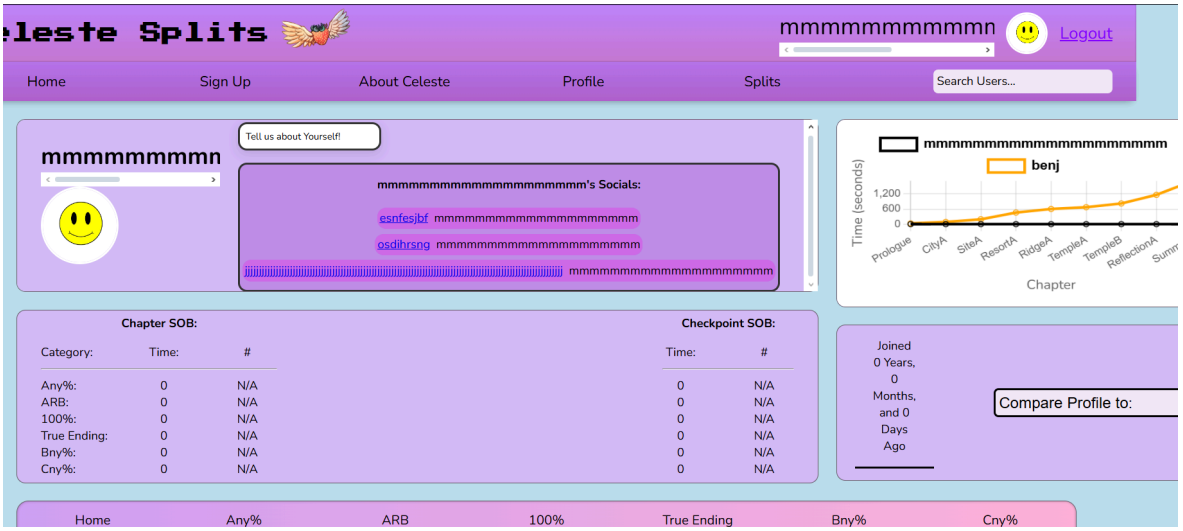
Above is in the get_times route where only the actual user with their password can edit times. There haven't been many changes here.



Above is the profile route where people can view another user's times.

Feedback for Sprint 3:

Logan Feedback: Times displayed as placeholder in inputs good. The pink line on the home page should decrease in width. Make times display in format of hh:mm:ss.msmsms, currently hours aren't included.
Ben feedback: Make the logout tag more readable; change colour. Add border to compare profile input box and change back colour. Look out of place. Placeholder text for that should be greyed to show its a placeholder. Footer - make it look better.
Oliver feedback: Change the social links; they look bad.
Jess feedback: Shouldn't add usernames with only spaces as characters, should be code that checks if not input (e.g. no socials), and if so should not just put in an empty box. The graph is very buggy, the profile picture is very buggy. Should not have to scroll all the way down the splits page when wanting to change something: feels bad when you only change 1 thing.
Matthew: Matthew's feedback was mostly aesthetic over adding more features. He found that some items on the home page (like the whole leaderboard or nav links) were not confined by their borders. This made it so there was some overlap when there was a border radius with the background colour sticking out. He also wanted me to make big words stay on 1 line. This is a major issue that I didn't think of.



As you can see. This is an issue.

Further testing and fixing of this issue has found that the description can just be behind the socials?
I have fixed most things, not fully happy with the results

Finn:
Make sign up page not crash
Make the checkpoint boxes for times less wide
Less pink on pink
Max whole profile header taller

Response to Feedback:

Logan Feedback:
I will make it do hours, it currently doesn't because it is kind of a pain to validate that another different section is correctly inputted.

Ben feedback: I'm not going to change the colour of the logout because I think that blue shows its a link even though it doesn't contrast with the purple. I will make changes to the compare to profile box to make it more evident its an input box that you can input stuff in.
Oliver feedback: I will change the social links; they do look bad.
Jess feedback: I can't really fix the graph and profile picture bugs; I really have no idea why they happen. I will make some extra restrictions on usernames so you can't add things like 3 spaces as a username. I will also try to add an extra way to submit splits without having to scroll all the way down.
Matthew feedback: Matthew pointed out a whole lot of minor errors that I fixed and a very big major error of big inputs will just go off the page. I have fixed this issue to the best of my ability by adding restrictions to lengths, so now the page actually stays on the page.
Finns feedback: Finn wanted the profile header to be taller because the box with username pfp and socials was going into scroll mode because it wasn't tall enough. This is something that will be good as it will allow me to fit more stuff in the profile box which I am struggling to layout well. He also wanted the time boxes to be less wide as there is a lot of empty space on either side of the numbers. I am conflicted on whether to do this as it would look more minimalistic, but maybe a bit cramped. I will do some testing and decide on the outcome. Finn also found a very annoying bug that is going to be very annoying to fix. It happens when you are logged in and then log in as someone else, everything crashes.

Testing/Feedback Stop 4: Beginning 16th July, Ending 18th September:

This sprint was all about fixing up small inconsistencies, adding quality of life features, fixing bugs, and adding things I didn't want to before.

Accomplishments:

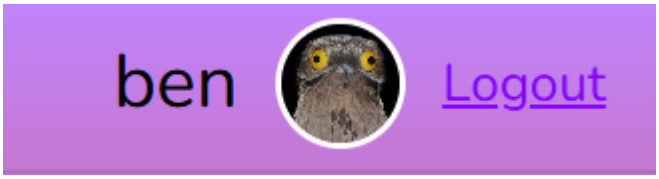
Along with major features below, I have fixed a load of bugs throughout the testing process, made a lot of minor changes to css, pep8ed my python etc.

Account system revamped:

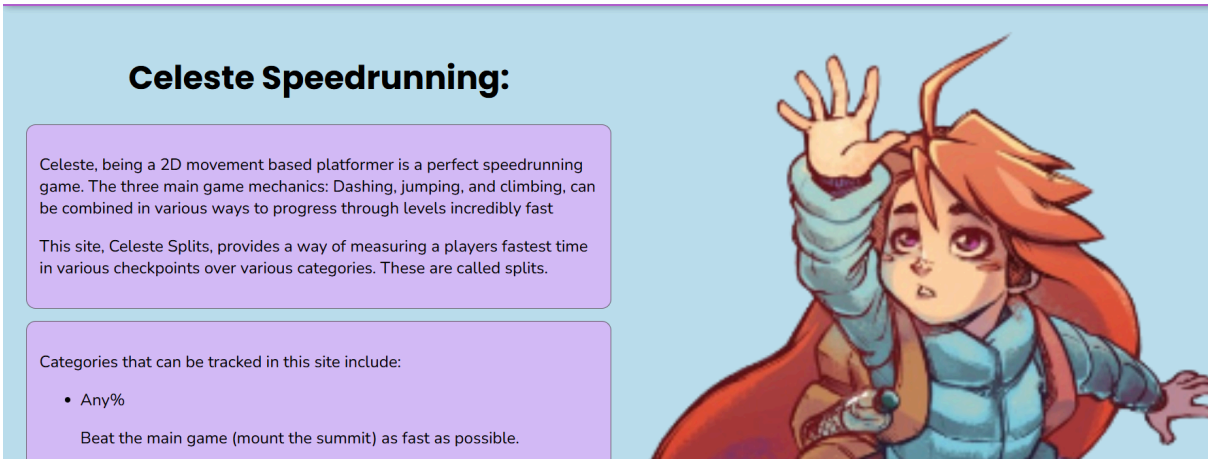
Sign in page:

Sign up page:

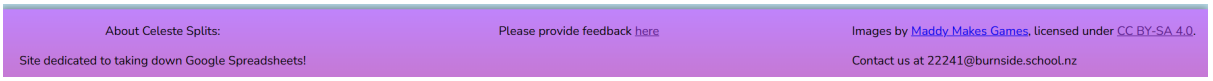
Header updated to show logged in user with profile picture.



About Page:



Footer:

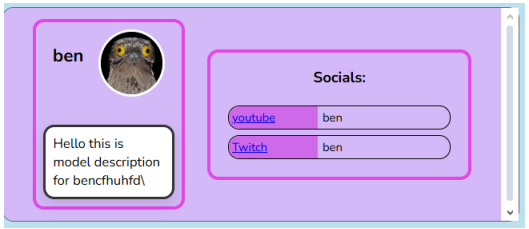


Feedback for Sprint 4 (Full site feedback):

Ben Feedback (Software Student):

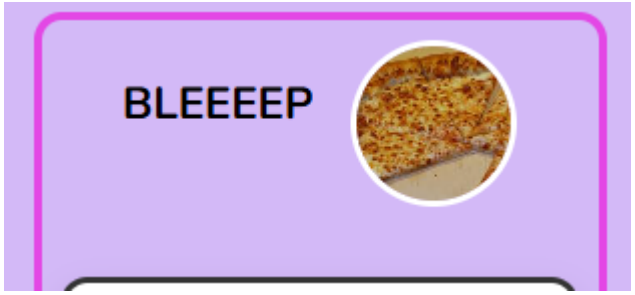
Doesn't support hours in inputs or sobs

Boxes different sizes looks weird



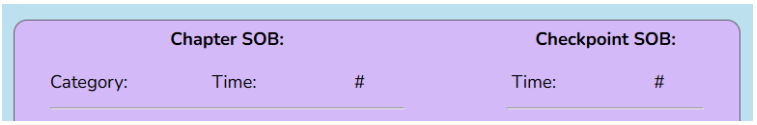
When signup it doesn't log you in, you have to log in separately

Name an pfp not centered vertically



Input fields are unstyled and boring

I don't like the line below time and time



Signup button and login button are too far apart so didn't realise you could login

Sign in button does weird wrapping

About page image blurry

Make leaderboard corners round

Add ability to compare all your times with someone else's

Logan Feedback (Regular Human):

Could make a nightlight mode where the background isn't blue

Placeholders in profile name section to fix the format shape

On hover profile nav text bold

Center the time and checkpoint name splits box / profile times box

Matthew feedback (Software Student):

There is no indicator for the maximum username length on signup. Indicating this would make it easier for the average user to understand why their input is being ignored. Signing up for the website then having to log in feels unintuitive without any text to indicate that the signup was successful and that you now need to login as opposed to something breaking.

Saving data is really slow. If you're impatient the site crashes due to recursive usage of cursors.

The run time graph looks really cool but allowing hovering over the graph to show each users time at a given time would be really cool. It would also help on runs where the time difference is substantial and there is a large difference between the lines of the two runs, causing the fast run to lose definition as it is compressed.

The website lacks an icon which makes it look out of place on the tab bar in the browser.

To a user unfamiliar with Celeste, some of the terminology will be unfamiliar which may discourage new players from using the tool as it increases the learning curve. Adding a dictionary or expanding the about celeste page to include more definitions could be useful.

Compare data to should be more clearly indicated as a text box, it looks like it should be a dropdown.

