Link to Github Repo

View the repo on GitHub

Styling

PEP8 Styling was used. -- how was it used? small para

Referenced sources

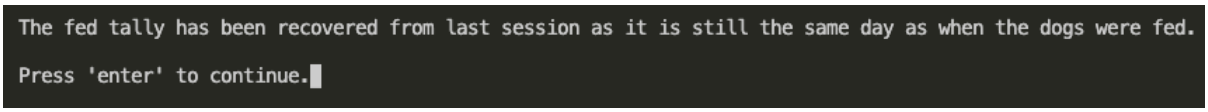
- Ed
- Documentation
- Youtube
- Classes

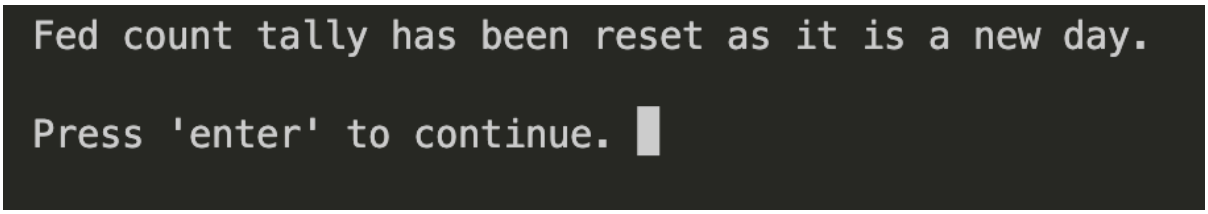
Dog Shelter Feeding Helper

- Application to manage a dog shelter's daily feeding schedule - keeping track of the feeding & the dietary information of dogs that come in.
- *The 'fed' status of the dog will reset each day*. The dogs name & information will be assigned a 'DoggyID' and remain in the database until it has been deleted (adopted).

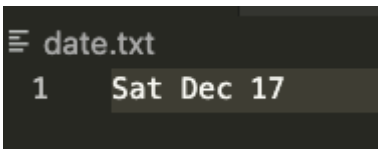
List of features

1. **Continuity message to indicate to user if fed count has been reset.**





   - On opening the application, a message will indicate if the fed count has been reset (new day) or carried on from last session (same day)
   - Using a separate file `date.txt`, a date in `%a %b %d` format (eg. Sat Dec 17) will be checked and compared with the systems date in the same format.



   - Using `if` statements, the fed counter remains the same from the last session if the dates match, aka it is the same day. The fed counter variable will reset to 0 if it recognises that it is a new day, done by changing all of the 'Fed' key values from 'Yes' to 'No' in the json

file `dogsdb.json` on recognition of it being a different date. In both situations regardless, `date.txt` is written over and replaced with the current date at time of use.

```
# Check if date stored in date.txt matches time
# pulled from datetime module (format inclusive).
def check_if_new_day():
    if ("".join(open("date.txt").read().split("\n"))) == (
            now.strftime(tz_fmt_for_check)):
        check_today = input(
            "The fed tally has been recovered from last session as it is "
            "still the same day as when the dogs were fed."
            "\n\nPress 'enter' to continue.")
    else:
        update_to_no()
        get_ok = input(
            "Fed count tally has been reset as it is a new day."
            "\n\nPress 'enter' to continue. ")
```

```
# Update all 'fed' results in database to no.
# To be called if check if new day defaults to else statement
def update_to_no():
    results = db.search(User.Fed == "Yes")
    for res in results:
        res["Fed"] = "No"
    db.update({"Fed": "No"}, Query().Fed.exists())
```

2. **Display of count of dogs in shelter, count of dogs that have been fed** (eg. 15/20 dogs have been fed today). *Count of dogs that have been fed will reset on a new day.*

```
_____________________________________________
Sat Dec 17 2022
20:07:17

Number of dogs that have been fed today:

.d888b.          dD     d8888b.
VP  `8D         d8'     VP  `8D
   odD'        d8'        oooY'
 .88'         d8'          ~~~b.
j88.         d8'         db   8D
888888D     C8'          Y8888P'




_____________________________________________
```

```
# Pulls fed status from database and updates tally and global variable
def fed_dog_counter():
    results = db.search(User.Fed == "Yes")
    db_fed_dog_tally = len(results)
    global fed_dog_count_main
    fed_dog_count_main = db_fed_dog_tally
```

- Shows current date and time using `datetime` from the in built python module.
- If all dogs are fed, the user gets a message on the display indicating that all dogs have been fed today.
- Uses pyfiglet to create a more 'eye catching' counter for the user.

```
# Displays amount of fed dogs. If all dogs are fed then
# another string is printed.
def fed_dog_count_tally():
    total_dogs_count_main = len(db)
    fed_dog_counter()
    # pyfiglet used to make counter stand out
    print("\nNumber of dogs that have been fed today: \n\n"
    + (pyfiglet.figlet_format(str(fed_dog_count_main)
    + "  /  "
    + str(total_dogs_count_main), font="basic",)))

    if str(fed_dog_count_main) == str(total_dogs_count_main):
        print(LINEBREAK_GRAPHIC)
        print("All dogs have been fed for today.")
```

3. **Menu to view which actions the user wishes to take. Entering an input takes the user to that menu option.**

```
------------------------------------------------------------

1. View dogs in shelter
2. Add dog information
3. Edit dog information
4. Update fed status
5. View dogs still to be fed
6. Delete dog from database
7. Exit

------------------------------------------------------------


Enter menu choice:
```

- Includes some error handling to catch index or value errors, accounting for the user inputting an invalid option.

```
except (ValueError, IndexError) as the_error:
    print(STARBREAK_GRAPHIC)
    value_error_input_rtm = input(
        f"Invalid error {the_error}. Hit 'enter' to return to menu.\n")
    print(STARBREAK_GRAPHIC)
    main_menu()
```

Menu options include:

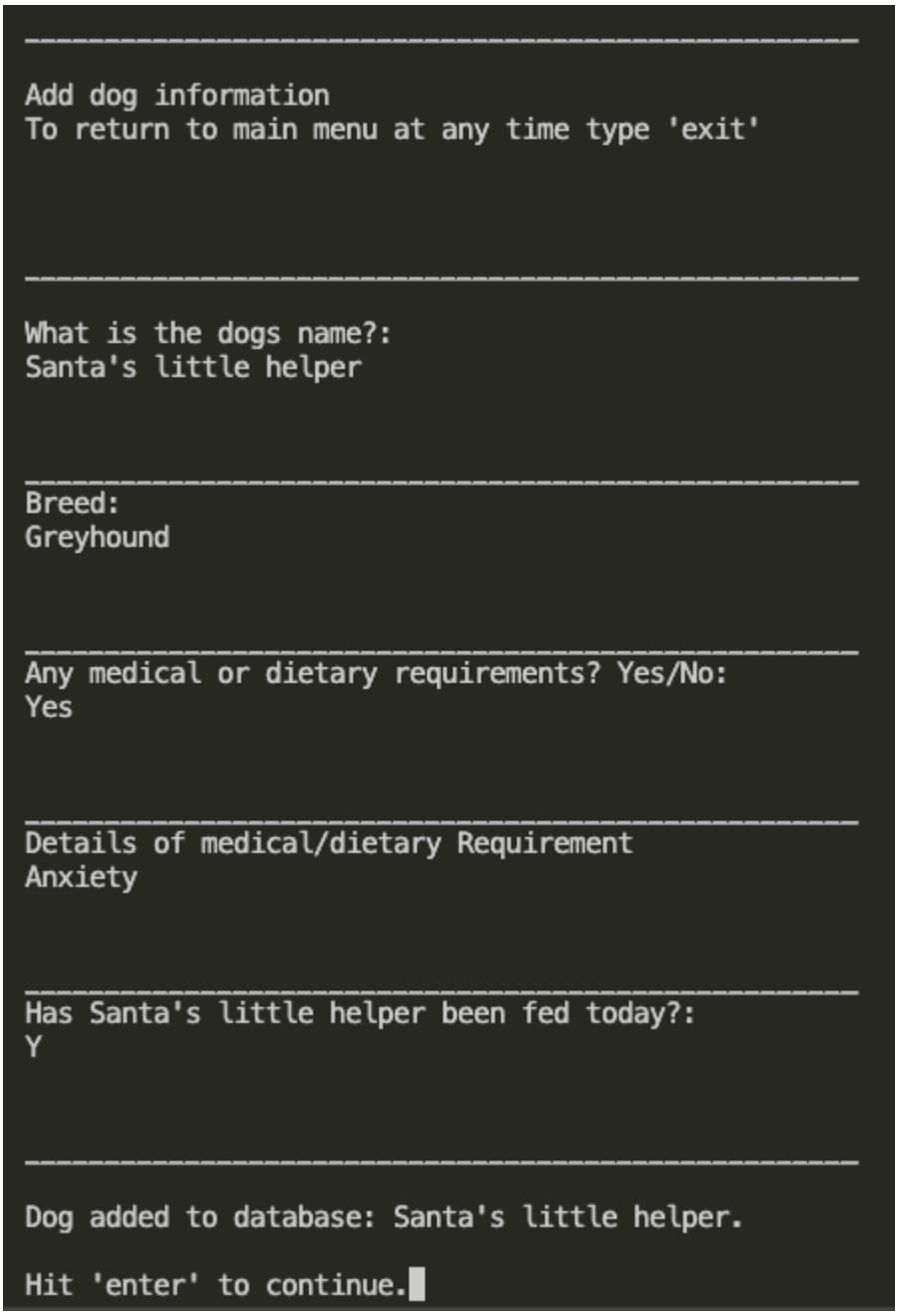
4. **View dogs & information (including dietary requirements/ in readable format)**

```
_____________________________________________


+-----------+----------------------+-----------+-------------------------------+-----------------------------+-------+
|   DoggyID | Name                 | Breed     | Medical/Dietary Requirements? | Details of M/D Requirement  | Fed   |
+===========+======================+===========+===============================+=============================+=======+
|        40 | Scooby               | Great Dane | Yes                           | Really likes scooby snacks  | Yes   |
+-----------+----------------------+-----------+-------------------------------+-----------------------------+-------+
|        41 | Scrappy              | Chihuahua | No                            | N/A                         | No    |
+-----------+----------------------+-----------+-------------------------------+-----------------------------+-------+
|        44 | Santa's little helper | Greyhound | Yes                           | Anxiety                     | Yes   |
+-----------+----------------------+-----------+-------------------------------+-----------------------------+-------+

Dogs in shelter

_____________________________________________
```

```
global dogs
dogs = db.all()

try:
    header = dogs[0].keys()
    rows = [x.values() for x in dogs]
    print("")
    print(tabulate.tabulate(rows, header,
          tablefmt="grid", maxcolwidths=[None, None]))
```

- Uses `tabulate` to show the dogs in an easily viewable format, looping through and pulling the key value pairs stored in the `dogsdb.json` file.

5. **Add a dog & its information to the database**




```
#Collecting the new dogs information
name = input("\nWhat is the dogs name?: \n")
if name.lower() == "exit":
    clear()
    main_menu()
else:
    name = name
```

```
#Assigning the new variables to the keys of new_dog{}
new_dog["DoggyID"] = dog_id
new_dog["Name"] = name
new_dog["Breed"] = breed
new_dog["Medical/Dietary Requirements?"] = medical_requirements
new_dog["Details of M/D Requirement"] = requirement_info
new_dog["Fed"] = has_been_fed

#Putting new_dog{} into dogsdb.json
db.insert(new_dog)
```

- User can add a new dog to the shelter, entering its 'name', 'breed', 'medical and dietary requirements' (yes/no), 'details of medical & dietary requirement' and whether or not the dog has been 'fed'. This information is stored in the dictionary format (in the case of the add dog menu; 'prompt': 'answer') and is stored in a separate json file `dogsdb.json` and used in conjunction with the python package `tinydb`.
- User can exit back to main menu at any time by typing exit.
- *Nice to have* next addition would be to have empty answers return "N/A" or something to that effect. User should also be able to exit using "0" with future updates to functionality. Additionally, a user can currently input a new record which is all empty. This issue needs to be fixed.

