

Calorie Counter App

Sooyeon Jeung

Calorie Counter App

- *A terminal app that counts and tracks the user's calories consumed.*

Problem

For anyone interested in weight management, monitoring & tracking calories become -

- emotionally stressful to be on top of personal recording at all times.
- tedious to manually calculate and create a readable data view.
- difficult to know how you've done it over a period.

Solution

- Navigate menu with clear directions to track and view calorie intake.
- Provide guideline comments daily/weekly.
- Give confidence in weight management.

Audience

Health enthusiast, Busy individuals interested in weight management.

Features

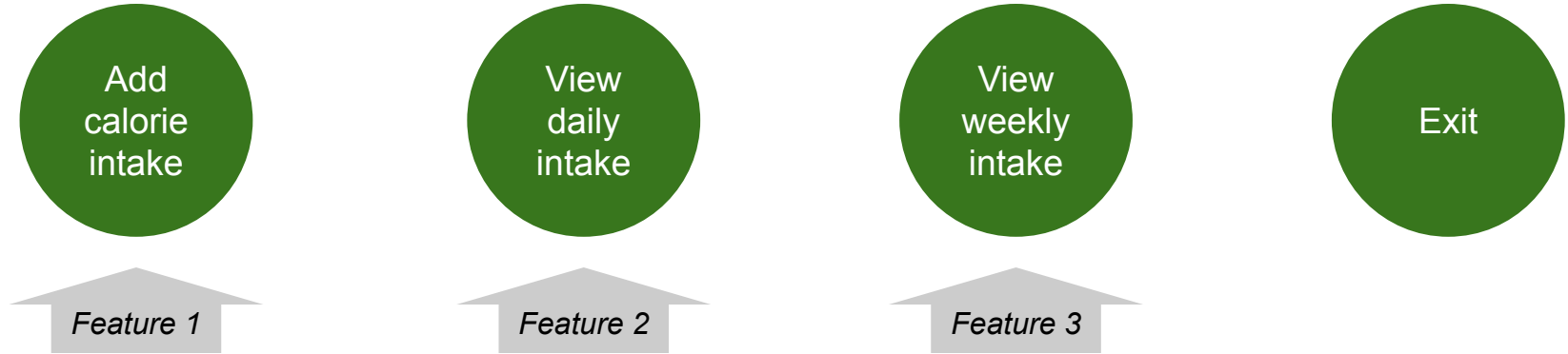
As a user, I want to be able to

1. Record my calorie intake.
2. Calculate and view my calorie intake daily and receive an appropriate health recommendation.
3. Calculate and view my calorie intake weekly and receive an appropriate health recommendation.

Demo

Menu Options

- *Easy to navigate, Clear instructions, Reliable*



Menu Options

Add
calorie
intake

User Inputs

- Date (Sat - Sun)
- Food Item (CSV file)
- Portion (0.5/1/1.5/2)

Functions

- Read/Write to CSV files
- Confirms the inputs
- Store user data (date, total calories)
- Error handling
- Add another data

View
daily
intake

User Input

- Date (Sat - Sun)

Functions

- Calculate daily total calories
- Print brief guideline messages
- Error handling
- Add a new data intake

View
weekly
intake

Functions

- Calculate weekly total calories per number of calorie entry
- Print brief guideline messages
- Error handling
- Add a new intake