Riki Feedback:

Riki gave me a lot of positive feedback which is very nice.

He said he liked the graph and layout a lot, as well as the fact that it seemed unbreakable. He also said the colour scheme sort of doesn't work, but I really don't understand how colours work and he didn't say it was bad so it's fine.

Hayden Feedback:
In the sign in thing, Maybe say *Invalid account or password* or something.

When creating an account, a show password feature and confirm password feature would be nice in case I mistype something e.g. Pqssword. Because it's hidden I don't know what I've typed or if it's typed correctly.

Perhaps have some sort of filtering or report system on about me/description?

When entering a time it would be nice for it to say time saved or invalid input.

Perhaps saying invalid social as well but at least it has a 404

Response to Feedback:

Ben's feedback:

I'm not going to add hours into input acceptance, but I will add hour display, e.g. currently the site will say 61:11.111, but I will make it say 1:01:11.111.

I can’t make the box sizes the same as changing description and username size would cause it to break

I will add the feature that signing up auto logs you in

I will style my input fields

I do like that line so I’m leaving it

I think having the login in the top right is pretty standard so its fine

I fixed the weird wrapping

I will try to make leaderboard corners round, I don’t think it will work as there will be an overlap of the background colour of the nav

This is a great feature idea but I don’t have time to add it

Logan Feedback:

Night mode is a good idea, but I don’t have time to add it

This would structure the profile header better, but I don’t think it would really look good given a small number of inputs

I don’t like it when bolded

I tried earlier centering the text but it looks strange, I think it looks the best as it is.

Matthew’s Feedback:

I think it’s pretty obvious that you have hit the max length when you can’t type any more letters. I will fix the login on signup issue.

This is caused by my laptop being bad

Having fast runs look not that much faster on the graph is something I considered when making. I thought making the top of the graph logarithmic could be ideal as it emphasises how much harder it becomes to improve time, but I decided not to.

I don’t know how to do this

No new Celeste players are going to be on a speedrunning website

I think I will remove the border from the text box as that might be the issue.

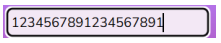
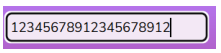
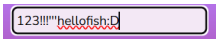
Testing:

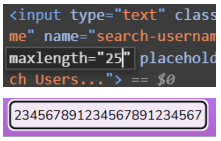
Put video of site tour here:

Header Testing:

Test Case	Expected Output	Real Output	What it's testing	Notes/pass/fail
Navbar home page link from home page	Goes to home page	Goes to home page	Does the link take to correct place	Pass 127.0.0.1:5000
Navbar home page link from different page	Goes to home page	Goes to home page	Does the link take to correct place	Pass 127.0.0.1:5000
Navbar sign up link from signup page	Goes to signup page	Goes to signup page	Does the link take to correct place	Pass 127.0.0.1:5000/signup
Navbar signup link from different page	Goes to signup page	Goes to signup page	Does the link take to correct place	Pass 127.0.0.1:5000/signup
Navbar signup link while logged in.	Goes to signup page	Goes to signup page	Does being logged in break anything	Pass
About page link nav about page	Goes to about page	Goes to about page	Does the link take to correct place	Pass 127.0.0.1:5000/about
About page link nav different page	Goes to about page	Goes to about page	Does the link take to correct place	Pass
Sign in link sign in page	Goes to signin page	Goes to signin page	Does the link take to correct place	Pass 127.0.0.1:5000/signin
Sign in link	Goes to signin	Goes to signin	Does the link	Pass

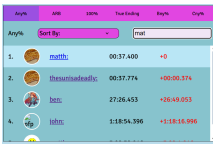
different page	page	page	take to correct place	127.0.0.1:5000/signin
Sign in link logged in	Shouldn't exist	Doesn't exist	Does logging in remove the ability to signin again	Pass
Logout link logged in	Logs out sends to home page	Logs out sends to home page	Can user sign out	Pass 127.0.0.1:5000/logout (sends to logout page as that route will reset session)
Profile link nav logged in profile page	Goes to profile page (any% category) as logged in user	Goes to profile page any% category as logged in user	Does the link take to correct place	Pass 127.0.0.1:5000/profile/27/1
Profile link nav logged in different page	Goes to profile page (any% category) as logged in user	Goes to profile page any% category as logged in user	Does the link take to correct place	Pass
Splits link nav logged in splits page	Goes to splits page (any% category) as logged in user	Goes to splits page any% category as logged in user	Does the link take to correct place	Pass 127.0.0.1:5000/get_times/27/1
Splits link nav logged in different page	Goes to splits page (any% category) as logged in user	Goes to splits page any% category as logged in user	Does the link take to correct place	Pass
Search bar real user search E.g. ben	Goes to that users profile	Goes to that users profile	Does searching for user work	Pass
Search bar invalid user search E.g. ojgjhsg	Reloads home page	Reloads home page	Does app recognise invalid users	Pass
Search bar logged in current logged	Goes to that users profile	Goes to that users profile	No edge case of searching for current	Pass

in user search E.g. ben, logged in as ben			logged in user	
Search bar no input	Reloads home page	Reloads home page	Doesn't try to search for any user with any characters or something like that	Pass
Search bar 19 chars E.g. 1234567890123456789	Reloads home page (unless there is an actual user with that name)	Reloads home page (there are no users with 19 character usernames)	Max length working correctly	Pass 
Search bar 20 chars (max) E.g. 12345678901234567890	Reloads home page (unless there is an actual user with that name)	Reloads home page (there are no users with 20 character usernames)	Max length working correctly	Pass 
Search bar 21 chars (beyond max) E.g. 123456789012345678901	Shouldn't be possible to enter	Isn't possible to enter	Max length working correctly	Pass
Search bar special characters E.g. 123!!!""hello fish:D	Should just count as an invalid username	Does just count as an invalid username	Check that special characters don't break anything	Pass 
Search bar sql injections e.g. ' OR 1=1 -- ' for SELECT hash from User WHERE name = '{user	Counts as invalid username	Did count as invalid username (had to user inspect tool to change maxlength to	SQL injections don't work. (They shouldn't because no f strings were used in	Pass

input}’;		even fit it)	queries)	
Search bar inspect tool change max to bigger than 20 e.g. 25 and input 25 long string E.g. 1234567890123456789012345	Shouldn’t even process it	Does process it	Changing maxlength to be able to break something is bad	 Fail - need to add back end validation for this.
Search bar logged in at users profile search same users profile E.g. on ben’s profile search ben	Reloads page	Reloads page	Nothing breaking by not having to change url	Pass

Home Page Testing:

Add new user to site	Runner account + 1	Runner account + 1	Check that it updates	Pass
Complete new category entry	Category entry +1	Category entry + 1	Check that it updates	Pass
Default category leaderboard load	Displays runs from any% sorted by time	Does it	Check default load works	Pass
Default category sort by alphabet load	Still any% sorted alphabetically	Works	Check changing sort works	Pass
Change category leaderboard	Displays runs for that category	Works	Check different categories work	Pass

Change sort by different category	Displays runs for that category sorted by whatever	Works	Check sort by works not on default category	Pass
Search user real no completed run	Does nothing	Does nothing	Check doesn't select any runner	Pass
Search user real has completed run in category E.g. matthew	Scrolls down to their position and highlights	Works	Check you can search user	Pass 
Search user not real	Does nothing	Does nothing	Checks invalid user doesn't break anything	Pass
Search partial user e.g. mat for matth	Scrolls down to position and highlights	Works	Checks you can search partial user	Pass 
Search invalid user special characters e.g. 111””!!!fish	Does nothing	Does nothing	Nothing breaks when special characters	Pass
Search sql injections e.g. ' OR 1=1 -- ' for SELECT hash from User WHERE name = '{user input}'	Does nothing	Does nothing	No hacking my site	Pass
No input search	Does nothing	Does nothing	Shouldn't do anything	Pass

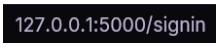
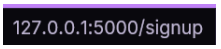
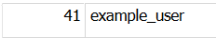
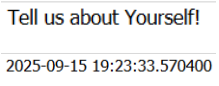
Sign Up / Login Testing:

Signin valid username valid password E.g. ben wray	Logs in	Logs in	Check login works	Pass
Signin valid username invalid password E.g. ben WRONGPASSWORD	Doesn't log in	Doesn't log in	Checks password integrity	Pass
Signin valid username no password E.g. ben	Doesn't let submit	Doesn't let submit	Checks for no empty input	Pass
Signin invalid username valid password E.g. FAKEUSER wray	Doesn't let in	Doesn't let in	Checks username and password must match	Pass
Signin no username valid password E.g. '' wray	Doesn't let submit	Doesn't let submit	Check for no empty input	Pass
Signin invalid username invalid password	Doesn't let in	Doesn't let in	Checks for no fishy business	Pass

FAKEUSER FAKEPASSWORD				
Signin no username no password	Say required field	Works	Checks for no input	Pass
Signin take away required tag from inputs	Home page	Works	Checks for no hacking	Pass
Signin change max and min input size	Shouldn't matter, there will be no users with bigger than max size name anyway	Works	No reason	Pass
Signin invalid user and password 14 characters E.g. 12345678901234	Works as usual	Works	Check boundary	Pass
Signin invalid user and password 15 characters E.g. 123456789012345	Changes nothing	Works	Check boundary	Pass
Signin invalid user and password 16 characters E.g. 1234567890123456	Won't be able to enter 16	Works	Check boundary	Pass
Logged in	Go to home	Fails - lets me	Make sure	Fail - need to

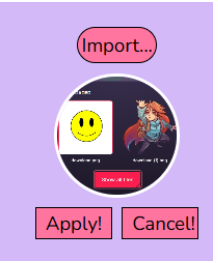
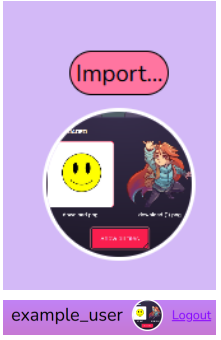
enter url /signin	page - don't let them	go to that page and crashes when log in	they can't log in while already logged in	check condition when running login route whether they are logged in. Fixed
Logged in enter url /signup	Go to home page - don't let them	Fails - lets me go to that page sign up as new person and crashes when log in	Make sure they can't sign up / login while already logged in	Fail - same thing as above Fixed
Signup valid username valid password E.g. newperson newpersonpassword	Signup new user	Does it	Make sure new uses can be creates	36 qwerty
Signup valid username taken password E.g. Newperson wray	Should work	Does it	Nothing weird with non-unique passwords	Pass
Signup valid username no password E.g. Newuser	Password field required	Does it	No people with no password	Pass
Signup taken username valid password E.g. ben newpersonpassword	Reloads signup page	Does it	No non-unique username	Pass