6. **Edit dog information in database**

```
_____________________________________________


+-----------+-------------------------+------------+-----------------------------------+--------------------------------+-------+
|   DoggyID | Name                    | Breed      | Medical/Dietary Requirements?     | Details of M/D Requirement     | Fed   |
+===========+=========================+============+===================================+================================+=======+
|        40 | Scooby                  | Great Dane | Yes                               | Really likes scooby snacks     | Yes   |
+-----------+-------------------------+------------+-----------------------------------+--------------------------------+-------+
|        41 | Scrappy                 | Chihuahua  | No                                | N/A                            | No    |
+-----------+-------------------------+------------+-----------------------------------+--------------------------------+-------+
|        44 | Santa's little helper   | Greyhound  | Yes                               | Anxiety                        | Yes   |
+-----------+-------------------------+------------+-----------------------------------+--------------------------------+-------+


[1] Name
[2] Breed
[3] Medical/Dietary requirements
[4] Details of medical/dietary requirement
[5] Has been fed today

[0] Return to main menu

Please choose from above what you would like to edit:
```

```
if selection == "1":
    linebreak()

    edit_name = input(
        "\nWhat would you like to update the dogs name to?: \n")
    #updating the db key with its new value
    db.update({"Name": edit_name}, User.DoggyID == int(dog_to_edit))

    confirm = input("\nUpdated Dog "
        + str(dog_to_edit)
        + "'s name to: "
        + edit_name
        + "\n\nHit 'enter' to continue.\n")
    edit_dog_info_menu()
```

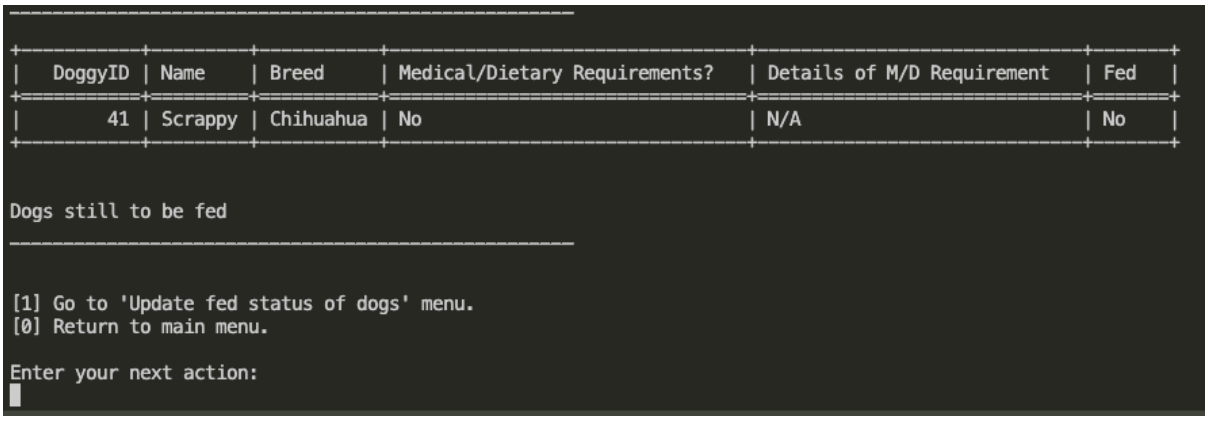
- Using view dogs from above, the user can enter in an identifier 'DoggyID' to select a dog from the database and change their information.
- Using tinydb, a record is matched.
- User then gets a secondary menu with prompts of which key value they would like to change. Users new response overrides previous value for the selected key (menu option), which is then updated in the database.
- User receives confirmation of update.

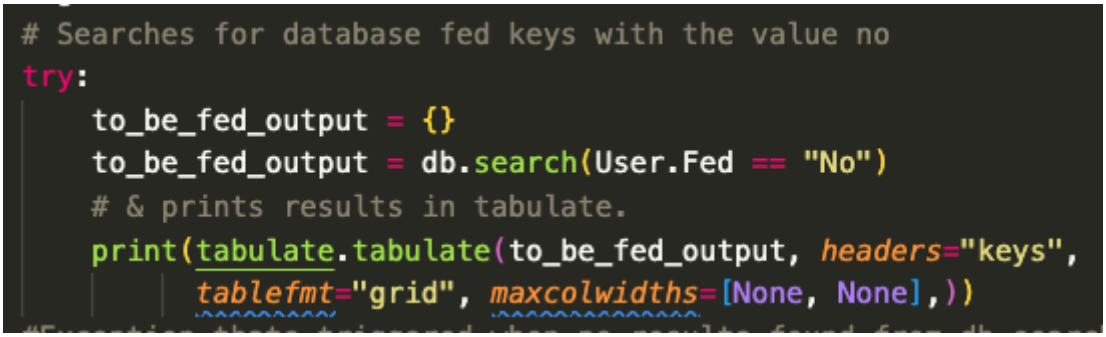
7. **Update the fed status of a dog**

```
Update fed status
_____________________________________________________


Enter [0] to return to main menu.

Enter the DoggyID of dog to update fed status:
```

- Very similar process to menu option 3, however streamlined to update fed status. As with all other ways to update the fed status of the dogs, changing 'fed' value will update the fed count display. Displays 'view dogs' table to see dogs and their fed status.
- User gets yes/no option to update a dogs fed status.
- New page appears confirming change of value.

8. **View dogs still to be fed**





- Returns the dogs that need to be fed, and gives the user the option to go to the 'mark dog as fed' menu option or return to main menu.
- Uses same functionality as view dogs menu option, however filters out to only show dogs with the 'Fed' key value of 'No'.

9. **Remove a dog from the shelter**

```
#Removes entry from db
db.remove(User.DoggyID == int(dog_to_dlt))

print("Dog removed: "
    + str(dog_to_dlt)
    + " - "
    + (dog_rec_to_dlt["Name"])
    + ".")
print("Goodbye "
    + (dog_rec_to_dlt["Name"])
    + "!!!")
```

- Gives the user the ability to remove a dog from the database/shelter. User is able to enter in a 'DoggyID' of an entry that they wish to remove from the shelter. Doing so will remove the entry from the database in `dogsdb.json`.
- Uses view dogs from menu option 1 to showcase dogs in shelter.
- User will be given two forms of confirmation, the first being to hit `enter` if their 'DoggyID' choice is correct, and finally to enter a 'y / n' response to confirm its permanent deletion. Different forms of confirmation used so that the user has to do something 'different' to confirm the deletion of the entry.

10. **Exit the application**

```
  ![Goodbye ASCII](./T1A3-images/23-goodbye-ascii.png)

  - Allows the user to exit the terminal after displaying a goodbye
message to the user.
```

Additional notes

The notion of it being a dog shelter can be interchanged (ie. the application could also work for a dogsitter that looks after a lot of dogs..?) alternatively, a similar model could be used for something like a nursing home medication tracker with some slight change in variables (nursing home residents profile, breakfast, lunch, dinner medication), or kindergarten/boarding school/special needs school meal/feeding schedule.

- *Nice to have additions*:
  - Different shelter locations, each with a database of their own animals
  - Search for dog by name, type or dietary requirement/medication
  - Adding of dog weight and meal size variable

- View *which type of dietary requirements* are required as part of the how many meals/& how many have dietary requirements menu option
- Change to an 'animal' shelter, incorporating different animals (not just dogs) which can be a new variable to sort/view by. Inclusion of 'animal type' variable in information database.

Implementation plan

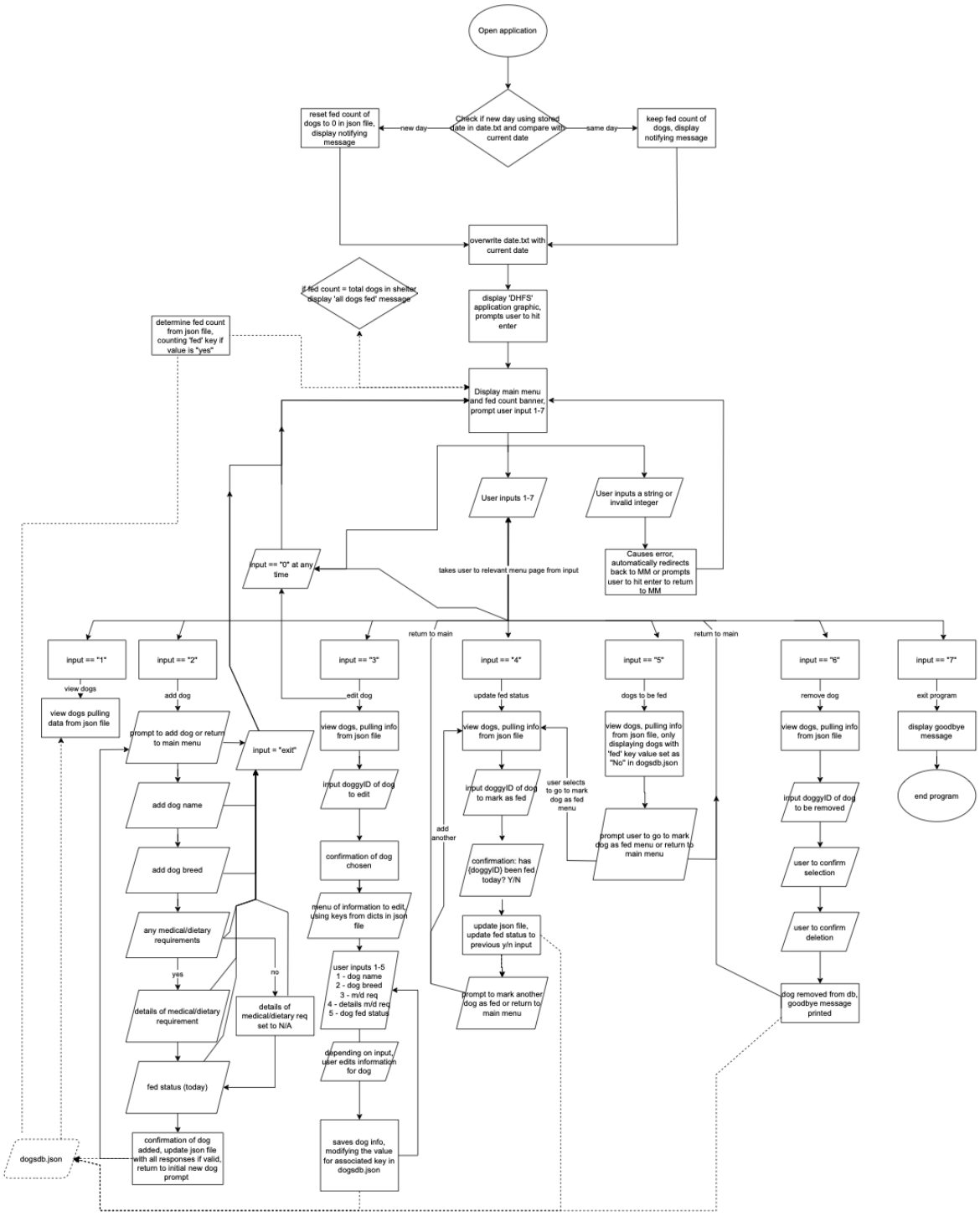
View the trello board here

In the early stages of this project, I knew that I had a lot of catching up to do before proceeding to start the build process. The first few days of the project began with me brainstorming various ideas. After re-evalutating my first idea (an application which would work with a musical keyboard, suggesting a users next key to play in a scale based on certain variables), the idea of a Dog Shelter Feeding Helper came to mind.

An application that is pretty flexible in its use (some small changes in the code can make it very clear that its not strictly for dogs, and can be used as a generalised database type of application) and that would be able to store a users input information, and then change it automatically based off of a certain condition (fed count resetting if session is on a new day).

Because of these features, I knew that I needed to do further research into how a type of database could interact with a python program, and I also had to figure out the challenge of getting the application to check if the program had been used on the same day or a if it was being used on a new, different day.