Exit

Functions

- Error handling
- Exit

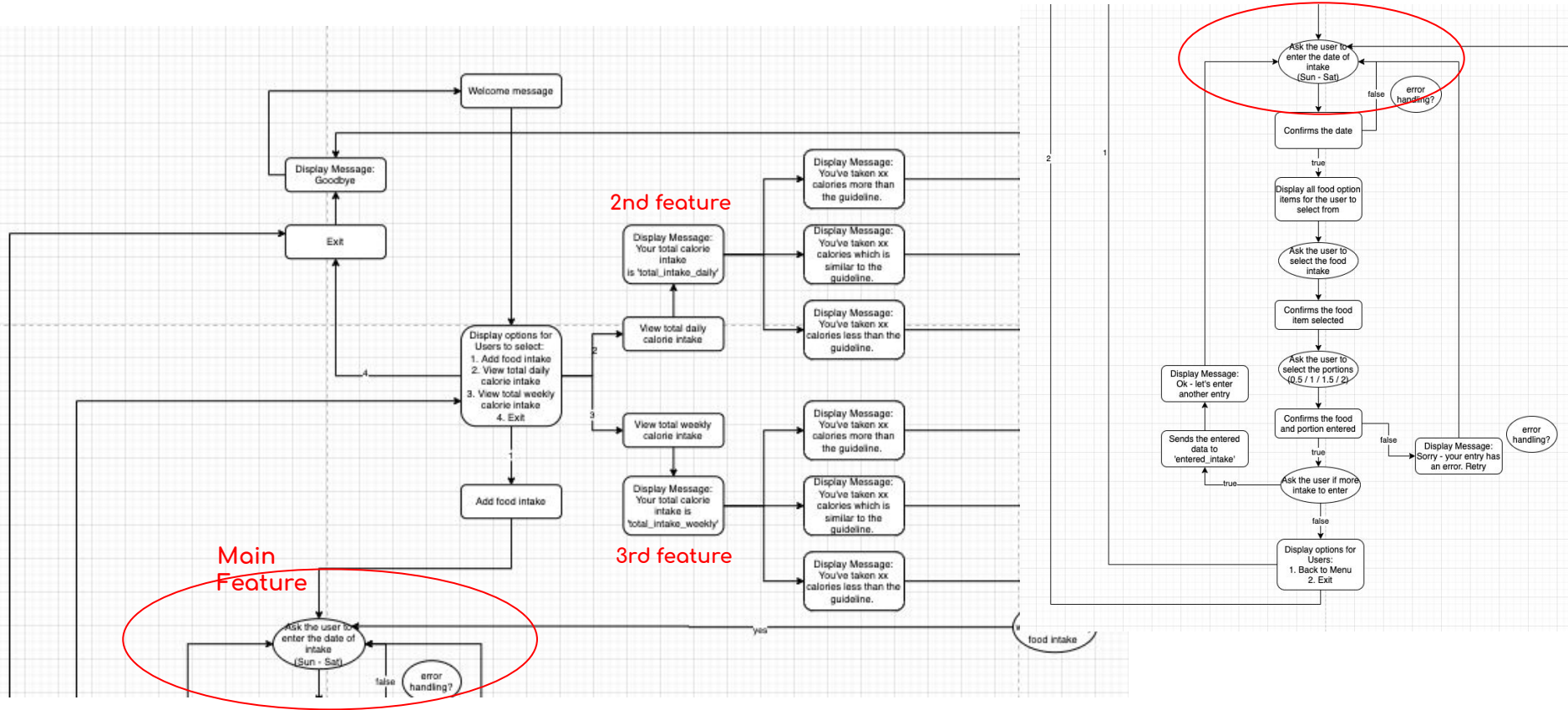
Development Process - Trello board

The screenshot shows a Trello board for a project named 'CalorieTracker'. The board is organized into six columns, each representing a different stage or category of the development process. Each column contains several cards with detailed task descriptions, progress indicators, and due dates.

- ToDo - MVP**: Contains cards for creating a fixed database of calories, getting dates from the user, displaying food items, displaying portion options, and menu options for adding calories, viewing daily/weekly intake, and exiting. It also includes a card for printing a brief message at the end of menu options.
- Questions**: Contains cards asking about streamlined user entry, suitable data files, and MVP version requirements.
- Good to Haves**: Contains cards for adding new food items, personalized recommendations, counting calories, setting personal goals, and storing user data.
- Features (extends from User Story) - CONFIRMED**: Contains four cards detailing specific features for calorie intake tracking and feedback.
- Key logics**: Contains cards for menu options, class instantiation, while loops, error handling, if/else statements, and module nesting.
- Gem ideas**: Contains cards for bundler, TTY-Prompt, Artii, pry, and Colorize.

Each card includes a progress bar, a due date (e.g., Dec 12, Dec 16, Dec 17, Dec 18), and a 'Add another card' button at the bottom.