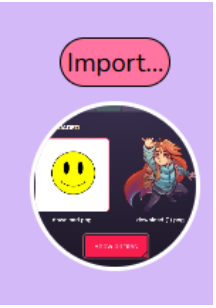
Signup taken username no password E.g. benj	Required password field	Does it	No people with no password	Pass
Signup no username valid password E.g. ‘’ newpersonpassword	Username field required	Does it	No people with no username	Pass
Signup no username no password	Username and Password required	Does it	No people with no data	Pass
Signup username and password 14 chars E.g. 12345678901234 12345678901234	Adds them	Does it	Boundary Case	Pass
Signup username and password 15 chars E.g. 123456789012345 123456789012345	Adds them	Does it	Boundary Case	Pass
Signups username and password 16 chars E.g. 1234567890123456 1234567890123	Doesn’t work	Does it	Boundary Case	Pass

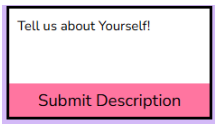
456				
Signup delete max length > 15 chars both	Doesn't work 404	Does it	Can't be hacked	Pass
Signup delete required field both	Doesn't work	Does it	Can't have users with no data	Pass
Signup special characters e.g. !!!111'"hellofish	Works	Does it	Doesn't break anything	Pass
Search sql injections e.g. ' OR 1=1 -- ' for SELECT hash from User WHERE name = '{user input}'	Adds them as a user if less than 15 chars otherwise do nothing	Does it	Can't hack my database	Pass
Signup delete minlength < 4 both and enter	404	Does it	Can't break anything	Pass
Click signin link on signup page	Goes to sign in page	Does it	Check that link works	Pass 
Click signup page on sign in page	Goes to signup page	Does it	Check that link works	Pass 
On create new user are all fields on user field filled	Should have their username with password hashed with default profile picture, default description, and current time as date joined.	Does it	Check that new user creation works	Pass  Hash: 50d858e0985ecc7f60418aaf0cc5ab587f42c2570a884095a9e8ccacd0f6545c (no ss because not readable) 

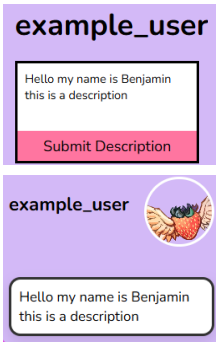
				download.jpg
On create new user check all categories are created with time entries	Should have all 6 categories in Run table with that users id with	Does it	Check that new user creation works	Pass - no ss because each new user has like 400 lines of run entries.

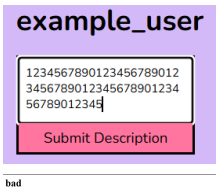
Edit Profile Testing:

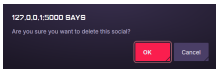
Profile picture default display	Should display picture cheese.jpg	Does it	Check default pfp works	Pass 
Profile picture click import button	Should open file explorer and display images that user can upload	Does it	Check you can see uploading files	Pass 
Profile picture open file	Should close file explorer, replace the current pfp with new one and display two buttons saying cancel and apply	Does it	Check you can upload and view a pfp	Pass 
Click apply button pfp	Should change pfp in header and on profile/splits page and change database with new pfp. Apply and cancel buttons should disappear and import should	Does it	Check you can upload new pfp	Pass 

	come back			
Click cancel button pfp	Should go back to original pfp, apply and cancel buttons should disappear, and import button should come back	Does it	Check you can cancel a pfp upload	Pass  After clicking cancel: 
Click import pfp button while already selecting a pfp	Should just go back to file explorer repeat process	Does it	Check you can go back to selecting file while already file selected	Pass 
Go to inspect tool and take away image required tag	On click apply of not image back end should not upload the new file to db. Should reset back to original pfp	App lets you upload any file as a pfp, although it doesn't display, it is in the database and the project files.	Make sure you can't upload files to my site.	Fail  Lets you upload anything. Below is an image of me uploading the celeste application to my site. Initially, I thought this wasn't a big deal, but then it crashed github, made me miss commits for a week because

				I couldn't push the changes, and really just broke everything.  Here is the fix: I added if not file.content_type.startswith('image/'): Which validates that the contents of the upload are that of an image.
Description default on user creation	Should be 'Tell us about yourself!'	Does it	Testing that new user has a description	Pass 
Submit same description	Should do nothing	Does it	Testing that uploading same description doesn't break anything	Pass
Submit description empty	Should just give them no description - in profile should not display the box that description goes in	Gives them no description, but the box still appears on profile	Checking if no input doesn't break anything	Fail - will only display description box on profile if description != '' - Fixed
Submit description < 50 E.g. Hello my name is	Updates db with the entered description	Doesn't work - back end is saying that a 49 length description is length 53.	Checking if updating description works	Fail - it's because apparently new lines in text areas count as 2

Benjamin this is a description				characters. I removed these extra characters per line and now it works 
Submit description == 50 E.g. 12345678901234567890123456789012345678901234567890	Updates db with entered description	Works	Checking boundary	Pass
Submit description == 51 E.g. 123456789012345678901234567890123456789012345678901	Doesn't let you type it in	Works	Checking boundary	Pass
Submit description special characters E.g. hello123!!!"" ☺	Updates db with entered description	Works	Checking weird examples	Pass 
Submit description sql injection	Updates db with the entered description	Works	Checking for no hacking	Pass

E.g. ' OR 1=1 -- ' for SELECT hash from User WHERE name = '{user input}'				
Submit description change maxlength attribute enter 55 char description E.g. 12345678901234567890123456789012345678901234567890123456789012345	404	Works	Checking no getting past boundaries	Pass 
Add social name with social link. E.g. https://example.com Example.com	Adds the social and opens another area to add another	Works	Check you can add socials	Pass 
Add social name no link E.g. example.com (name)	Doesn't let submit	Works	Check you can't add empty inputs	Pass 
Add social link no name E.g. https://example.com no name	Doesn't let submit	Works	Check you can't add empty inputs	Pass 
Submit with no link or name	Doesn't let submit	Works	Check you can't add empty inputs	Pass

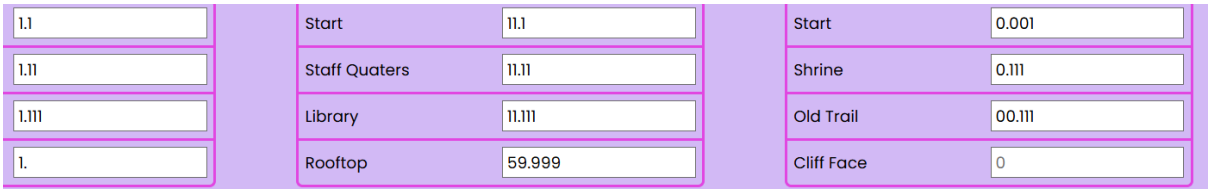
Delete required tag from link and name and submit	Doesn't update	Uploads an empty social tag	Check you can't add empty inputs	Fail Changed it to check if there is a social in backend Fixed
Edit existing social name keep link constant E.g. example.com → examplechange.com	Updates the existing name, link stays the same	Works	Check updating existing social works	Pass
Edit existing social link keep name constant E.g. https://example.com → https://examplechange.com	Updates existing link name stays the same	Works	Check updating existing social works	Pass - sorry about tiny image
Edit existing link and name E.g. example.com → examplechange2.com https://examplechange.com → https://examplechange2.com	Updates both	Works	Check updating existing social works	Pass - no image too tiny
Delete existing social	Deletes it - asks are you sure first	Works	Check you can delete socials	Pass 

Add 4 socials (max)	Adds them	Works	Check you can add up to max socials	Pass
Try to add 5th social	Doesn't work - user shouldn't even be able to add one as the add shouldn't show up	The user can't add a new one, but the inputs for them to do so are still there.	Check you can't add more then max socials	Fail - fixed by only adding the add new social form if length of socials list is < 4.

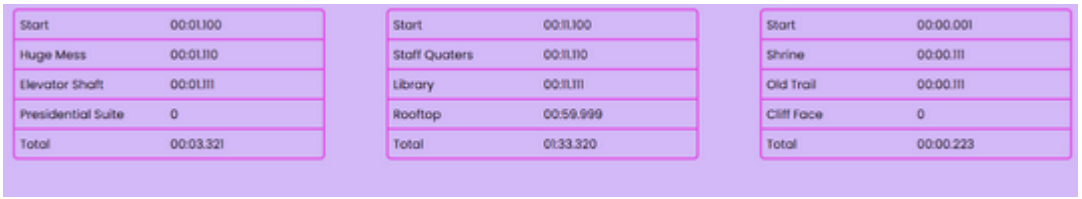
Submitting Times testing:

Submit time valid	Same time shows up in same category and checkpoint	Works	Time submission works	Pass

In the image below are entered times that should all be considered valid (except the 1. for the presidential suite, I forgot to delete it). These are all times that only contain second and millisecond values.



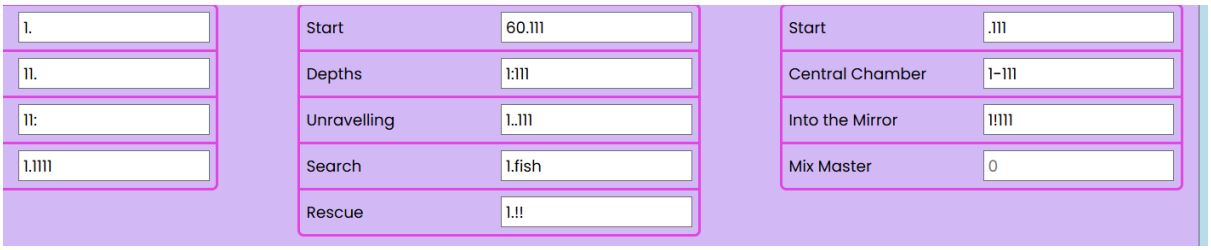
After submit, this is the output for those times. Any time that ends as 0 means that it has been rejected as invalid, or that the time is just equal to 0.



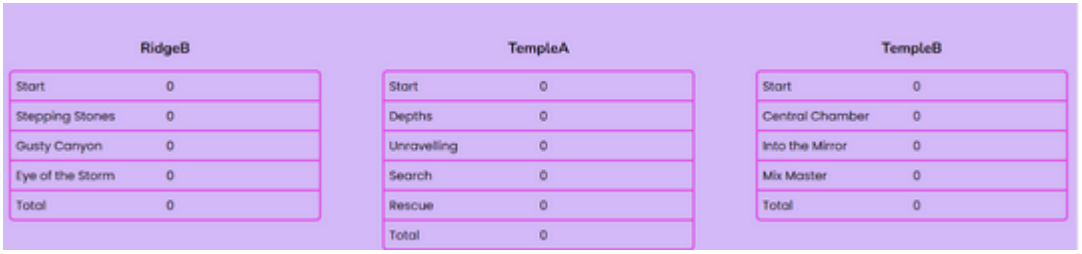
As you can see, everything passed (except the presidential suite as that was an invalid time in the first place). Furthermore, the programme formatted everything correctly into mm:ss.msmsms format. If there is a missing digit (e.g. 1.111 misses a 0

before (01.111)) the program should fill it in. As you can see 1.111 turns into 00:01.111 in the output on profile. This is correct

Image below is all the second and millisecond times that should be considered invalid.