Into the build process, a flowchart was made to show the logic and operation of the program



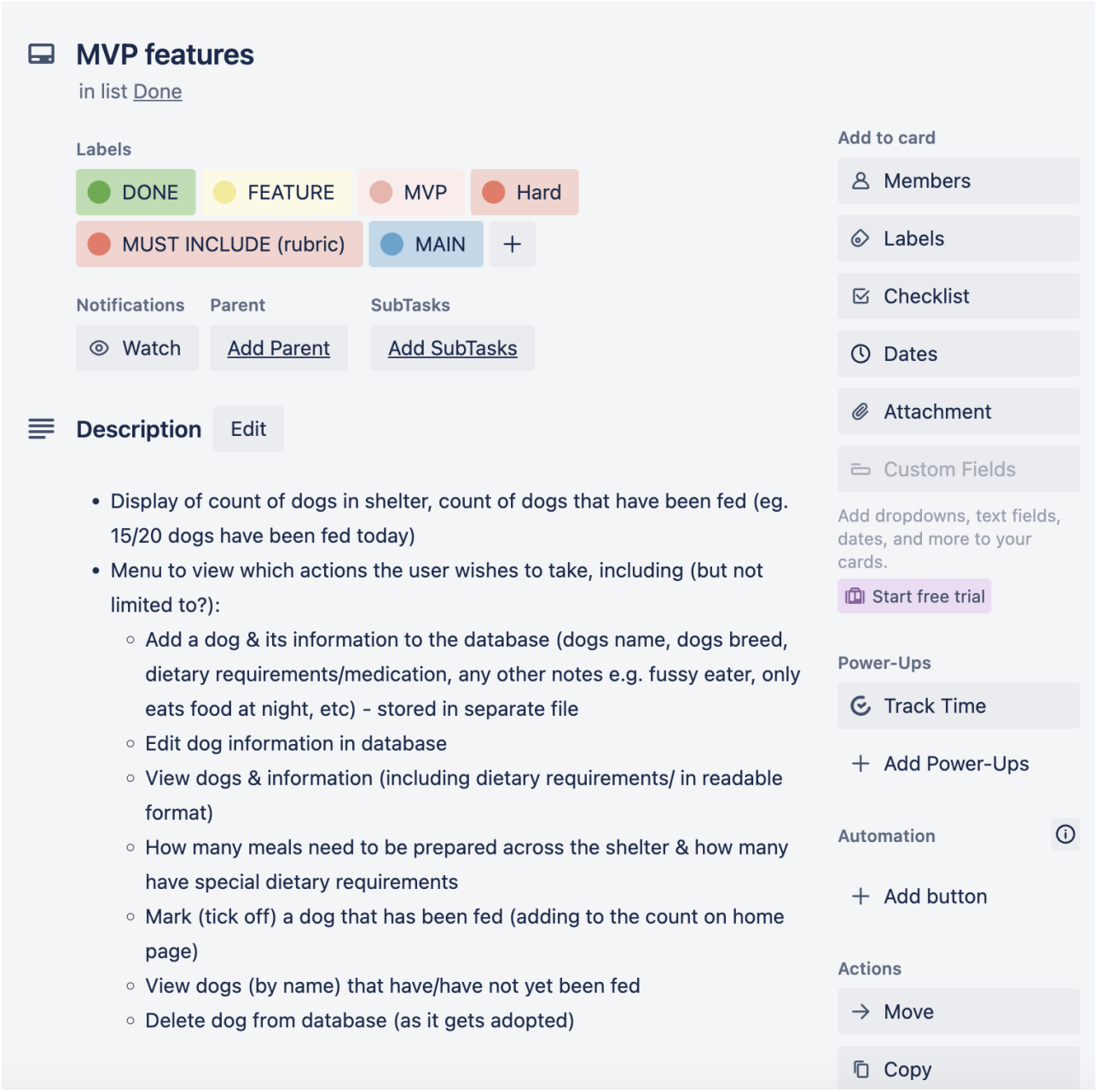
As I was learning things that were quite new to me, I found it hard to allocate set amounts of time to certain actions. For the purpose of measurement, I used ‘Easy’, ‘Medium’, ‘Hard’, and ‘Hard to gauge’ indicators on the tasks that I set out to do. If I used more than one of these indicators on a card, it would lie ‘somewhere in between’ the ones used.

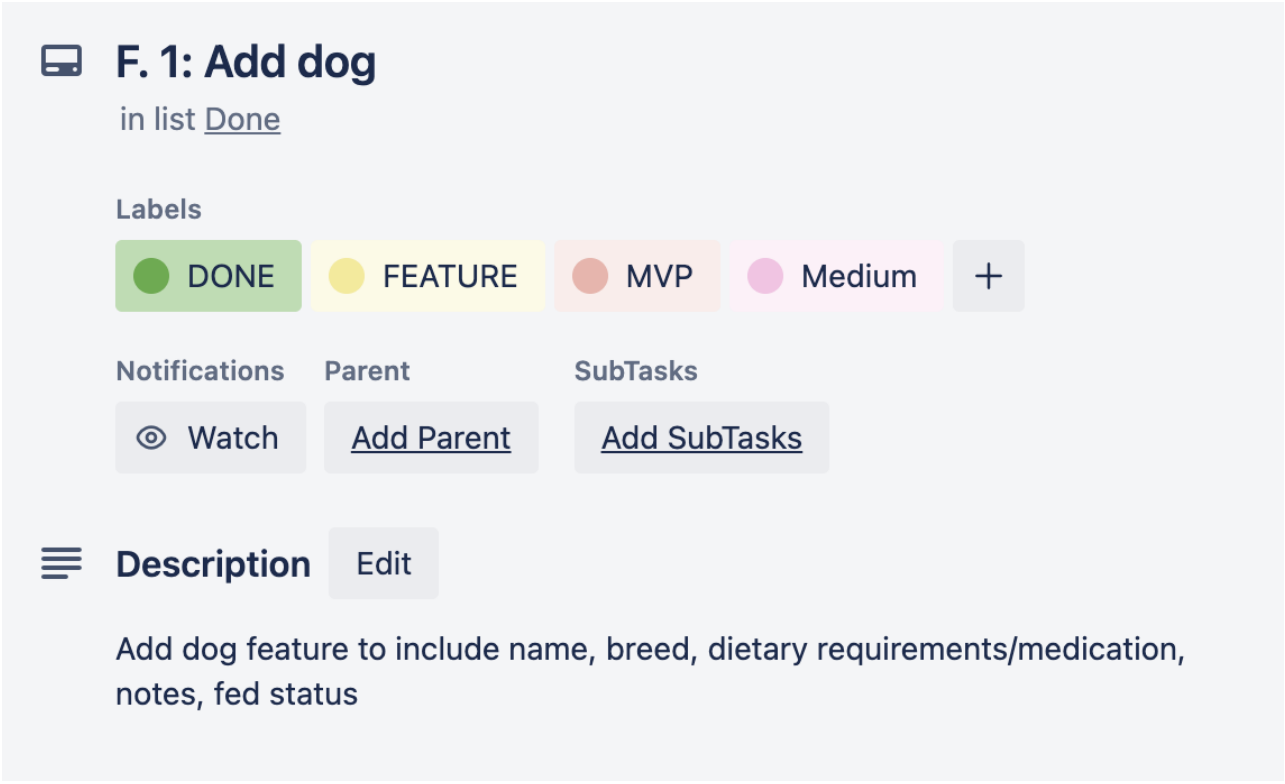
- ‘Easy’ would mean that I *assume* that the task should be easy and I could do a few ‘Easy’ tasks within one day.
- ‘Medium’ meant that it would likely take around a day to implement.
- ‘Hard’ meant several days or more or *assume* difficult concept to grasp.
- ‘Hard to gauge’ meant that I was not sure how long it would take to implement this feature or grasp this concept.

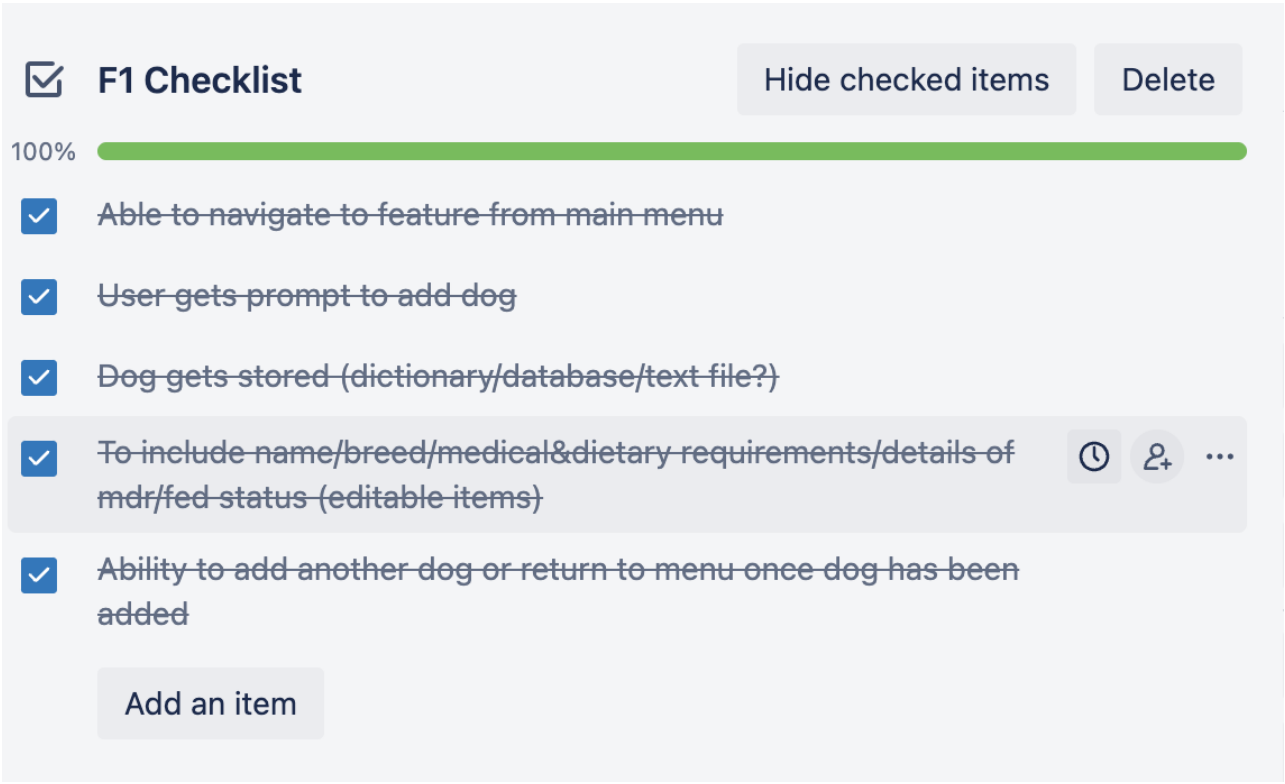
This measurement method ended up not working out as accurately as I had planned, as some tasks that were marked as medium or hard, such as ‘F. 6 - View dogs not fed’ which was comparatively pretty easy and done fairly quickly, whereas learning about python testing took me several days. This judgement will hopefully improve as I become more experienced as a coder.

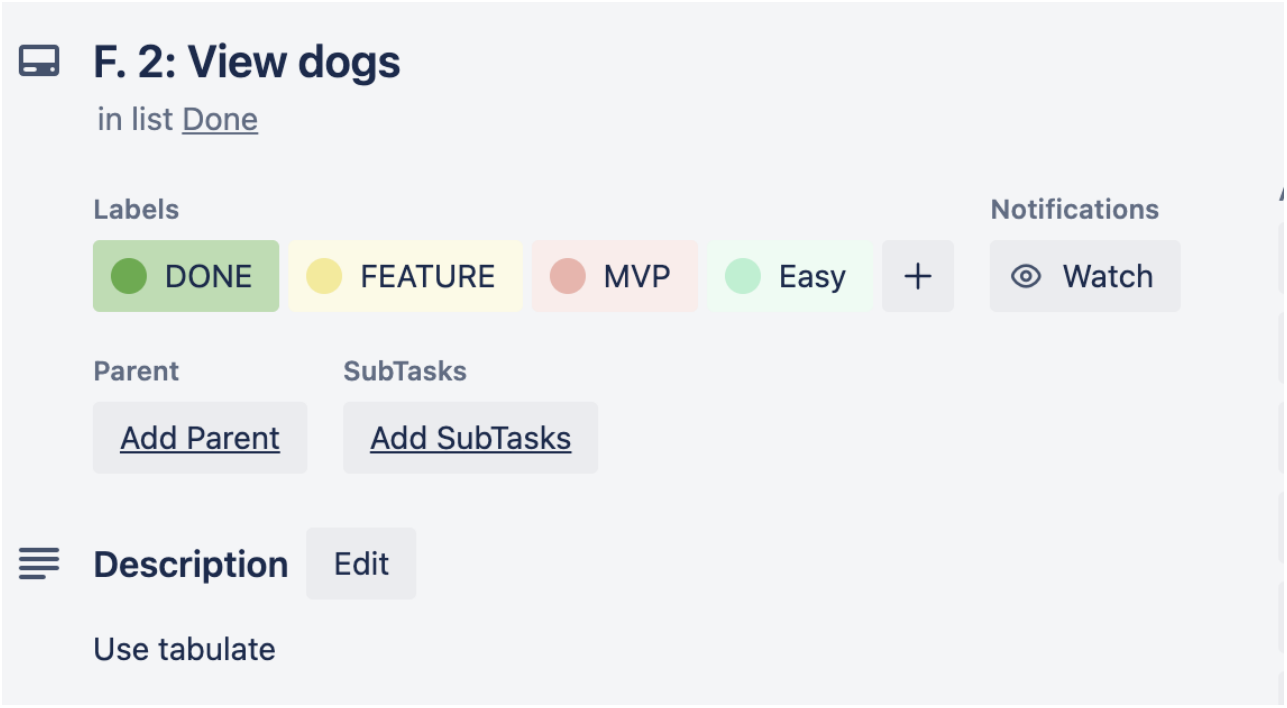
Additional labels for the tasks were also used:

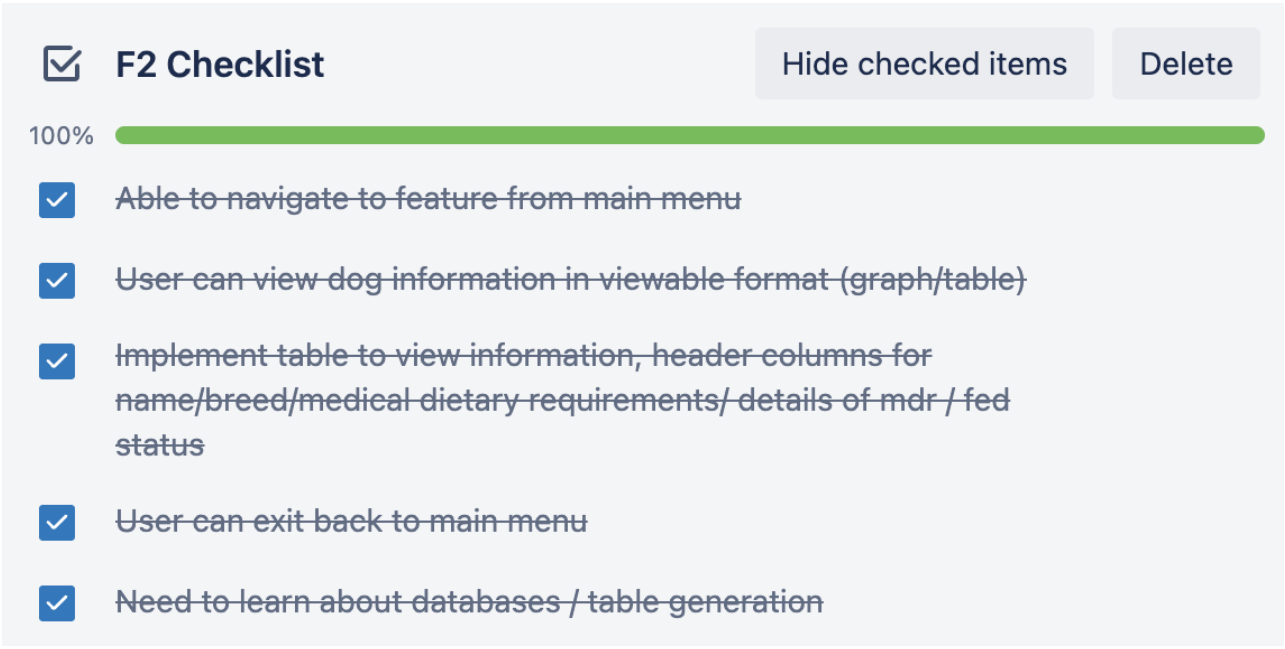
I used a Trello board (seen below) to help visualise how I approached these tasks to create the applications features, and to tick off the tasks once they had been done. The following are the main features' Trello cards, and their associated checklists.

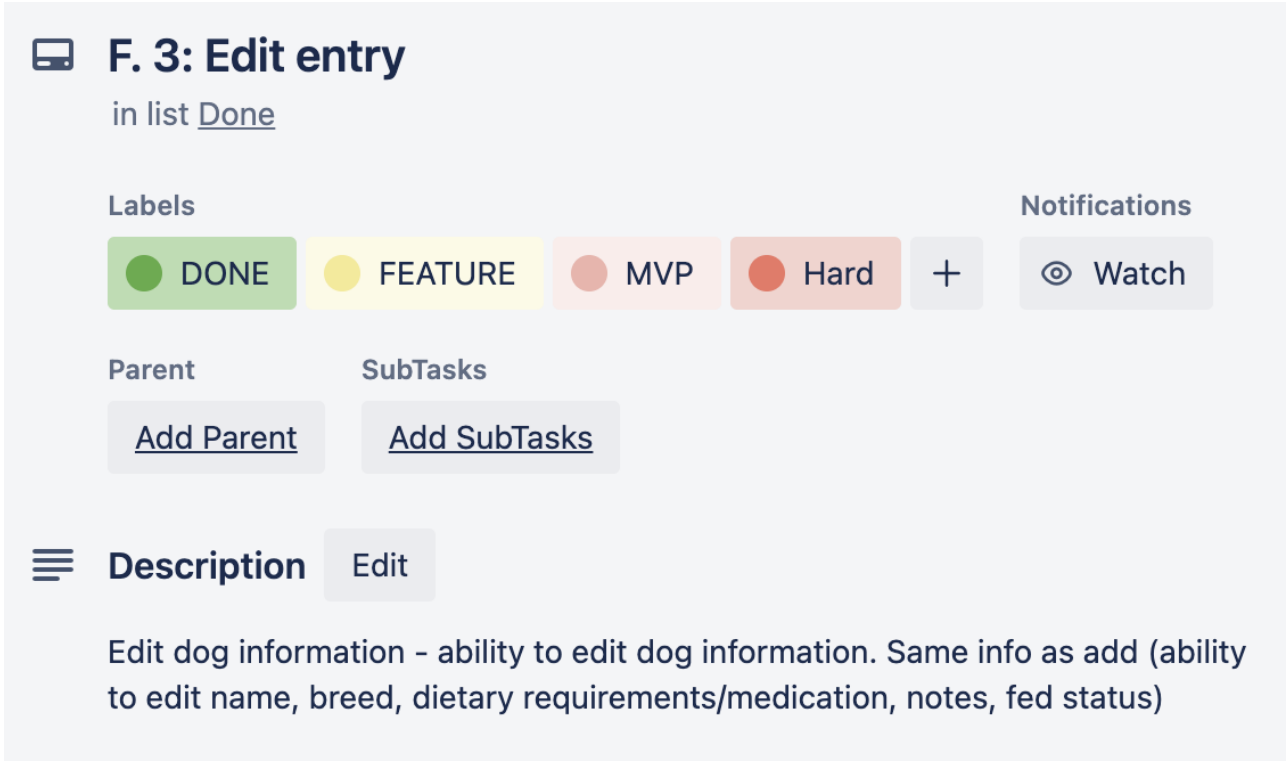

MVP features
in list Done
Labels
DONE
FEATURE
MVP
Hard
MUST INCLUDE (rubric)
MAIN
Notifications
Watch
Parent
Add Parent
SubTasks
Add SubTasks
Description
Edit
Display of count of dogs in shelter, count of dogs that have been fed (eg. 15/20 dogs have been fed today)
Menu to view which actions the user wishes to take, including (but not limited to?):
Add a dog & its information to the database (dogs name, dogs breed, dietary requirements/medication, any other notes e.g. fussy eater, only eats food at night, etc) - stored in separate file
Edit dog information in database
View dogs & information (including dietary requirements/ in readable format)
How many meals need to be prepared across the shelter & how many have special dietary requirements
Mark (tick off) a dog that has been fed (adding to the count on home page)
View dogs (by name) that have/have not yet been fed
Delete dog from database (as it gets adopted)
Add to card
Members
Labels
Checklist
Dates
Attachment
Custom Fields
Add dropdowns, text fields, dates, and more to your cards.
Start free trial
Power-Ups
Track Time
Add Power-Ups
Automation
Add button
Actions
Move
Copy

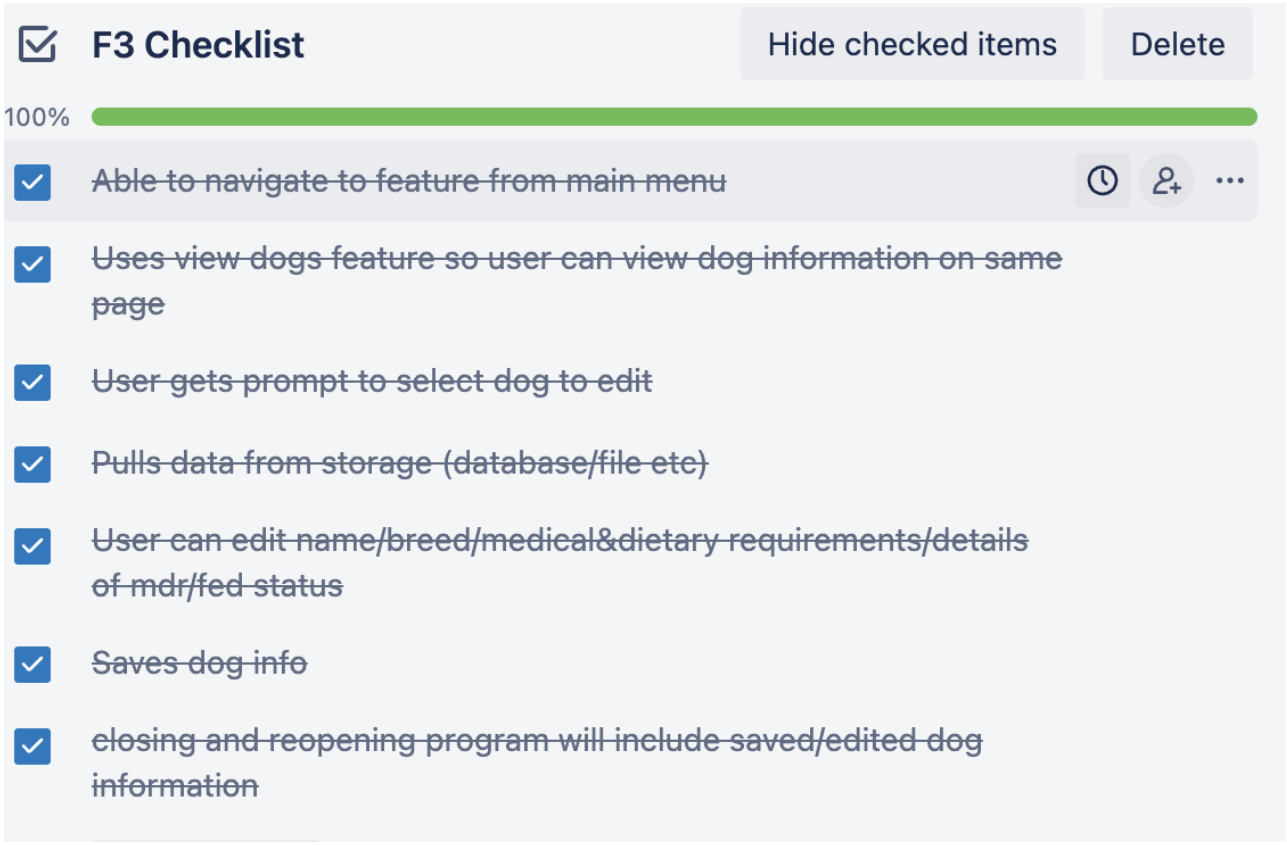

F. 1: Add dog
in list Done
Labels
DONE
FEATURE
MVP
Medium
Notifications
Watch
Parent
Add Parent
SubTasks
Add SubTasks
Description
Edit
Add dog feature to include name, breed, dietary requirements/medication, notes, fed status

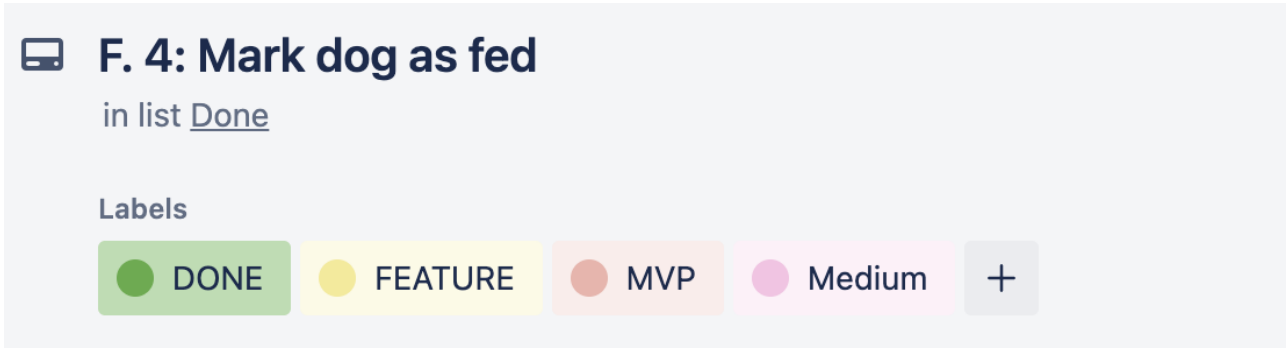

F1 Checklist
Hide checked items
Delete
100%
Able to navigate to feature from main menu
User gets prompt to add dog
Dog gets stored (dictionary/database/text file?)
To include name/breed/medical&dietary requirements/details of mdr/fed status (editable items)
Ability to add another dog or return to menu once dog has been added
Add an item

