

# Terminal App Documentation

---

## Software Development Plan

---

### Application Function:

With the recent and rapid spread of viruses like COVID, Ebola, and Swine Flu. Businesses need to be able to adapt quickly and easily to the new world. The Ruby Terminal Ordering App will allow restaurants and bars to open their doors to customers and let them order without the need for human interaction. The reasons for developing this app are two-fold, firstly to learn more about how the Ruby programming language works, secondly to become a millionaire after graduating by selling this amazingly sophisticated application to every restaurant in the (possibly) post-apocalyptic world.

My target audience will be for the cautious patron who loves going out for a meal and drinks, but hates getting Covid or other highly infectious diseases.

The Ruby Terminal Ordering App will allow customers in restaurants and bars to order their food and drink without requiring waitstaff or a QR code. Once the Ordering App has started, the customer is greeted with an artistic 'welcome' message to make them feel comfortable from the start. Below the welcome message, a highly-customisable menu is displayed with the items available to order on the left, and the prices on the right. The customer is then prompted to input their order and quantity, in the result of a larger group, the Ordering App can handle multiple orders without having to restart each time. Once the table has ordered everything they would like, a simple input of 'done' is all that is required to send their order to the kitchen. They are then rewarded for using the Ordering App by receiving a confirmation of their order together with the total price owed, an artistic 'goodbye' message, and a request for a quick survey of their experience. The restaurant or bar management also receive the total amount of each order for cash-up at the end of the day, and the results of the survey, in a separate text file.

To clarify, the features of the app will involve:

- A 'welcome' and 'goodbye' message using ASCII art,
- A loop to continue taking orders until all customers have finished including quantities,
- A highly-customisable menu and code to allow quick implementation for a large number of businesses,
- A receipt with the total calculated and printed,
- A survey that uses TTY prompt to simplify the experience (making it more likely for the customer to leave a review)
- Creating txt files for kitchen orders, total amounts, and survey results.

## Features

### Welcome & Goodbye Art:

The Ordering App will display Ascii art which will be developed using the Ruby Gem 'Artii'. Initially I was planning on using a simple 'puts' string however, I didn't like how messy it made my code look so I will add the art to a .txt file and call it using a block inside a method.

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer Bar:** On the left, it shows the project structure with files: `ruby_ordering_spec...`, `ascii_goodbye_art.txt`, `ascii>Welcome_art.txt`, `daily_orders.txt`, `Gemfile`, `Gemfile.lock`, `menu_item.rb`, `menu.rb`, `order.rb`, `README.md`, `restaurant.rb` (which is selected and highlighted in blue), and `ruby_ordering.rb`. There are also sections for `OUTLINE` and `TIMELINE`.
- Terminal:** A terminal window titled `TERMINAL_APP` is open, showing line numbers 29 through 45. The code in the terminal is as follows:

```
def welcome_art
  File.readlines("ascii_welcome_art.txt")
  do |line|
    puts line
  end
end

def goodbye_art
  puts
  puts "Thanks for using:".colorize(:cyan).
  bold
  puts
  File.readlines("ascii_goodbye_art.txt")
  do |line|
    puts line
  end
end

def intro
  puts "At #name! Ordering Is Easier Than"
end
```
- Status Bar:** At the bottom, the status bar shows the current branch as `master`, file count as `0`, warning count as `0`, and a `Live Share` icon.
- Bottom Right:** There are icons for navigating between files (`ruby_ordering.rb` and `restaurant.rb`), a search icon, and a settings gear icon.

```
TERMINAL ... 1: ruby + ⌫ ⌘ ⌂ ⌃ ⌄ no changes added to commit (use "git add" and/or "git commit -a") samhammond@sams-mbp ordering-app % git add . samhammond@sams-mbp ordering-app % ruby ruby_ordering.rb
```

At Sam's Covid-Safe Restaurant Ordering Is Easier Than Ever!