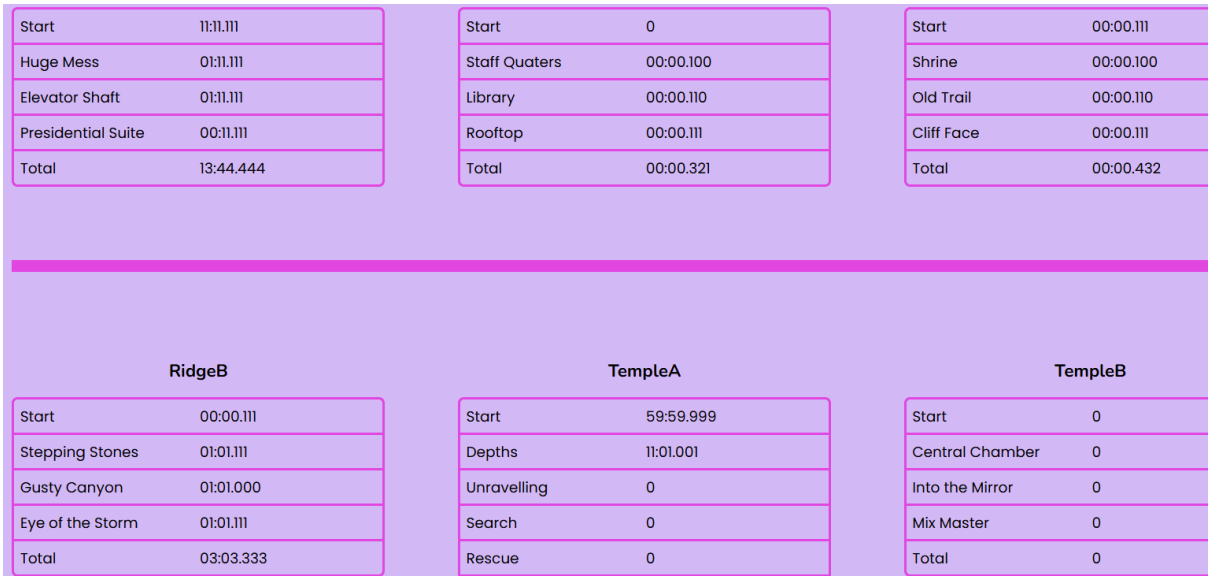
On submit, everything here should fail (show up as 0 in output). Below is the outputted results.



As you can see, everything failed, which is good.

Next is testing for times that include minutes. Below are all the ways of entering times that should be considered valid with minute, second, and millisecond values.

Below is the output after submitting. All times submitted, showing that the app can validate that times are valid. Same as before, it has formatted times to fit the mm:ss.msmsms format. This has all worked well.



Start	11:11.111
Huge Mess	01:11.111
Elevator Shaft	01:11.111
Presidential Suite	00:11.111
Total	13:44.444

Start	0
Staff Quaters	00:00.100
Library	00:00.110
Rooftop	00:00.111
Total	00:00.321

Start	00:00.111
Shrine	00:00.100
Old Trail	00:00.110
Cliff Face	00:00.111
Total	00:00.432

RidgeB

Start	00:00.111
Stepping Stones	01:01.111
Gusty Canyon	01:01.000
Eye of the Storm	01:01.111
Total	03:03.333

TempleA

Start	59:59.999
Depths	11:01.001
Unravelling	0
Search	0
Rescue	0

TempleB

Start	0
Central Chamber	0
Into the Mirror	0
Mix Master	0
Total	0

Image below is all the times including minute second and millisecond values that should be considered invalid.

Start	11:11:111
Scrap	11.11.111
Contraption	:11.111

Start	:11.
Intervention	11-11-111
Awake	11:11.11w

Start	11f11i11
Combination Lock	!1:11.111
Altar	11:11.11.

ResortA

Start	HELLO
Huge Mess	60:00.001
Elevator Shaft	11:111.11
Presidential Suite	111:11.11

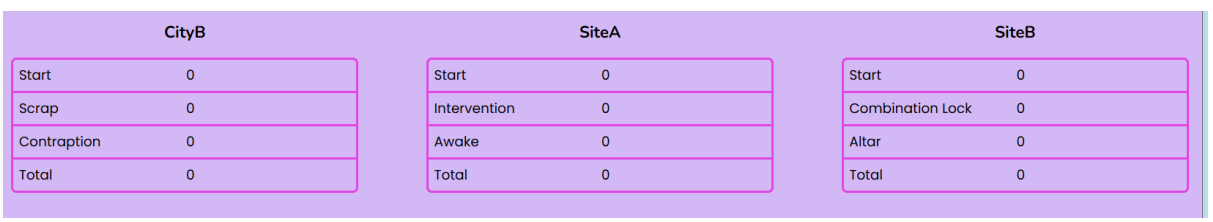
ResortB

Start	5m 2s 400ms
Staff Quaters	05 20 100
Library	11/11/111
Rooftop	-10:11.111

RidgeA

Start	00:00.111
Shrine	00:00.100
Old Trail	00:00.110
Cliff Face	00:00.111

Output after submission: They are all 0 so everything was counted as invalid.



CityB

Start	0
Scrap	0
Contraption	0
Total	0

SiteA

Start	0
Intervention	0
Awake	0
Total	0

SiteB

Start	0
Combination Lock	0
Altar	0
Total	0

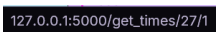
Start	0
Scrap	0
Contraption	0
Total	0

Start	0
Intervention	0
Awake	0
Total	0

Extra cases:

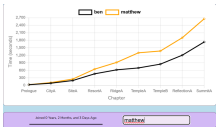
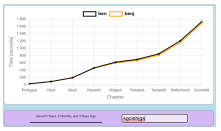
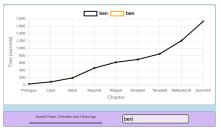
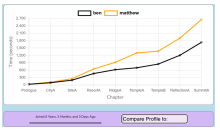
' OR 1=1 -- ' for SELECT hash from User WHERE name = '{user input}' - Counted as invalid - did nothing - SQL Injection

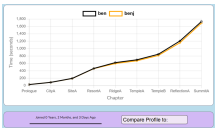
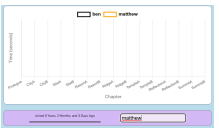
Overall everything worked well.

Category navbar splits home button	Take to home page	Takes to home page	Checks the home page link works	Pass 
Category nav bar splits current category E.g. on any% page click any% link	Go to same page as current user with times available	Works	Check clicking link on same page works	Pass 
Category nav bar splits different category E.g. on any% click bny%	Go to the clicked page as current user with times available	Works	Check clicking link for different page works	Pass 
Category nav bar profile home page	Go to the home page	Works	Check clicking home page link on profile page works	Pass 
Category nav bar profile current category E.g. on any%	Go to the same profile category page	Works	Check clicking on same category link works	Pass 

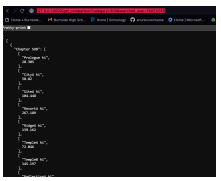
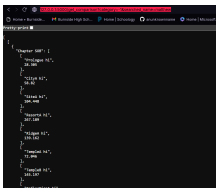
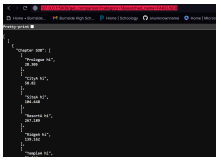
click any%				
Category nav bar profile different category E.g. on any% click bny%	Go to the clicked page profile category	Works	Check clicking on different category link works	Pass 127.0.0.1:5000/get_times/27/5

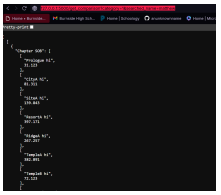
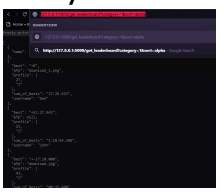
Profile Display Testing:

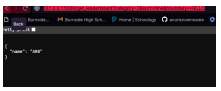
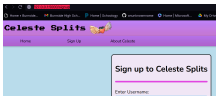
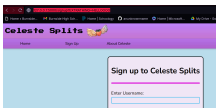
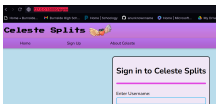
Click social link, e.g. Youtube link for ben	Goes to the url of the link	Works	Check social links work	Pass https://youtube.com/ben
Compare user real user search E.g. matthew	Compares to that users profile	Works	Does comparing to user work	Pass 
Compare user invalid user search E.g. sgoishigsi	Compares to person with fastest time (benj)	Works	Does app recognise invalid users	Pass 
Compare user same user search E.g. on ben profile compare to ben	Just compare to the same person - will look like 1 line	Works	No edge case of searching for current logged in user	Pass 
No input	Doesn't do anything	Works	Doesn't try to search for any user with any characters or something like	Pass 

			that	
Compare user with special characters e.g. 123!!!”’hello fish:D	If there is a user with that username then compare to them otherwise compare to person with fastest time (benj)	Works	Check that special characters don’t break anything	Pass
Search bar sql injections e.g. ' OR 1=1 -- ' for SELECT hash from User WHERE name = ‘{user input}’;	If there is a user with that username then compare to them otherwise compare to person with fastest time (benj)	Works	SQL injections don’t work. (They shouldn’t because no f strings were used in queries)	Pass
Graph initial load	Should run person with fastest time in that category (benj) compared to current user profile	Works	Check default load works	Pass 
Change category on graph E.g. any% to bny% for comparison between ben and matthew	Should change category and compare entered users	Works	Check changing category works	Pass 

Adding things to URL Testing:

Type random stuff into url: E.g. 127.0.0.1:5000/AJKEFHJHKEAGFH	Runs 404	Works	Check no crash due to url that doesn't match any routes	Pass
/get_comparison route Run url with non real inputs e.g. /get_comparison?category=500&searched_user=NOTREALUSER	Should default to user id 26 and any% category	Defaults to user id 26 but doesn't default to any% category. Goes with category id 500 and just returns null as it doesn't exist	Check Get_comparison route is unbreakable	Fail Fixed by checking if category id is in database 
/get_comparison route Run url with more non real inputs but boundaries E.g. /get_comparison?category=-1&searched_user=matthew /get_comparison?category=1&searched_user=NOTREALUSER	Any non real parameters in url should make it run default.	Works	Check that boundaries don't crash server	Pass  
/get_comparison route	Should run and give expected	Works	Check it does actually work	Pass