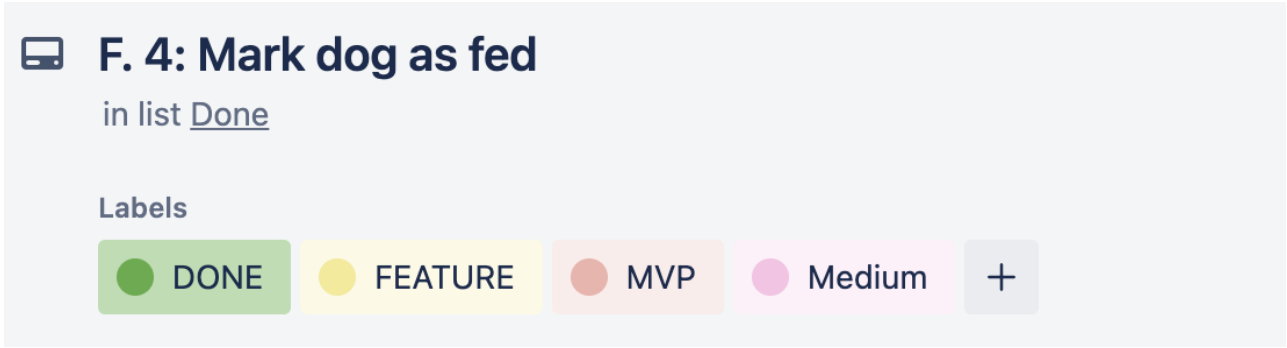

F. 2: View dogs
in list Done
Labels
DONE
FEATURE
MVP
Easy
Notifications
Watch
Parent
Add Parent
SubTasks
Add SubTasks
Description
Edit
Use tabulate

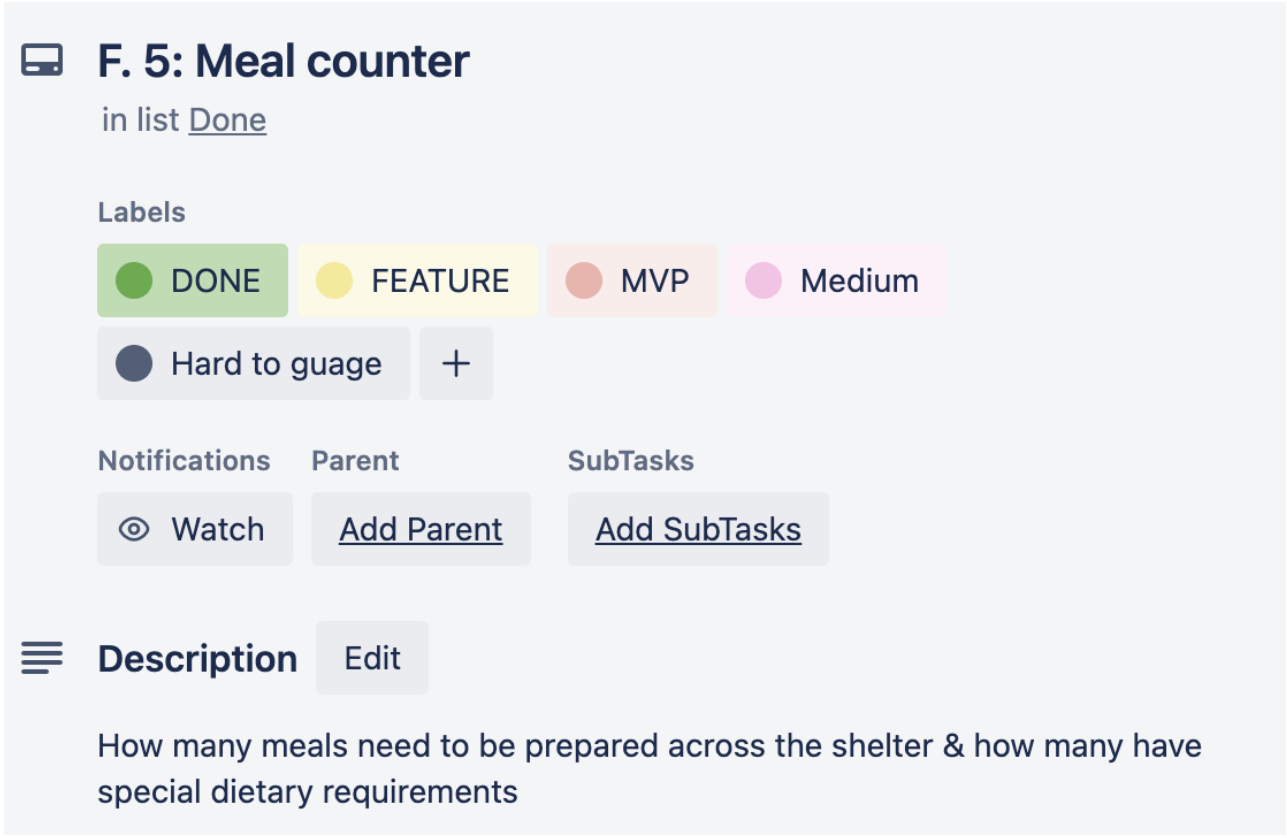

F2 Checklist
Hide checked items
Delete
100%
Able to navigate to feature from main menu
User can view dog information in viewable format (graph/table)
Implement table to view information, header columns for name/breed/medical dietary requirements/ details of mdr / fed status
User can exit back to main menu
Need to learn about databases / table generation

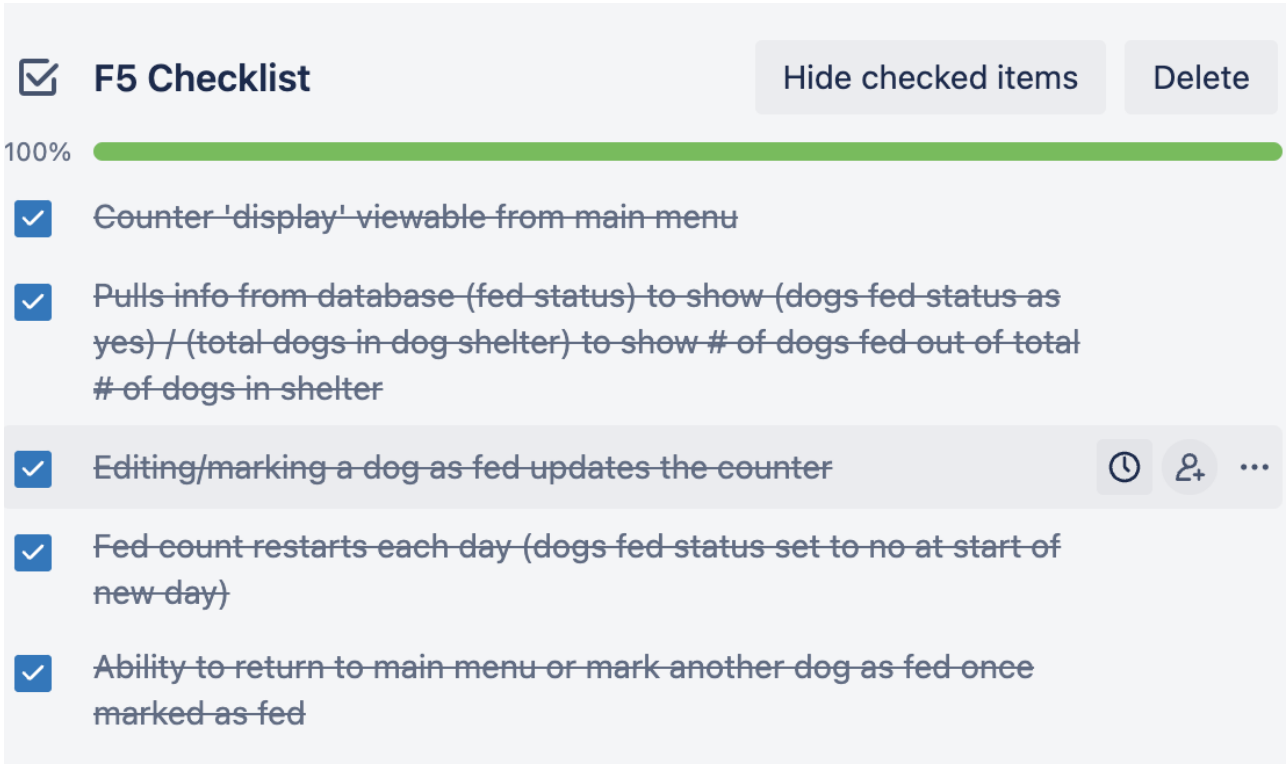

F. 3: Edit entry
in list Done
Labels
DONE
FEATURE
MVP
Hard
Notifications
Watch
Parent
Add Parent
SubTasks
Add SubTasks
Description
Edit
Edit dog information - ability to edit dog information. Same info as add (ability to edit name, breed, dietary requirements/medication, notes, fed status)

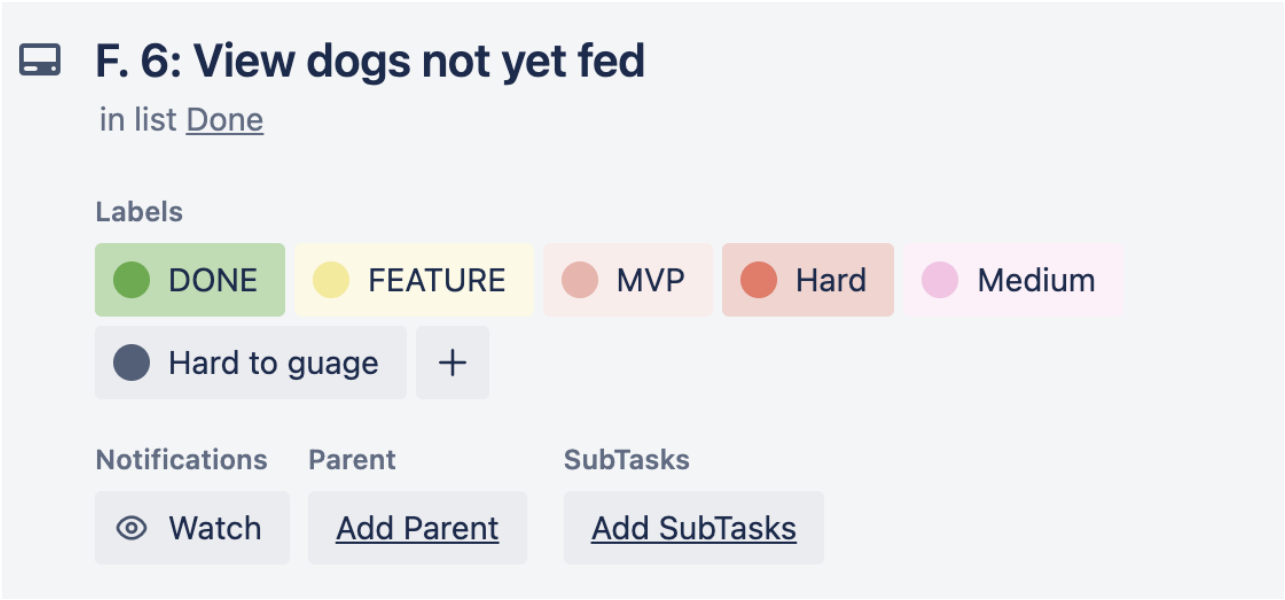

F3 Checklist
Hide checked items
Delete
100%
Able to navigate to feature from main menu
Uses view dogs feature so user can view dog information on same page
User gets prompt to select dog to edit
Pulls data from storage (database/file etc)
User can edit name/breed/medical&dietary requirements/details of mdr/fed status
Saves dog info
closing and reopening program will include saved/edited dog information