Development Process - Flow Chart



Key logics of the application

- class AddCalorie
 - initialize objects, create instance variables, added attr_accessor/reader
 - Interacting objects across methods
- case, when for the main menu options
 - while loop to re-run 'menu 1' when needed
 - active use of instance variables across the code
 - boolean (re-direct, print calorie analysis)
 - iterators for hash and array
- methods
 - Art
 - welcome
 - Initialize
 - user_select
 - find_food_item
 - handle_input

Key logics of the application

- CSV file

- Read files (CSV.foreach, CSV.read)
- Write files: CSV.open("file_name", "a +")

- Error Handling

- begin/rescue
- when/else
- boolean (if/elsif/else)
- data structure (e.g. .to_i/.to_f, value = 0.0, @date = "")
- TTY-Prompt
- pry
- clear instructions

```
@food_calorie_table = CSV.read("food_calorie_file.csv", headers: true, header_converters: :symbol)
```

```
daily_total_calorie = 0.0
CSV.foreach("user_data.csv", options).with_index do |(d,t), i|
  if d == @date
    t = t.to_i
    daily_total_calorie += t
  end
end
```

```
CSV.open("user_data.csv", "a+") do |csv|
  csv << [@date, total_calories]
end
```

Key logics of the application - main menu

```
class AddCalories
  attr_accessor :date, :food_item, :portion
  attr_reader :calorie
  def initialize
    @date = ''
    @food_item = ''
    @portion = 0.0
    @calorie = 0
    @prompt = TTY::Prompt.new
    @food_calorie_table = CSV.read("food_calorie_file.csv", headers: true, header_converters: :symbol)
    ...
  end
end
```

Key logics of the application - redirect

```
when 1
  while_add_intake_start = true
  while while_add_intake_start == true
  ...
    puts "Do you have more to add? (Y/N)"
    answer = gets.chomp
  ...
    if answer == 'Y'
      while_add_intake_start = true
    else
      while_add_intake_start = false
      puts "Goodbye"
      exit(0)
    end
  end
end
end
```

```
when 3
  ...
  puts "Do you have any new food intake to add? (Y/N)"
  redirect_to_menu_one = gets.chomp
  if redirect_to_menu_one == 'Y'
    puts "Enter 1 to add new calorie intake or enter return to exit"
    AddCalories.new().handle_input
  else
    puts "Goodbye"
    exit(0)
  end
end
```

Key logics of the application - working w/ csv files

```
def user_select
  options = {:encoding => 'UTF-8', :skip_blanks => true}
  CSV.foreach("food_calorie_file.csv", options).with_index do |(f,c), i|
    if f == @food_item
      puts "You've eaten #{@food_item} which is #{c}.".colorize(:light_blue)
      @calorie = c
      user_taken_index = i
    end
  end
end
```

To Read / Write on CSV files

```
CSV.open("user_data.csv", "a+") do |csv|
  csv << [@date, total_calories]
end
```

Key logics of the application - error handling

```
#Error handling: If the `food_calorie_file.csv` cannot be located,  
#1)print a error message, 2)create a new csv file with the data, and 3)ensure the user can proceed navigating the app normally.  
begin  
  @food_calorie_table = CSV.read("food_calorie_file.csv", headers: true, header_converters: :symbol)  
rescue  
  puts "Alert: Couldn't open the food calorie file, so a new food calorie file has been created for you."  
  puts "Select the above menu option you want to access (1-4)."  
  system('touch food_calorie_file.csv')  
  CSV.open("food_calorie_file.csv", "a+") do |csv|  
    csv << [:food_item, :calories]  
    csv << ['Soy milk with muesli', '236']  
    csv << ['Quinoa salad', '222']  
    csv << ['Avocado toast', '157']  
    ...  
  end  
  @food_calorie_table = CSV.read("food_calorie_file.csv", headers: true, header_converters: :symbol)  
end
```

Key logics of the application - error handling

```
when 4
  puts "Goodbye"
  exit(0)
else #Error handling: If the user input is wrong, print a helpful message so that the user
can enter valid entry.
  puts "Invalid entry: Please input one of the numeric option between 1-4.".colorize(:red)
  AddCalories.new().handle_input
end
```

Reflection

Challenges:

- Thinking about logic details, sequence at the flowchart stage
- Debugging, Loop errors, Method usage
- Error handling
- Difficulty in composing right questions, reading the web resources and understanding the contents, applying the resource into solving my issues

Favourite parts:

- Project Management (flowchart, implementation planning)
- Making it work, and tuning the code (improving)

All done!