Run url with real inputs E.g. /get_comparison?category=1&searched_user=matthew	values			
Home route run E.g. 127.0.0.1:5000 Or 127.0.0.1:5000/ Or 127.0.0.1:5000// etc	Runs the home page	Works	Check home page runs	Pass
/get_leaderboard route real parameters E.g. /get_leaderboard?category=1&sort=alpha	Gets the leaderboard with specified parameters	Crashes - I put default category id = '1' instead of 1 meaning no database entries were found	Check leaderboard route works	Fail - but easily fixed 
/get_leaderboard route no parameters e.g. /get_leaderboard	Runs with default parameters, any% and sort by time	Works	Check no parameter route run works	Pass
/get_leaderboard route non-real parameters E.g. /get_leaderboard?category=FAKECATEGORY&sort=FAKESORT	Should run with default parameters	Crashes - I was setting category name to default if the inputted one doesn't exist, but I didn't change category itself	Check invalid parameters don't crash anything	Fail - fixed now

/get_leaderboard route boundary cases E.g. /get_leaderboard?category=-1&sort=times /get_leaderboard?categoryy=1&sort=time /get_leaderboard?category=8&sort=alpha&boundary=HIII	Should take all parameters that are valid and use them but otherwise revert to defaults. E.g. /get_leaderboard?category=-1&sort=time Should sort by time but default to id 1 for category	Works	Check boundaries	Pass  No inputs because arb category has no completed runs
/signup route logged out	Goes to signup page	Works	Check route works	Pass 
/signup route with extra parameters E.g. /signup?EXTRATHING=HELLOOOO /signup/1/2/3	Goes to signup page	Works	Check route works no unexpected errors on more parameters	Pass 
/signup route logged in	Should not work, goes to home page	Works	Check you can't signup while already logged in	Pass
/signin route logged out	Goes to sign in page	Works	Check route works	Pass 
/signin route with extra parameters	Goes to signup			

E.g. /signin?EXTRATHING=HELLO				
/signup route extra slashes E.g. /signup/1/2/3	404	Works	Check it will 404	Pass
/signin route extra slashes E.g. /signin/1/2/3	404	Works	Check it will 404	Pass
/signin route logged in	Should not work, goes to home page	Works	Check you can't log in while already logged in	Pass
/logout route logged in	Logs user out	Works	Check you can log out	Pass
/logout route not logged in	Does nothing	Works	Check logging out while not logged in doesn't break anything	Pass
/new_user route	405 method not allowed	Works	Check you can't make new users with the url	Pass
/new_user with extra parameters E.g. /new_user/1/2/3	404	Works	Check nothing breaks	Pass
/search route	405 method not allowed -	Works	Check you can't search	Pass

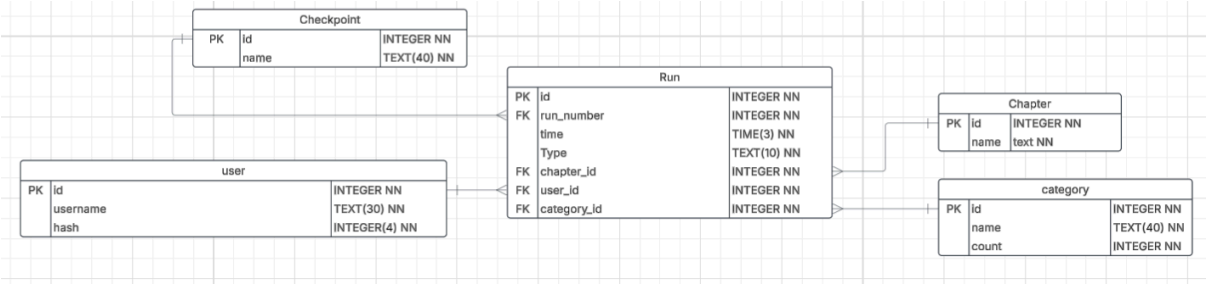
	shouldn't be able to run this from url		with no actual input	
/search route with extra parameters E.g. /search/1/2/3	404	Works	Check nothing breaks	Pass
/pfp route logged out E.g. /pfp/27/1	405 - shouldn't be able to change pfp in this way	Works	Check you can't change pfp in url	Pass
/pfp route logged in	405 - same reason	Works	Same reason	Pass
/socials route	405	Works	Same reason	Pass
/add_socials	405 - same reason	Works	Same reason	Pass
/descriptioner route	405	Works	Same reason	Pass
/get_times route logged out empty	404	Works	This route doesn't exist	Pass 
/get_times route logged out valid parameters E.g. /get_times/27/1	404	Works	Not allowed to go to a users account not logged in	Pass
/get_times route logged out invalid parameters E.g. /get_times/HELLO/50	404	Works	Same as above, and parameters are invalid	Pass 

/get_times/1/fish				
/get_times route logged out boundary case parameters E.g. /get_times/-1/1 /get_times/18/1 (user_id 18 doesn't exist) /get_times/1/-1	404	Works	Same as above, no breaking with boundaries	Pass
/get_times route logged in valid parameters E.g. /get_times/27/1	Goes to specified route	Works	Route should work	Pass 
/get_times route logged in invalid parameters E.g. /get_times/HELLO/50 /get_times/1/fish /get_times/1/50	404s	Ones with only integers will crash it as they still run the route but query returns none.	Route should be invalid	Fail - needed to give default parameters Fixed
/get_times route logged in boundary case parameters E.g.	404s	Doesn't 404. Funny thing I realised is that if you are logged in, it meant you could log into	Route should be invalid	Fail Fixed

/get_times/-1/1 /get_times/18/1 (user_id 18 doesn't exist) /get_times/1/-1		anyones account. This has now been fixed, and it 404s		
/get_times route logged in empty	404s	Works	Route should be invalid	Pass
/update_times route E.g. /update_times/27/1 (valid) /update_times/50/500 (invalid)	Doesn't really matter if parameters are invalid should 405 error	Works	User shouldn't be able to do this from url	Pass
/profile route no parameters	404	Works	Route should be invalid	Pass
/profile route valid parameters E.g. /profile/27/1	Goes to the entered route	Works	Should work	Pass
/profile route invalid parameters E.g. /profile/hello/fish /profile/50/hi	404	Works	Route should be invalid	Pass
/profile route boundary case parameters E.g. /profile/1/-1	404	Works	Route should be invalid	Pass

Database iterations:

During the planning phase, this was my final iteration of the database.



Iteration 1: Count column removed from category - unnecessary.

Iteration 2: Added CategoryCheckpoint bridging table to database.

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	category_id	INTEGER								NULL
2	checkpoint_id	INTEGER								NULL
3	orderer	INTEGER								NULL

This table defines which checkpoints are in each category, and they are ordered by the orderer column. I added this because in each category section of the profile, I wanted the checkpoints to be ordered by the way you run the category, instead of just random checkpoints everywhere. This table allows me to find the correct order everything should be in.

Iteration 3: Added chapter_order column to chapter table - needed to order chapters correctly within the chapter sum of best table and for the graph.

Iteration 4: Added date_joined, pfp_path, and description to user.

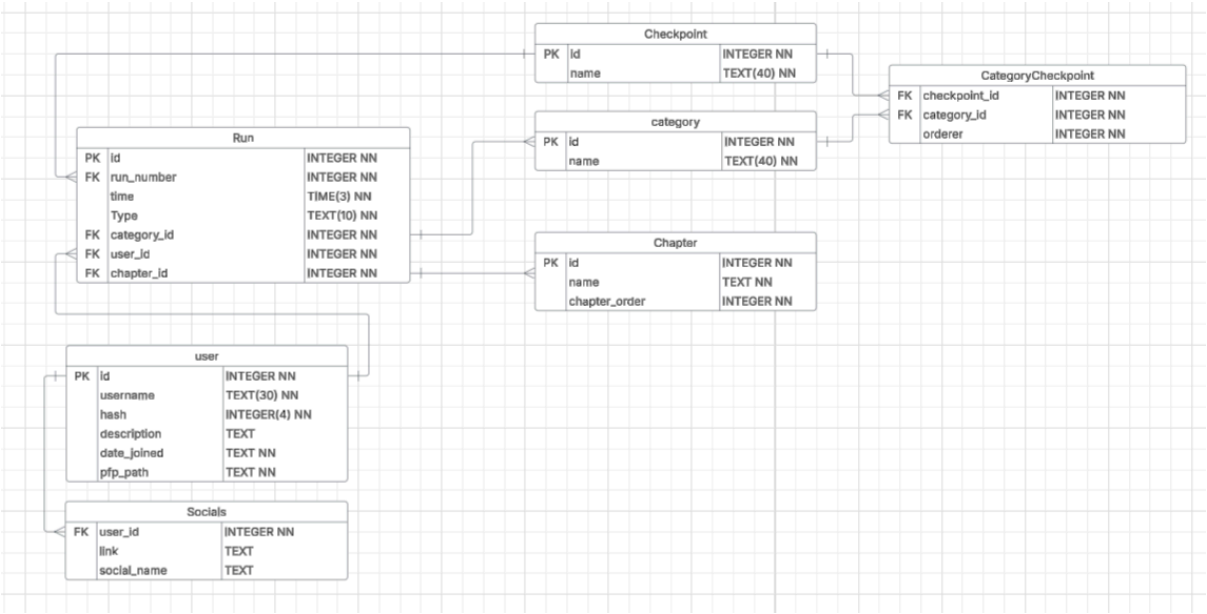
	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	id	INTEGER								NULL
2	name	TEXT								NULL
3	hash	TEXT								NULL
4	description	TEXT								NULL
5	date_joined	TEXT								NULL
6	pfp_path	TEXT								NULL

Needed to store user account info.

Also added socials table:

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	user_id	INTEGER								*NULL*
2	link	TEXT								*NULL*
3	social_name	TEXT								*NULL*

Used to store multiple socials per user with their name. This is necessary because otherwise a user could only have a set amount of socials, which I would have to put columns in the user table for. This is a better solution.



Project final ERD - with all changes implemented.

Data Integrity:

Time integrity:

RidgeA	
Start	00:37.060
Shrine	00:16.813
Old Trail	00:54.059
Cliff Face	00:43.911
Total	02:31.843

This is the user ‘ben’s times for RidgeA in the Any% category.

27	ben	c20f9025a70ed23b81c0b811b2aa2de1ba026ca3e1d4056fb2aebab9a71ac307	[illegible]

Here is the database row for ben.
He has id 27.