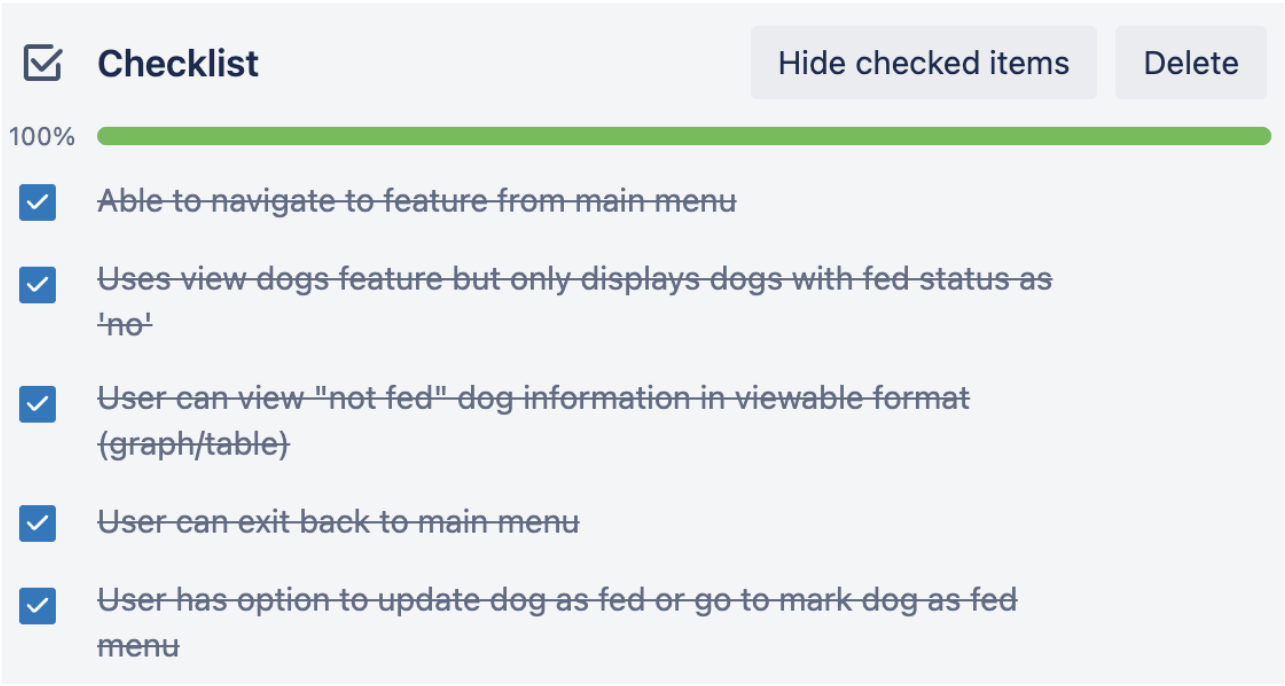

F. 4: Mark dog as fed
in list Done
Labels
DONE
FEATURE
MVP
Medium

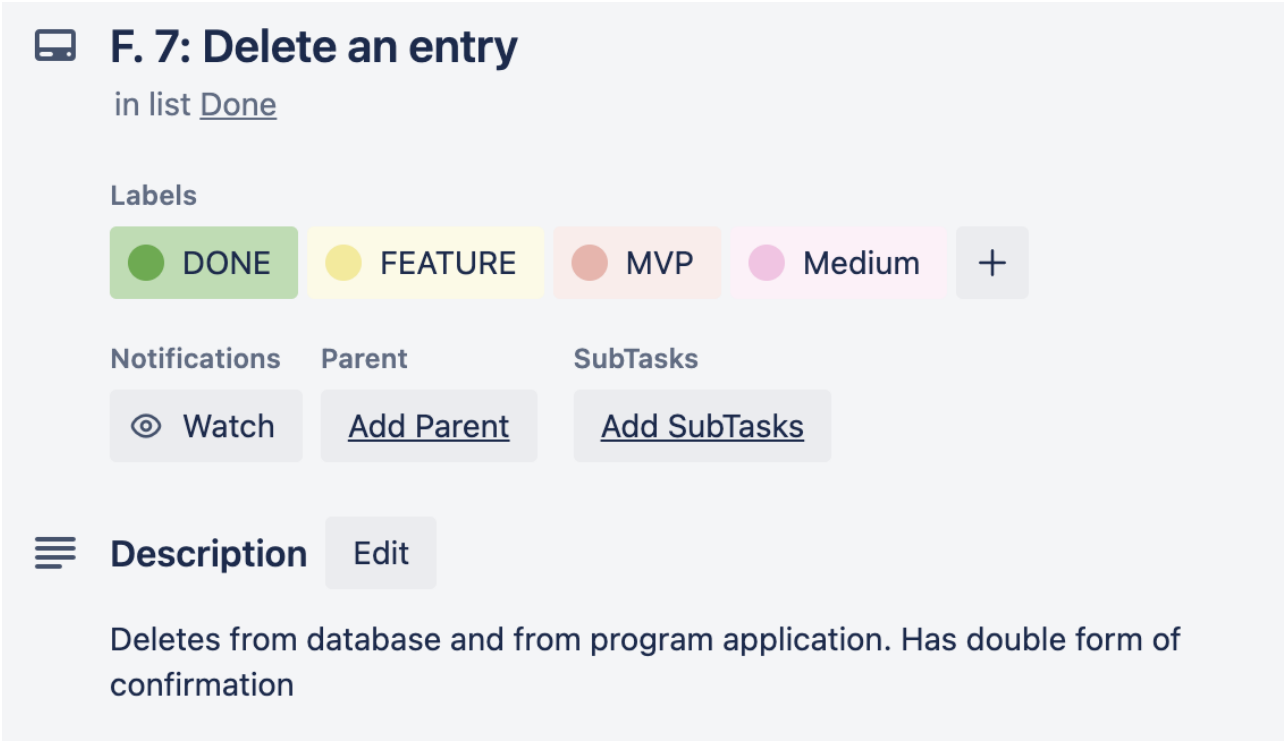

F. 4: Mark dog as fed
in list Done
Labels
DONE
FEATURE
MVP
Medium

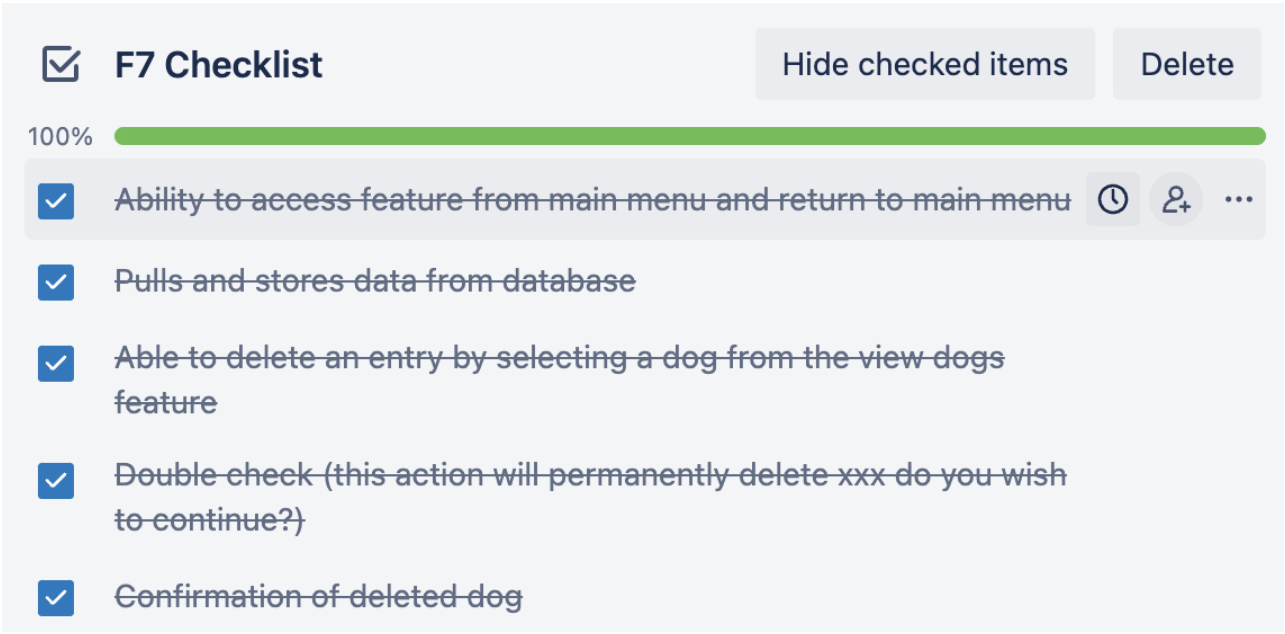

F. 5: Meal counter
in list Done
Labels
DONE
FEATURE
MVP
Medium
Hard to guage
Notifications
Parent
SubTasks
Watch
Add Parent
Add SubTasks
Description
Edit
How many meals need to be prepared across the shelter & how many have special dietary requirements

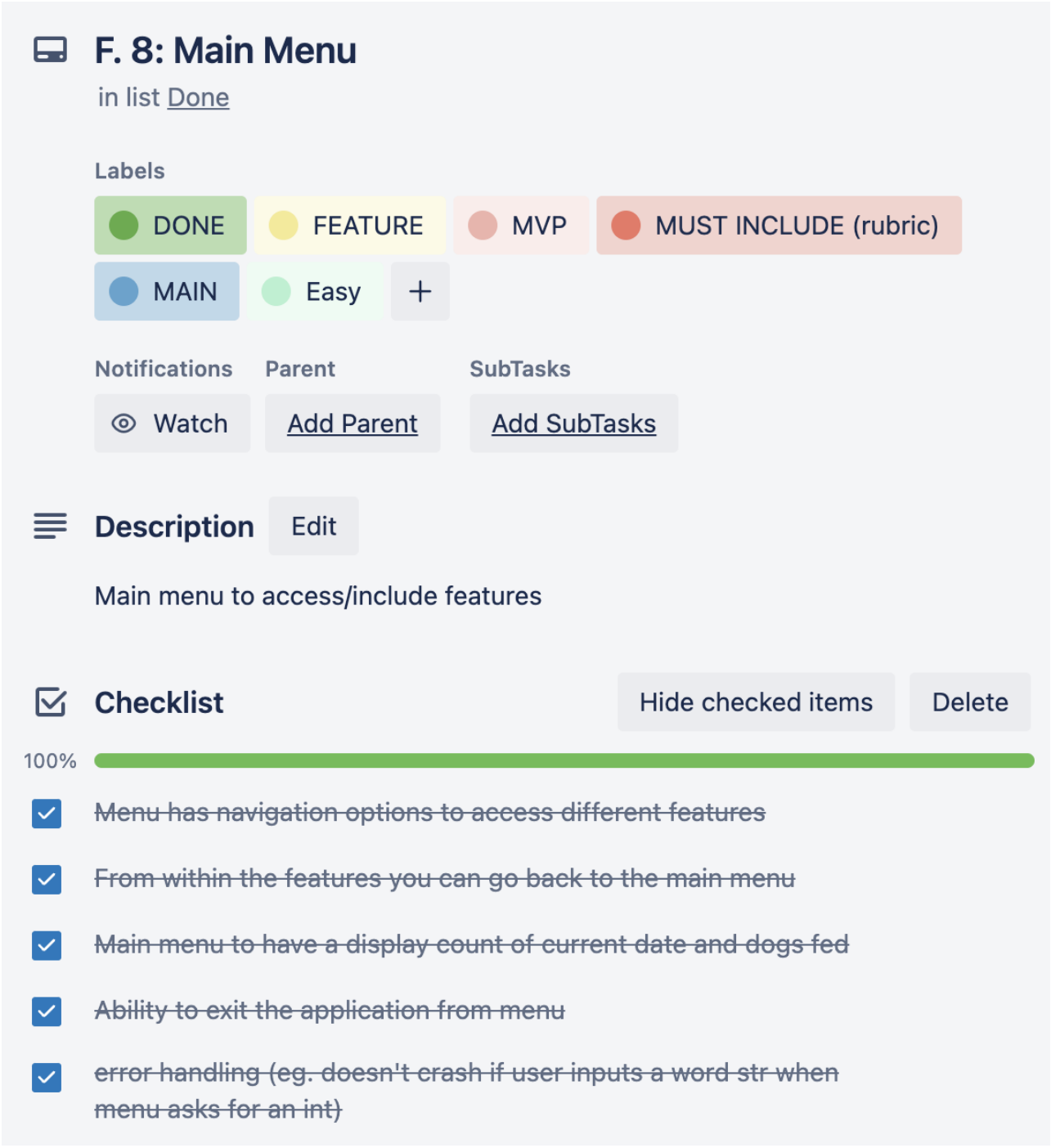

F5 Checklist
Hide checked items
Delete
100%
Counter 'display' viewable from main menu
Pulls info from database (fed status) to show (dogs fed status as yes) / (total dogs in dog shelter) to show # of dogs fed out of total # of dogs in shelter
Editing/marking a dog as fed updates the counter
Fed count restarts each day (dogs fed status set to no at start of new day)
Ability to return to main menu or mark another dog as fed once marked as fed

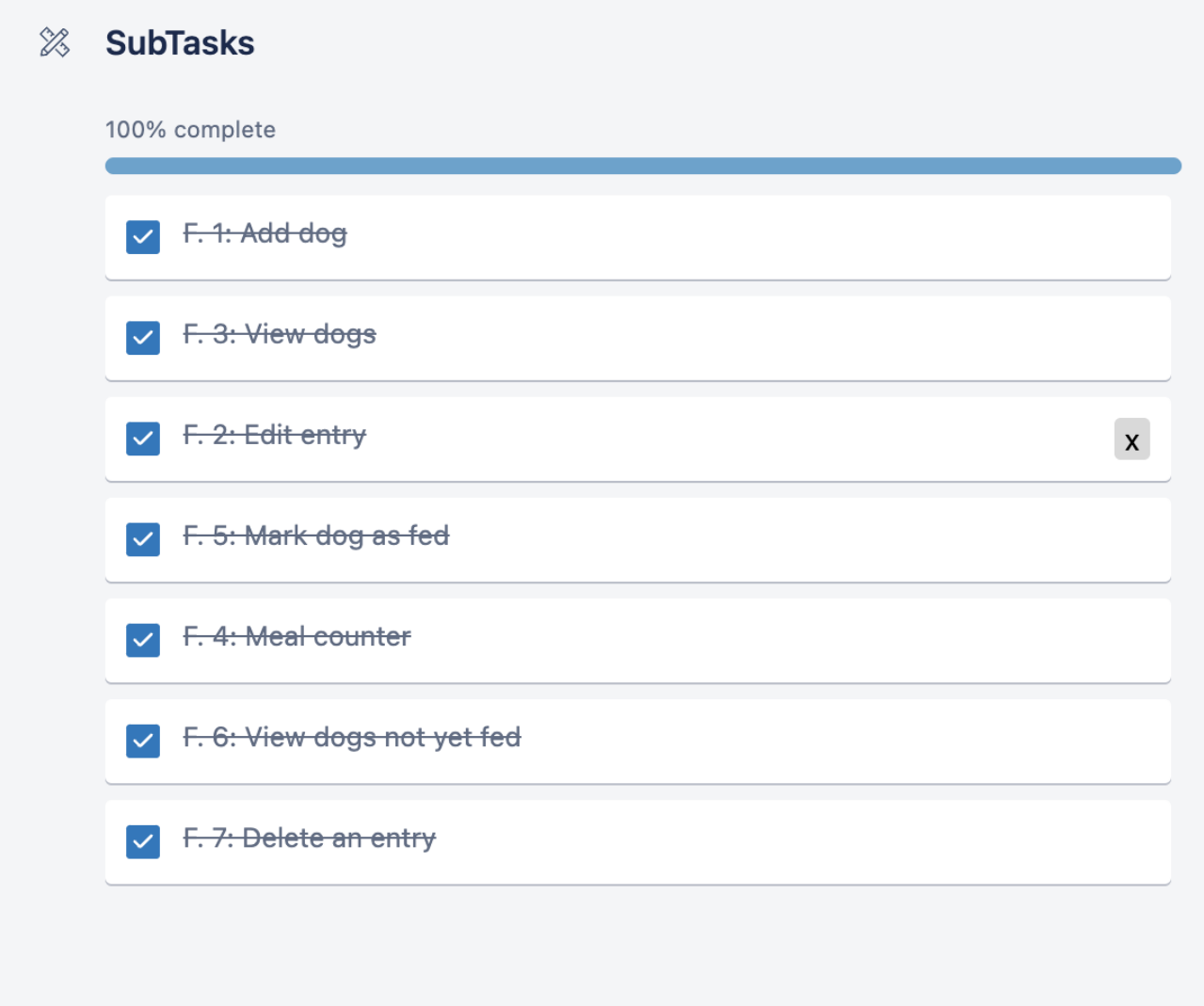

F. 6: View dogs not yet fed
in list Done
Labels
DONE
FEATURE
MVP
Hard
Medium
Hard to guage
Notifications
Parent
SubTasks
Watch
Add Parent
Add SubTasks


Checklist
Hide checked items
Delete
100%
Able to navigate to feature from main menu
Uses view dogs feature but only displays dogs with fed status as 'no'
User can view "not fed" dog information in viewable format (graph/table)
User can exit back to main menu
User has option to update dog as fed or go to mark dog as fed menu


F. 7: Delete an entry
in list Done
Labels
DONE
FEATURE
MVP
Medium
Notifications
Parent
SubTasks
Watch
Add Parent
Add SubTasks
Description
Edit
Deletes from database and from program application. Has double form of confirmation


F7 Checklist
Hide checked items
Delete
100%
Ability to access feature from main menu and return to main menu
Pulls and stores data from database
Able to delete an entry by selecting a dog from the view dogs feature
Double check (this action will permanently delete xxx do you wish to continue?)
Confirmation of deleted dog

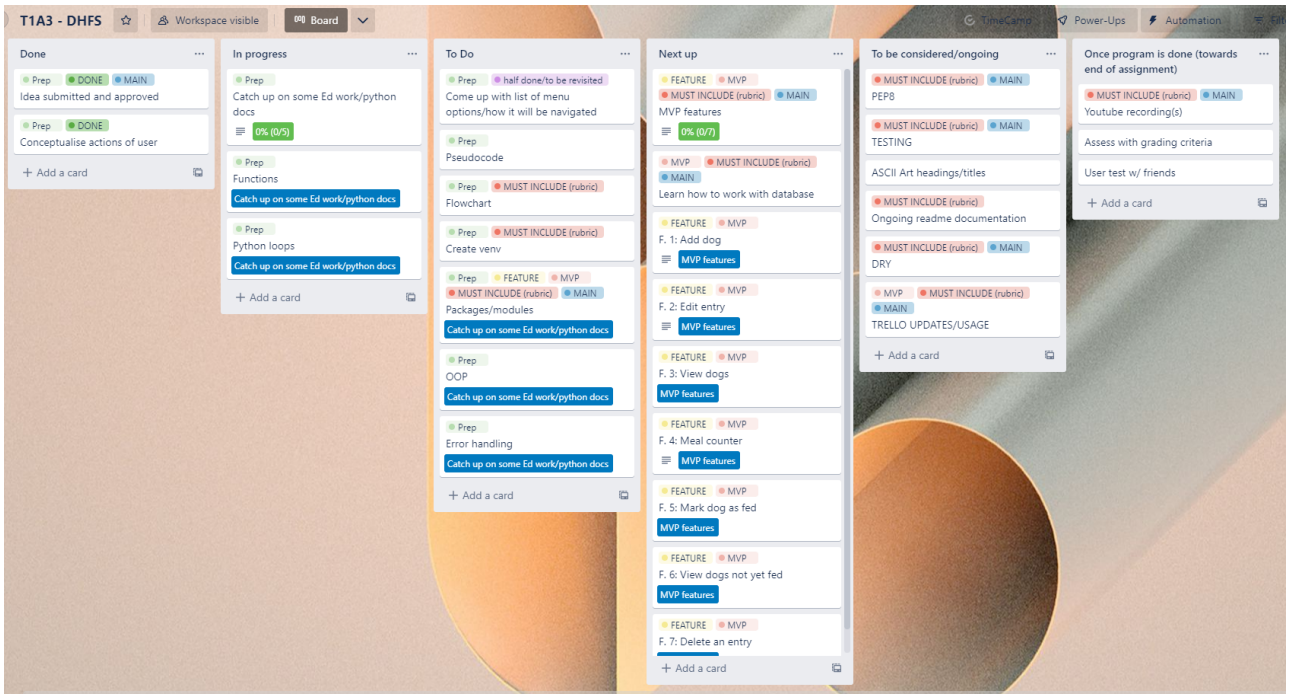

F. 8: Main Menu
in list Done
Labels
DONE
FEATURE
MVP
MUST INCLUDE (rubric)
MAIN
Easy
Notifications
Parent
SubTasks
Watch
Add Parent
Add SubTasks
Description
Edit
Main menu to access/include features
Checklist
Hide checked items
Delete
100%
Menu has navigation options to access different features
From within the features you can go back to the main menu
Main menu to have a display count of current date and dogs fed
Ability to exit the application from menu
error handling (eg. doesn't crash if user inputs a word str when menu asks for an int)



Trello usage

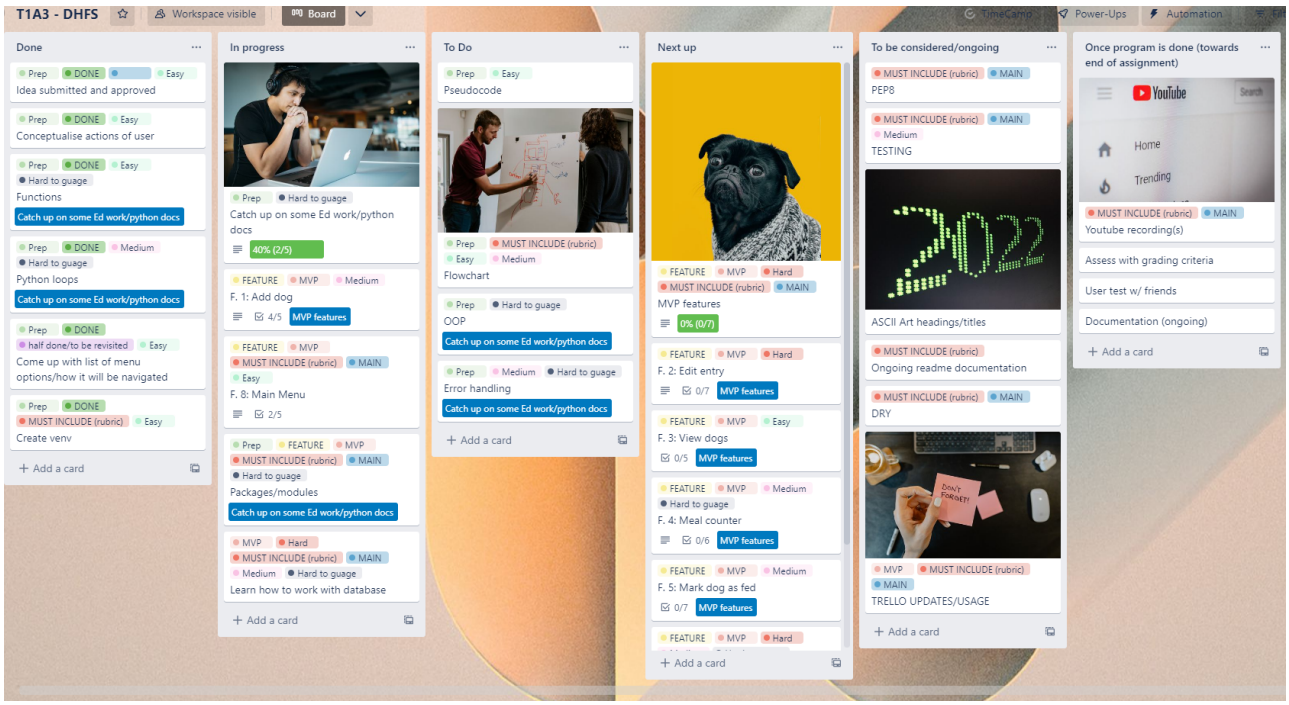
DAY2 Tuesday 6/12

- Most of the time from Monday 5/12 - Thursday 8/12 was spent on catching up on learning more about python, watching old classes, catching up on Ed coursework, and reading documentation and watching youtube tutorials to familiarise myself with how python operates, to a point where I felt comfortable being able to start building the application.



Day 5 Friday 9/12 - *jumped into build*

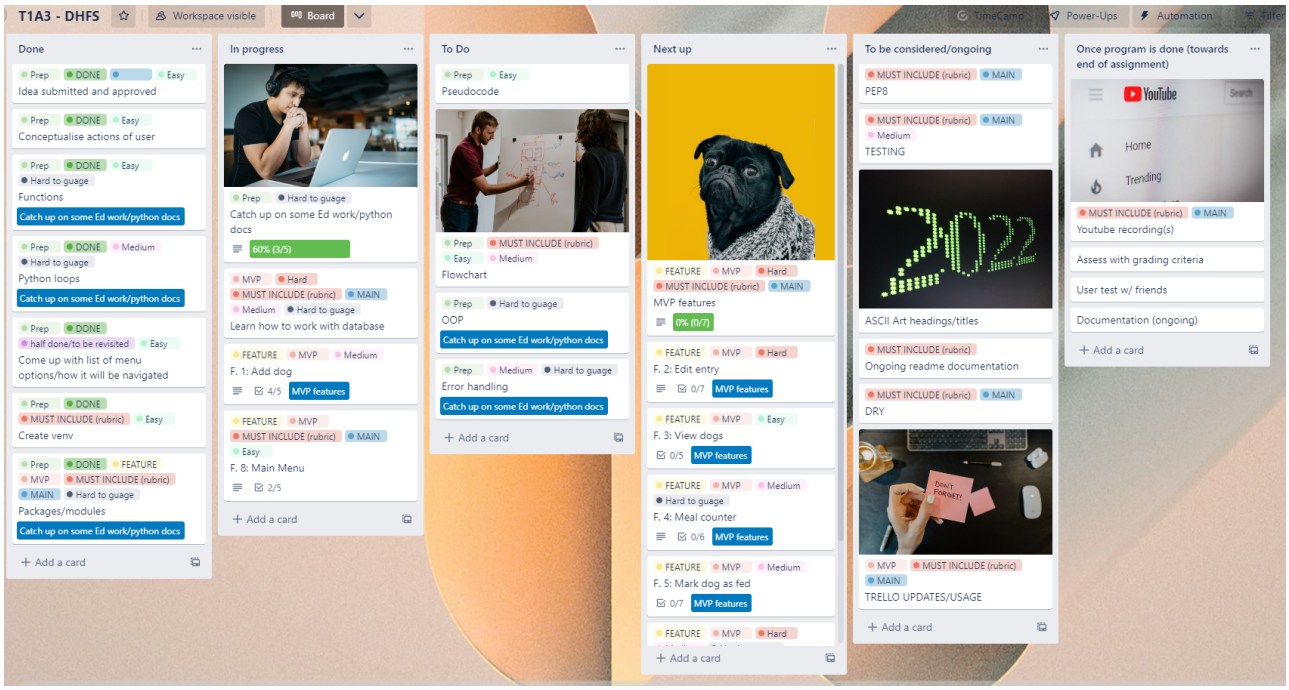
- Started the build after being inspired by watching this very basic video on how to create a menu and allow the user to input a dogs information. No database was implemented yet but I was able to create the main menu, a key feature from which all others would be accessed.