1726	1726	11	37.060	checkpoint	10	27	1
1727	1727	12	16.813	checkpoint	10	27	1
1728	1728	13	54.059	checkpoint	10	27	1
1729	1729	14	43.911	checkpoint	10	27	1

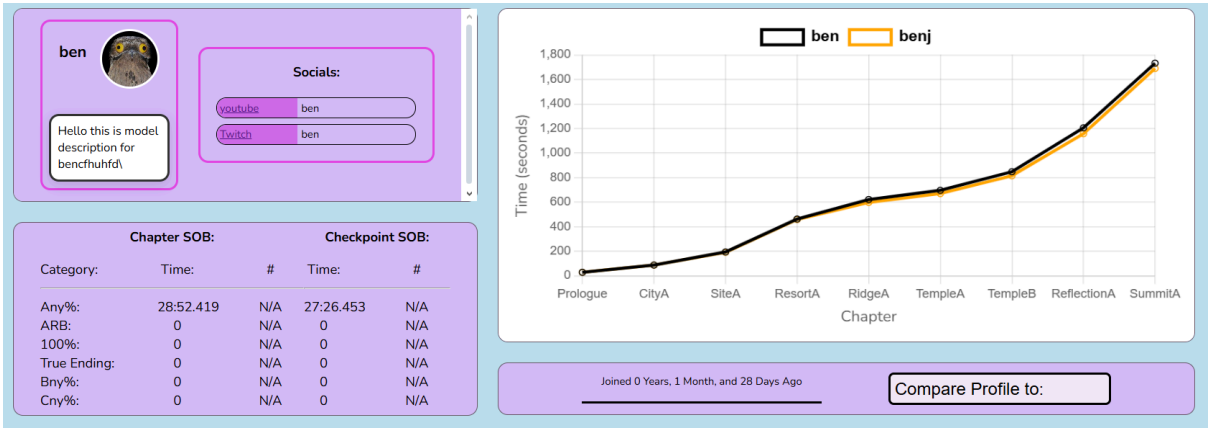
Here are the database rows for this. All of the data reflects what is displayed on the screen, so it is correct. The 10 column is saying chapter ID 10, which is RidgeA.

10	10	RidgeA	8

The 1 column is saying category id 1, which is Any%.

1	1	Any%

User information integrity:



Above is the profile information for ben that the app displays.

This is located on user id 27 from the URL

	id	name	hash
1	19	bobby	69ff8042237aeef5ddf1ced10e851bc6104e1ebb15a304061dba43e
2	20	samuel	aaf5ad63ac417e5002bdac202e07287cf90f35b1d419464d2c4fc79
3	21	awd awd	71a6ed6fec5ecf244123232857be1eda630667940fa284e182bdb1
4	22	awf	ec050389058e19c20fb882b91bc769f4e0fe23499b50697a55df9c5
5	23	john	930a68a51a2db950f58fd3b0b5f1d76f56afaa16e12a418d71ca6c2
6	24	sfgsj	8c135c6ac6d64a2b2f5744da9e6af8740d4f806575d526144c3533f
7	25	matthew	d38681074467c0bc147b17a9a12b9efa8cc10bcf545f5b0bccccf5a9
8	26	benj	04fa7274c8ae8aabbeb004d22b9c88f5968565e67e7464ebb76469
9	27	ben	c20f9025a70ed23b81c0b811b2aa2de1ba026ca3e1d4056fb2aeba

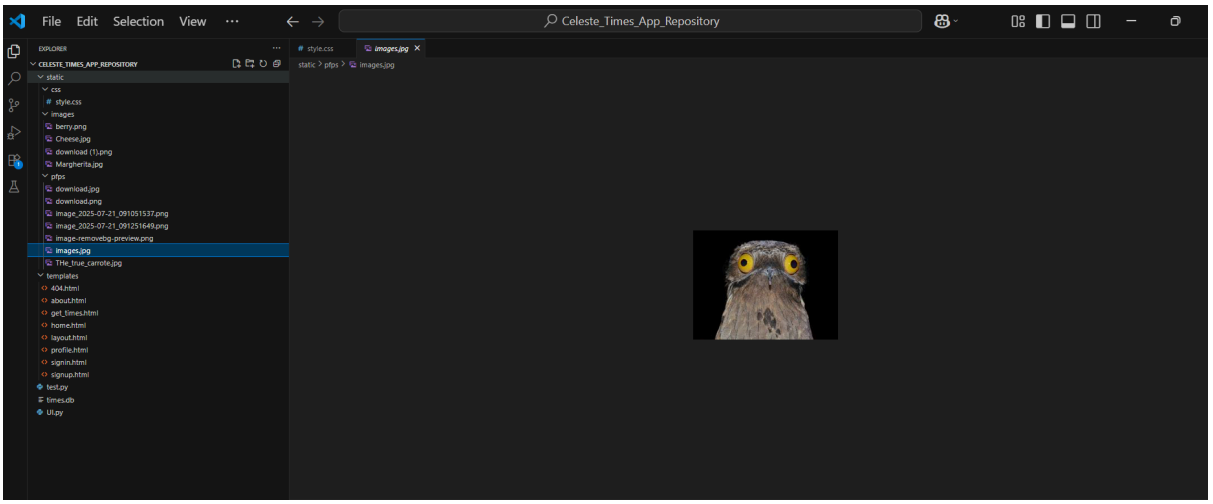
Here is ben in the database. His id is 27.

Hello this is model description for bencfhuhfd\	2025-07-13 21:21:29.720177	images.jpg

Here is ben's other information in the user table. The current date is the 11th of September, and the database says he joined on the 13th of July. This is exactly 1 month and 28 days difference like stated on the app.

The description also matches from database to app.

Ben's profile picture is image.jpg.



As you can see, images.jpg is this potoo, which is the same profile picture as displayed in the app.

user_id	link	social_name
23	https://www.twitch.tv/john	Twitch
23	https://www.youtube.com/john	Youtube
25	jim	hello
27	https://youtube.com/ben	youtube
25	><>><	12<>><
27	twitch.tv/logan_gardiner	Twitch

Here is the social table. The two entries with user_id of 27 are bens socials. The names of the socials are youtube and Twitch. This matches what is displayed on the app. If I click the links they take me to these URLs.

This is exactly what the link says. The twitch.tv/logan _gardiner has no https:// at the start so it continues on from the profile page that it was already on.

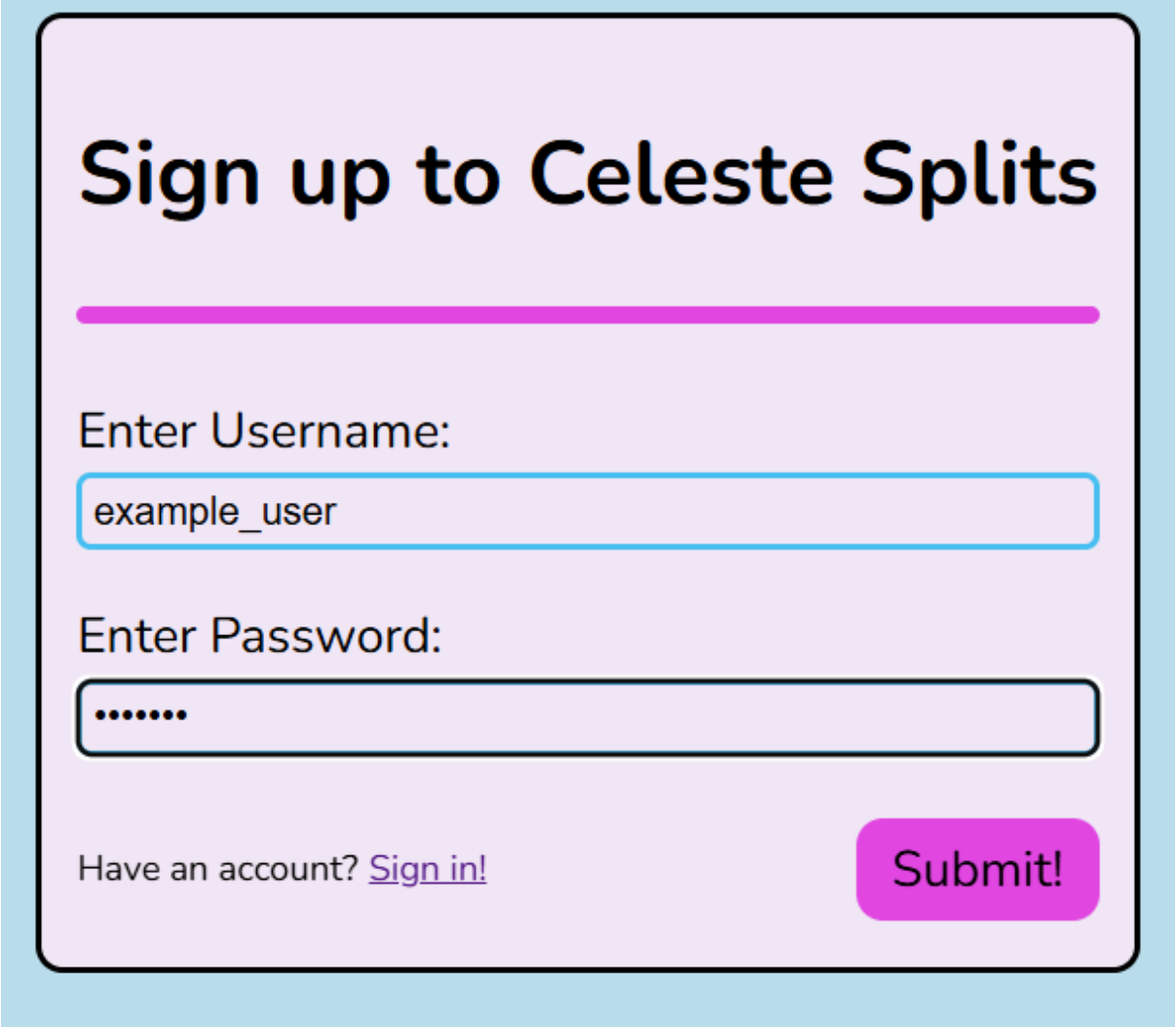
The app also says that ben's chapter SOB for any% is 28:52.419. This should be equal to all the chapter times for any% added together.

	id	run_numt	time	type	chapter_ic	user_id	category_
1745	1745	49	53.227	checkpoint	14	27	1
1746	1746	50	33.667	checkpoint	14	27	1
1747	1747	51	45.814	checkpoint	14	27	1
1748	1748	92	35.921	checkpoint	13	27	1
1749	1749	102	28.509	checkpoint	26	27	1
1750	1750	-1	58.888	IL	1	27	1
1751	1751	-1	106.828	IL	4	27	1
1752	1752	-1	268.226	IL	7	27	1
1753	1753	-1	159.205	IL	10	27	1
1754	1754	-1	74.766	IL	13	27	1
1755	1755	-1	357.697	IL	16	27	1
1756	1756	-1	526.558	IL	19	27	1
1757	1757	-1	151.742	IL	14	27	1
1758	1758	-1	28.509	IL	26	27	1
1759	1759	1	0	checkpoint	1	27	2

58.888+106.828+268.226+159.205+74.766+357.697+526.558+151.742+28.509 = 1732.419s = 28.87365 min = 28m 52.419s = The exact time displayed on the app.