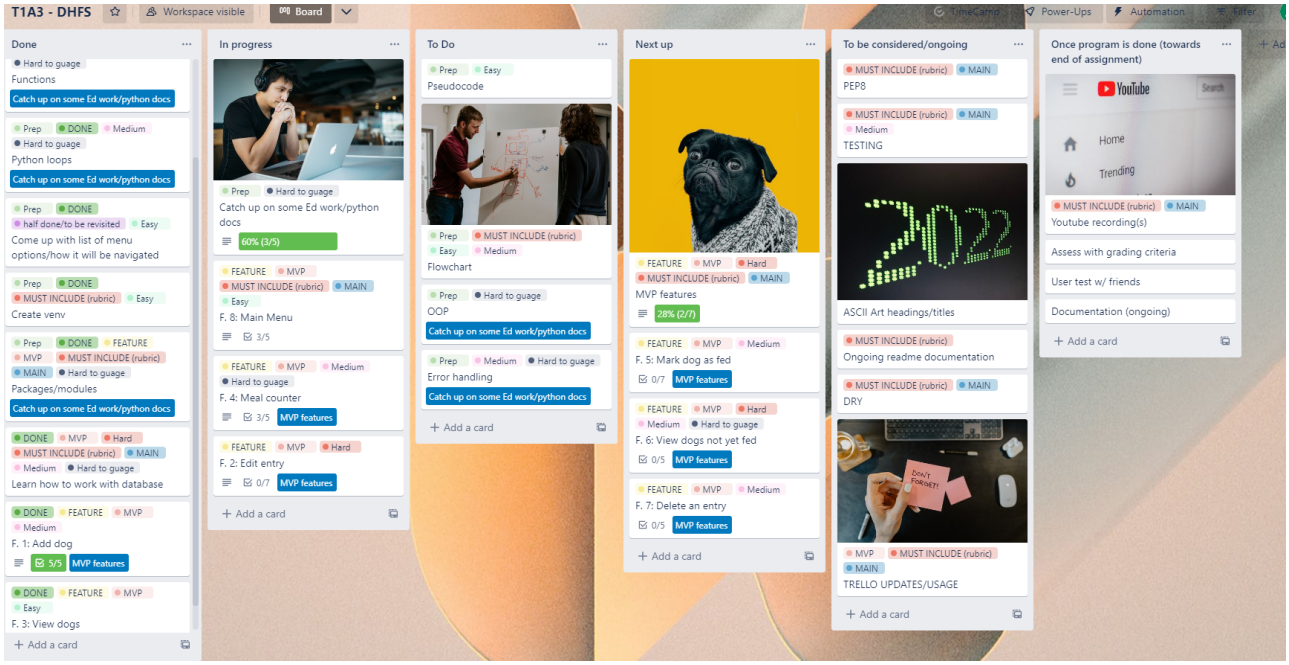
- *Was away Saturday 10/12 morning until Sunday 11/12 evening and did not work on the project these dates.*

Day 7 Sunday 11/12

- Studied modules/packages and imports and started implementing them into the program. Found the `tinydb` module and started learning about how to use it with a separate file to store data across terminal sessions.

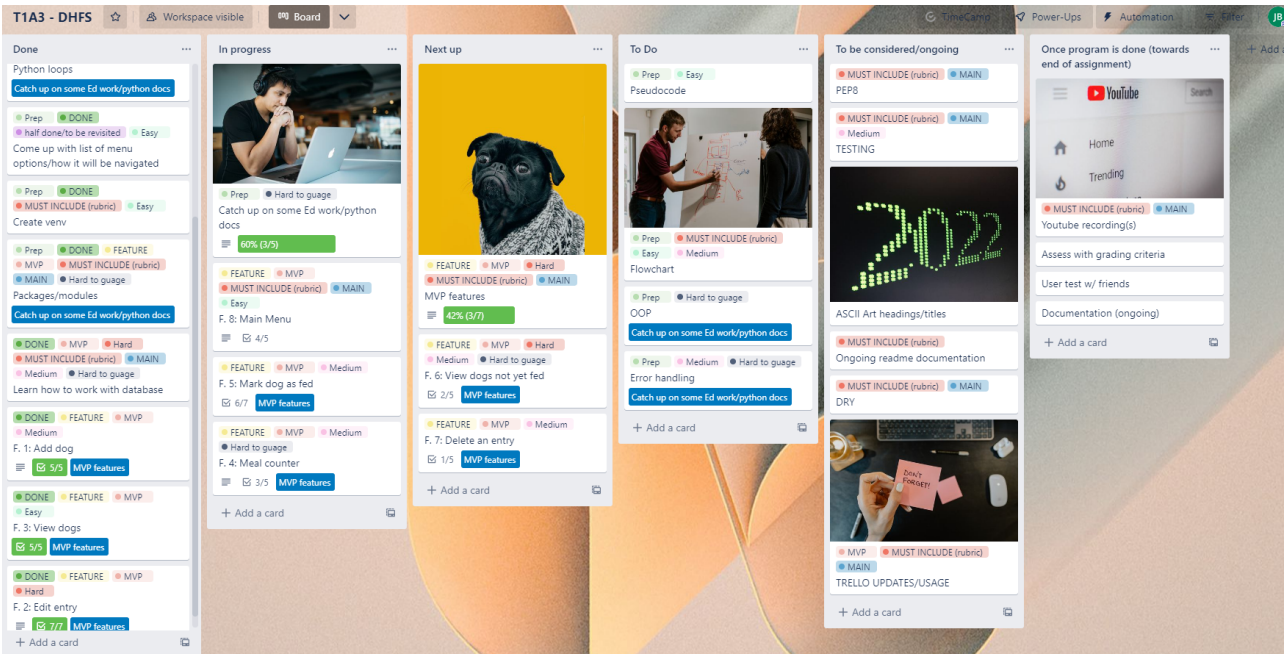


Day 8 Monday 12/12



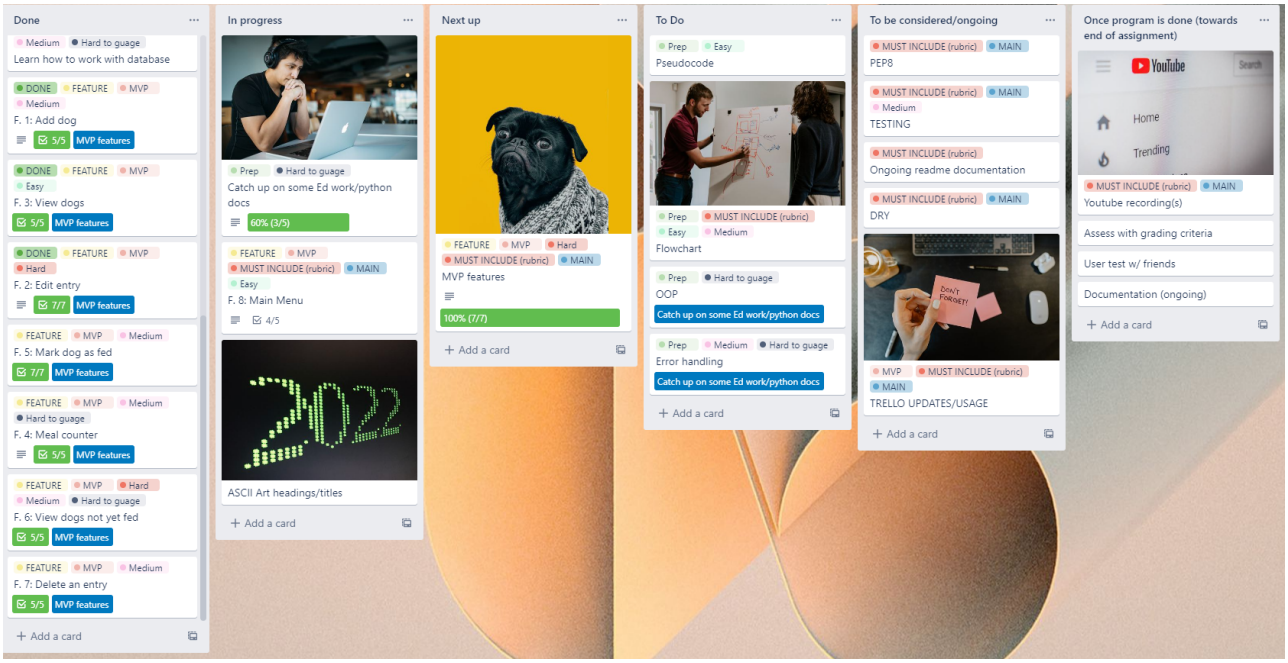
Day 9 Tuesday 13/12

- Had most of the application done by this point.



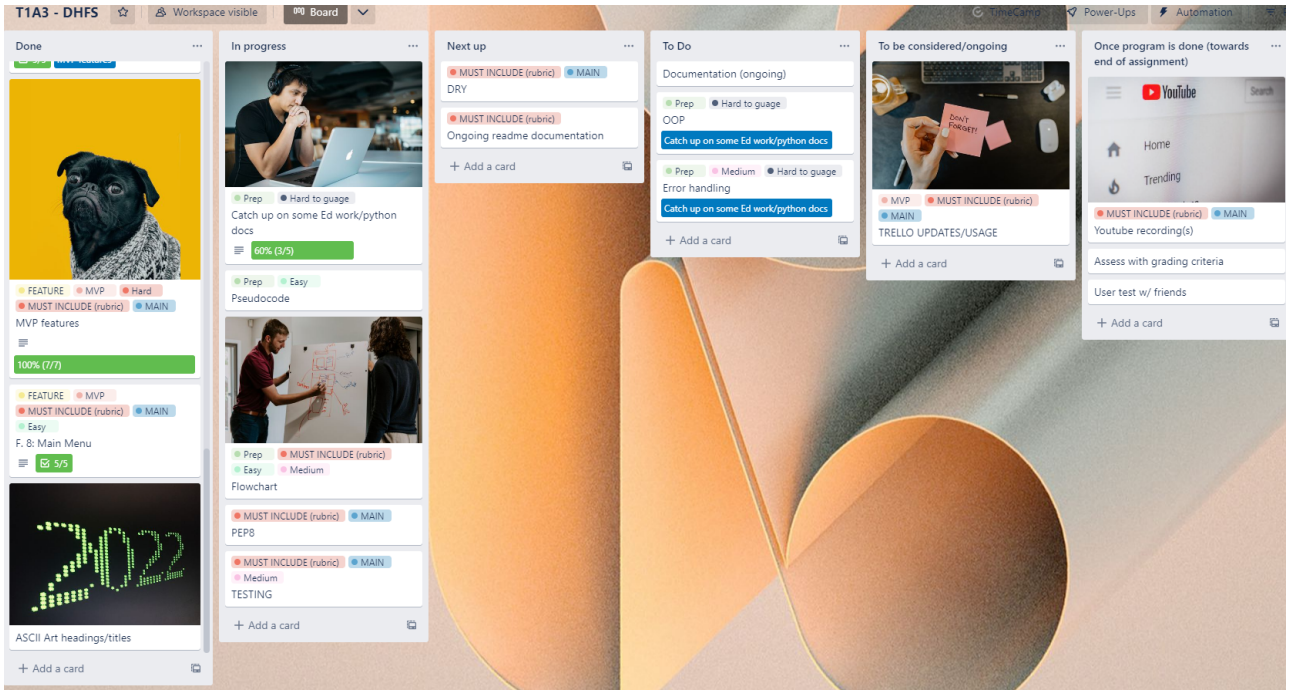
Day 10 Wednesday 14/12

- Application was pretty much done by this stage as it was functional and most errors appeared to be accounted for.



Day 11 Thursday 15/12 onwards

- Application was at a working and functional level. From this point onwards I made some minor changes to the code to fix some wording and display issues whilst also considering documentation for the project. Code was styled to PEP8 standards. A lot of time has been spent on trying to figure out testing with `pytest`.



Help documentation

- Describe how to use and install the application
  - steps to install the application
  - any dependencies required by the application to operate
  - any system/hardware requirements
  - how to use any command line arguments made for the application

SLIDE DECK

Slides must include:

1. An overview of terminal application
   - Must explain: Main features and overall structure
2. An overview of code:
   - Must explain: Explanation of important parts of code, including any crucial application logic