Signing up as user:

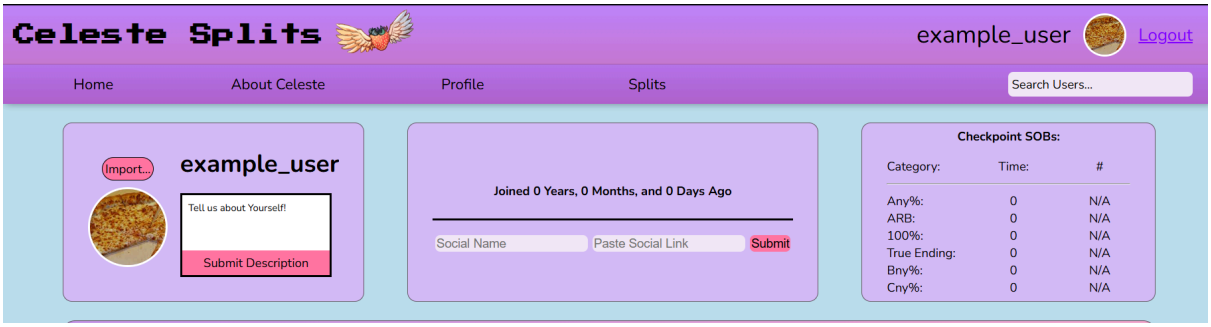
Here is a user signing up:



Here is that user in database:

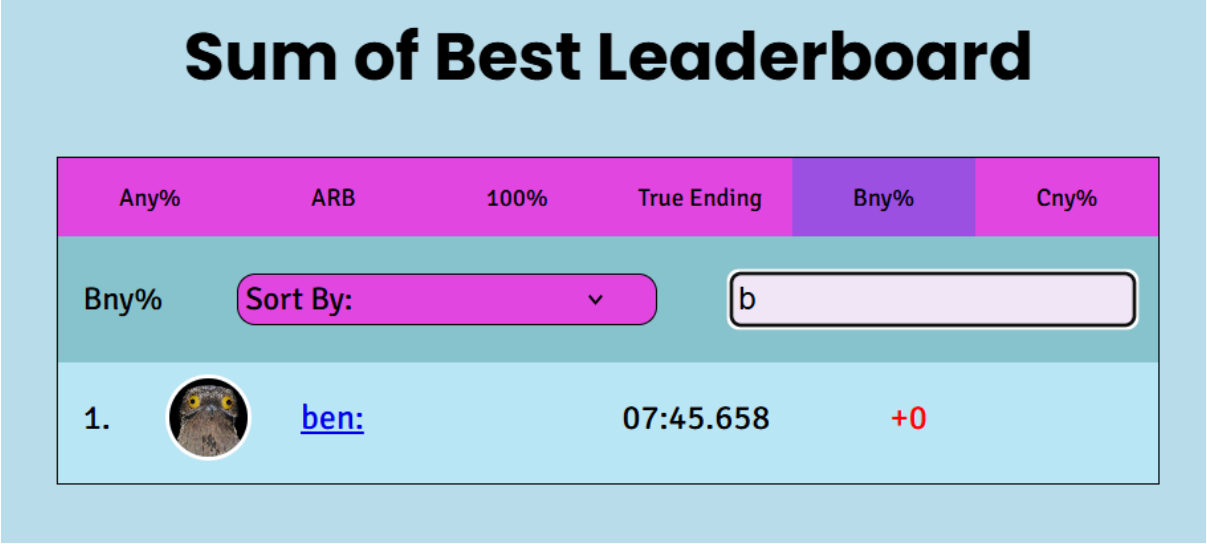
41	example_user	50d858e0985ecc7f60418aaf0cc5ab587f42c2570a884095a9e8cca

After signin:



All default data is there.

Leaderboard Data Integrity:



Here is the entry for the user, ben, in the Bny% category.

id	name
19	bobby
20	samuel
21	awd awd
22	awf
23	john
24	sfgsj
25	matthew
26	benj
27	ben

Here is the user ben in the database. He has id 26.

id	name
1	Any%
2	ARB
3	100%
4	True Ending
5	Bny%

Here is the Bny% category in the database. It has id 5.

Here are all the entries for user_id = 26 and category_id = 5 in the Run table.

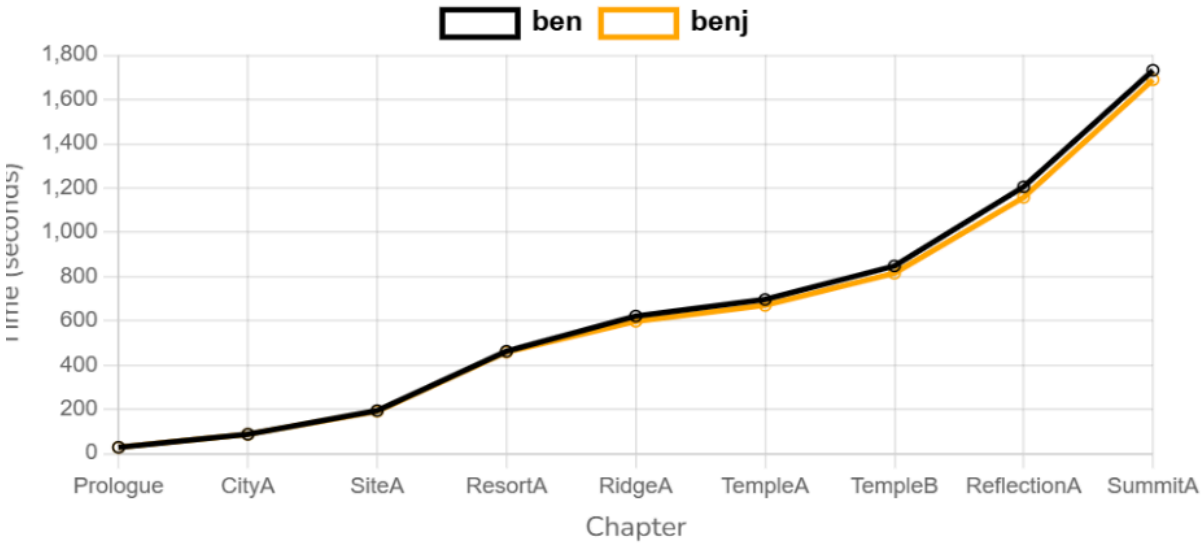
1980	1980	1	10.123	checkpoint	1	27	5
1981	1981	7	10.123	checkpoint	7	27	5
1982	1982	8	10.123	checkpoint	7	27	5
1983	1983	15	10.123	checkpoint	13	27	5
1984	1984	18	10.123	checkpoint	16	27	5
1985	1985	19	10.123	checkpoint	16	27	5
1986	1986	27	10.123	checkpoint	19	27	5
1987	1987	28	10.123	checkpoint	19	27	5
1988	1988	29	10.123	checkpoint	19	27	5
1989	1989	34	10.123	checkpoint	2	27	5
1990	1990	35	10.123	checkpoint	2	27	5
1991	1991	36	10.123	checkpoint	2	27	5
1992	1992	37	10.123	checkpoint	5	27	5
1993	1993	38	10.123	checkpoint	5	27	5
1994	1994	39	10.123	checkpoint	5	27	5
1995	1995	40	10.123	checkpoint	8	27	5
1996	1996	41	10.123	checkpoint	8	27	5
1997	1997	42	10.123	checkpoint	8	27	5
1998	1998	43	10.123	checkpoint	8	27	5
1999	1999	44	10.123	checkpoint	11	27	5
2000	2000	45	10.123	checkpoint	11	27	5

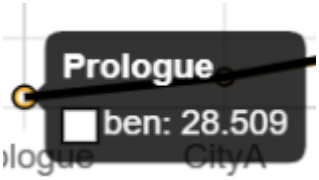
	id	run_numbe	time	type	chapter_id	user_id	category_i
2001	2001	46	10.123	checkpoint	11	27	5
2002	2002	47	10.123	checkpoint	11	27	5
2003	2003	48	10.123	checkpoint	14	27	5
2004	2004	49	10.123	checkpoint	14	27	5
2005	2005	50	10.123	checkpoint	14	27	5
2006	2006	51	10.123	checkpoint	14	27	5
2007	2007	52	10.123	checkpoint	17	27	5
2008	2008	53	10.123	checkpoint	17	27	5
2009	2009	54	10.123	checkpoint	17	27	5
2010	2010	55	10.123	checkpoint	17	27	5
2011	2011	56	10.123	checkpoint	20	27	5
2012	2012	57	10.123	checkpoint	20	27	5
2013	2013	58	10.123	checkpoint	20	27	5
2014	2014	59	10.123	checkpoint	20	27	5
2015	2015	60	10.123	checkpoint	20	27	5
2016	2016	61	10.123	checkpoint	20	27	5
2017	2017	62	10.123	checkpoint	20	27	5
2018	2018	88	10.123	checkpoint	1	27	5
2019	2019	89	10.123	checkpoint	4	27	5
2020	2020	90	10.123	checkpoint	7	27	5
2021	2021	91	10.123	checkpoint	10	27	5
2022	2022	92	10.123	checkpoint	13	27	5
2023	2023	93	10.123	checkpoint	16	27	5
2024	2024	94	10.123	checkpoint	19	27	5
2025	2025	102	10.123	checkpoint	26	27	5

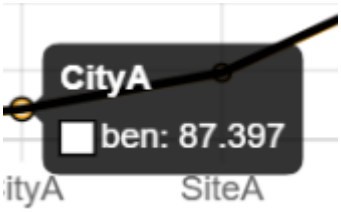
Total time is equal to (2025-1980)*10.123 (all times are the same)
= 46*10.123=465.658/60=7.760966667 0.760966667*60=45.65800002=45.658s
Total = 7m 45s 658ms = What the leaderboard says. Therefore the leaderboard is accurate.

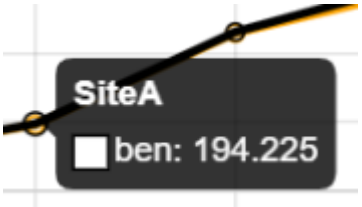
Graph Data Integrity

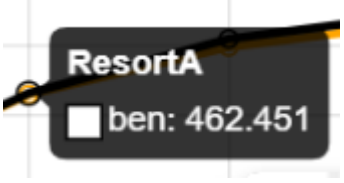
The graph shows discrete points for each user of the cumulative time for their chapters. If the data is accurate, each point should represent the sum of the chapter times before and the current chapter. Here is the graph comparing benj and ben.

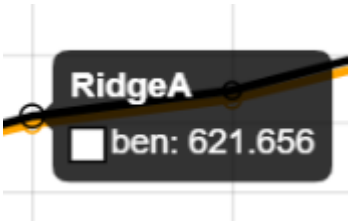


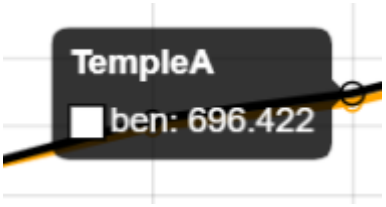


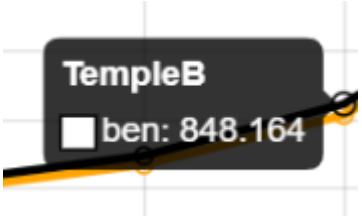


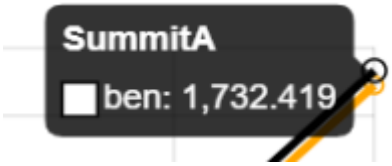












These times are cumulative. Therefore the actual time for city is city - prologue, for site = site - city - prologue.

Therefore the times for each are:
Prologue: 28.509
City: 58.888
Site: 106.828
Resort: 268.226
Ridge: 159.205
Temple A: 74.766
Temple B: 151.742
Reflection: 357.697
Summit: 526.558

Now checking the database for any% (category id 1) and user ben (user id 27) here are the actual chapter IL runs.

1750	1750	-1	58.888	IL	1	27	1
1751	1751	-1	106.828	IL	4	27	1
1752	1752	-1	268.226	IL	7	27	1
1753	1753	-1	159.205	IL	10	27	1
1754	1754	-1	74.766	IL	13	27	1
1755	1755	-1	357.697	IL	16	27	1
1756	1756	-1	526.558	IL	19	27	1
1757	1757	-1	151.742	IL	14	27	1
1758	1758	-1	28.509	IL	26	27	1

These times all match up so everything is right.

Relevant Implications:

Database Relevant Conventions:

Ethical Implications:

Databases have some relevant ethical effects as storing incorrect information could lead to misleading users or outputting unfair results. In a database, it's important to not only make sure data is accurate, but also that its output is presented and used in a fair and expected manner.

In my project, I've made sure that these ethical concerns are addressed. An example of how I have done this is by making sure that all entries on the leaderboard have been fully completed. In a full category run (like the entries on a leaderboard), it's critical that the run is fully completed. For instance, if a user only filled in half of the relevant times, the overall time for the entry would be much lower than it should be given they fully fill it in. I addressed this issue by only adding category entries to the leaderboard if every checkpoint within the category has a valid non-zero time. This means they must have filled it all in, and the time is valid. This addresses the ethical concerns that could arise from the leaderboard, as some users with incredibly fast times that are valid might feel that the system is unfair when they get beaten by someone who only filled in 1 checkpoint and got an impossible first place entry. This system maintains fairness within the ranked system and thus is ethically correct.

Privacy Implications:

It’s extremely important that data on a database is private, as it could contain information that is private to a user, like passwords, emails, phone numbers, etc. It’s crucial that this data cannot be in any way accessed through the site itself, and it’s also important that no person can maliciously get to that data through SQL Injections or the like. Information that must remain confidential should be encrypted within the database so that even if it were to be obtained, it cannot be read. Doing these things make sure that the database complies with privacy expectations, and maintains its user’s trust, as the data shared with the website should be confidential and not able to be accessed by anyone else.

I addressed the privacy implication through the use of hashing the sensitive information of passwords. When a user signs up and creates an account, they enter a password, sharing a very private piece of information that should only be known to them. Although the password to a speedrunning website could be considered irrelevant, it's important to consider that the user may be using the same password for this website as something very important, like banking, google, etc (although they should never do this!). To make sure that this password remains confidential the website will hash the password and enter the hash into the database. A hashing function turns an input into a 256 bit string (I'm using the algorithm SHA-256). This process will always give the same hash from the same input, and an important property is that a hash can not be turned back into its original input (with current day computing power). This hash is stored in the database, and when a user wants to log in, the entered password is hashed and compared to the database hash. This method makes sure that even if my database is hacked, which it shouldn’t be as SQL injections will not work, the hash cannot be restored to the original password, making that password private.

Website Relevant Conventions:

End User:

It’s important to consider the end user when doing web design, as the usability and aesthetic of the website should be suited to fit the needs of the users that are intended to be using it. This means making sure navigation and usage of the site reflect the expectation of the end user, making it easier for them to understand how to work and better at doing its purpose.

My website is a Celeste speedrunning time tracker application. Its end users are speedrunners who want to move their time tracking from the cluttered Google Spreadsheets to a clean and minimalistic site which can track their times in the same way. Because of this, users will have an expectation that the usage of my website is similar to that of Google Spreadsheets. An example of how I addressed this is in time inputting. In Spreadsheets, users store their times in the Google standard time format, mm:ss.msmsms. I made my time inputs also store and accept times in this way, so that the end user doesn’t have to learn a new method of storing times, making the website easier to use.

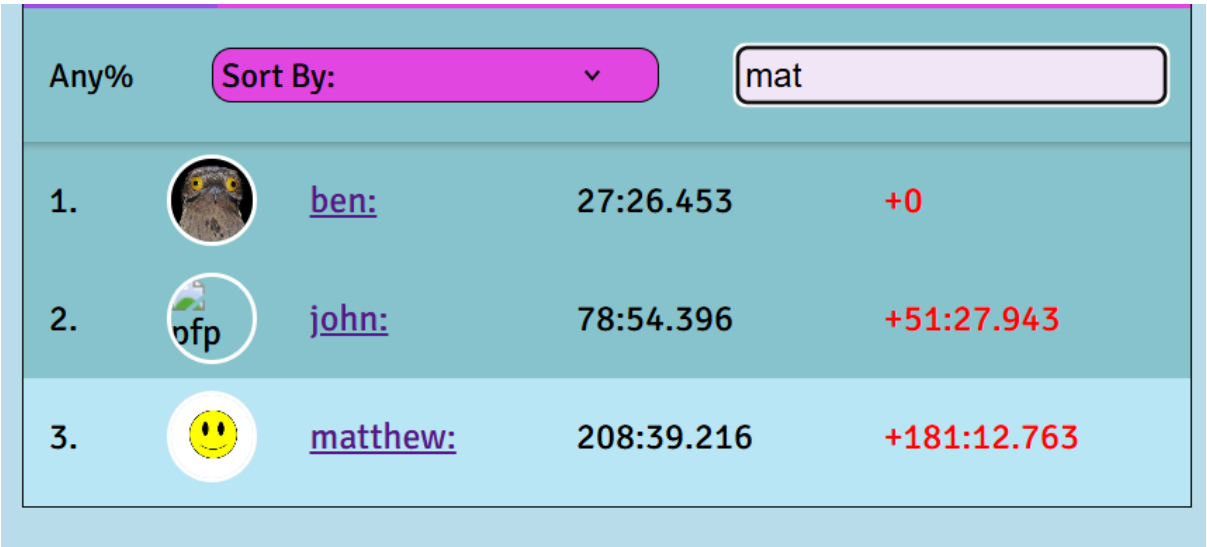
Aesthetics:

Aesthetics are a very important aspect of web design as the overall look and feel of a website can directly influence how a user perceives and forms opinions about it. A well designed aesthetic should portray the purpose of the website while also maintaining its feel. This includes colour schema, which can greatly affect how a user perceives a website and thus affects its productivity, but also its layout, by making the focus of the website the most relevant and attention catching.

For my website, I tried to bring these ideas together, making the feel of the website fit to the Celeste aesthetic, but also by trying to bring important speedrunning elements to the forefront. To fit the Celeste theme, I used light blues, pink and purples to fit the vibrant and playful feel of Madeline (the main character). Doing this makes the website feel connected to the game itself, which increases the interactivity for Celeste speedrunners as you feel more connected to the game while using it. I also tried to keep the focus of the website on speedrunning by maximising the focus on speedrun related elements. An example of this is on the splits page. I used a grid layout for all the categories to create a clean and minimalistic layout that put emphasis on the time input fields; the focus of the page. These design choices make sure that the most important features are emphasised, while the Celeste inspired aesthetic connects the app with the game. These features keep the aesthetic of the website consistent with its purpose, tracking Celeste speedruns.

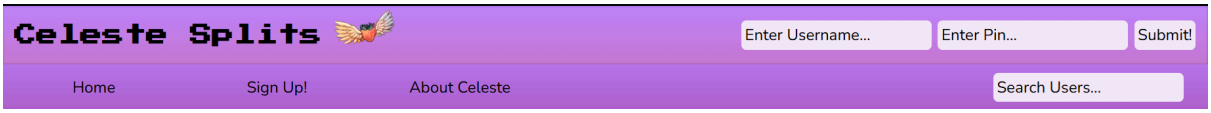
Applying relevant conventions to improve the quality of my website:

I utilised Nielsen's heuristic of recognition rather than recall in my leaderboard design to help minimise memory load for the user. How I did this was by making it so that a user doesn't have to search the full exact user name to snap to that user's position on the leaderboard. Instead, they only have to type part of it, and press enter, and if the part they typed matches a part of a full username, the leaderboard will snap down to that position and highlight the relevant leaderboard slot. E.g. if you type mat, like in the image below, it snaps to Matthew's position. If this wasn't the case, the user would have to recall the entire username exactly, which could be annoying, especially if they can't get the spelling exactly correct. With this feature, instead the user only has to recognise part of the name, which is very helpful and a good feature.

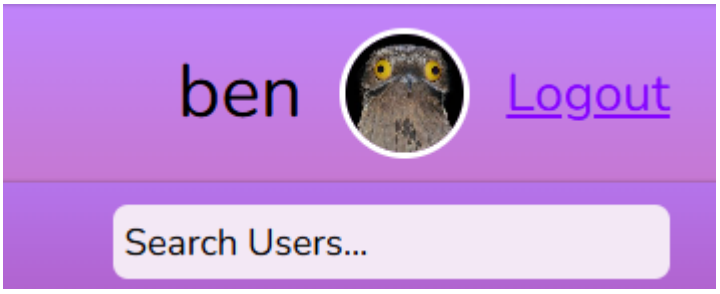


I also improve the external consistency of my website, improving the consistency and standards heuristic of the website. Initially, I had 3 text inputs in my header; one for searching for a user, and 2 for logging in (1 password and 1 username) as seen in the image below. This is different from what regular websites would do, as they would have a separate login page so as not to confuse the user by overwhelming them and confusing them. When there are 3 inputs it is confusing for the user for where to type their login, or the name of another user who they want to view their profile. To improve my external consistency, I got rid of this design, and instead made a login page, and only put the search to profile input in the header. This will make the site more user friendly as they will no longer get confused by the weird layout, and instead will know how to use the login page and single header input as that is very similar to what other websites do.

Before:



After:



Credits:

https://commons.wikimedia.org/wiki/File:Celeste_strawberry_with_wings.png

Final Project Reflection:

I’m pretty happy with this project overall. I think I managed to complete all the goals I set for myself, and learnt a few new things while doing it. I’m very happy with the way the profile page ended up, but think that my overall layout design on most other pages could do with improvement. For next year, I want to learn some more javascript, and really work on making my website look more professional.