Sam's Totally Trustworthy Sustenance Options for Humans

---

RED CURRY	.....	15.0
GREEN CURRY	.....	15.0
BEEF MASSAMAN	.....	15.5

master 0 0 Live Share Spaces: 4 UTF-8 LF Ruby ⌘ ⌓ ⌚ ⌚

The screenshot shows the Visual Studio Code interface. On the left is the sidebar with icons for file operations, search, and other development tools. Two files are open in the main editor area: `ruby_ordering.rb` and `restaurant.rb`. The `restaurant.rb` file contains Ruby code for generating ASCII art. The terminal below shows the output of the application, which includes a welcome message and two pieces of ASCII art: a sandwich and a pizza.

```
def welcome_art
  File.readlines("ascii_welcome_art.txt") do |line|
    puts line
  end
end

def goodbye_art
  puts
  puts "Thanks for using:".colorize(:cyan).bold
  puts
  File.readlines("ascii_goodbye_art.txt") do |line|
    puts line
  end
end
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: zsh + □ ×

Thanks for using:

Before You Go!

How easy was Sams Terminal App to use? Easy-Peasy  
Thanks for letting us know!

master\* ↵ ⌂ 0 △ 0 ⌂ Live Share Ln 64, Col 5 Spaces: 4 UTF-8 LF Ruby ⌂ ⌂

## Continue Taking Orders & Quantities Until Everyone Has Ordered:

The Ordering App will continue asking for orders until the customer has typed 'done'. The app will first ask for you to enter what you would like to order, the output has specific instructions on how to do this. It will then verify that the restaurant has that item on the menu before asking how many they would like to order. If the input is a valid number it will add the item with its quantity to the order before asking you for the next item. Once you type 'done' the loop will break.

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows two files: `ruby_ordering.rb` and `restaurant.rb`.
- Code Editor:** Displays the `ruby_ordering.rb` file content. The code is a loop that asks the user what they would like to order, checks the menu, and adds items to the order until the user types 'done'.
- Terminal:** Shows the output of running the script. It lists items with their prices: COKE (3.5), ICE TEA (3.5), and WATER (3.0). Then it prompts the user to order Red Curry, asking for quantity (2). It then prompts for Pad Thai, asking for quantity (3).
- Status Bar:** Shows the current branch is `master*`, there are 0 errors and 0 warnings, and the file is saved. The terminal tab is active, showing the command `1: ruby`. The status bar also shows the line count (Ln 64), column count (Col 19), spaces (Spaces: 4), encoding (UTF-8), line separator (LF), and language (Ruby).

The screenshot shows the terminal window with the following interaction:

```

COKE      .... 3.5
ICE TEA   .... 3.5
WATER     .... 3.0

What would you like to order? Type your choice, then hit enter. When you are finished, type 'done'
Red Curry
How many would you like?
2

What would you like to order? Type your choice, then hit enter. When you are finished, type 'done'
Pad Thai
How many would you like?
3

What would you like to order? Type your choice, then hit enter. When you are finished, type 'done'
Coke
How many would you like?
5

What would you like to order? Type your choice, then hit enter. When you are finished, type 'done'

```

The terminal shows the application's menu and the user's responses to order Red Curry, Pad Thai, Coke, and a quantity of 5.

To address error handling, friendly instructions are given when the wrong input is received and the loop will start again.

```
COKE      ..... 3.5
ICE TEA    ..... 3.5
WATER     ..... 3.0

What would you like to order? Type your choice, then hit enter. When you are finished, type 'done'
Banana
Sorry, please check the spelling and try again
What would you like to order? Type your choice, then hit enter. When you are finished, type 'done'
3
Sorry, please check the spelling and try again
What would you like to order? Type your choice, then hit enter. When you are finished, type 'done'
Pad Thai
How many would you like?
0
Sorry, please enter a valid number and start again
What would you like to order? Type your choice, then hit enter. When you are finished, type 'done'
Pad Thai
How many would you like?
two
Sorry, please enter a valid number and start again
What would you like to order? Type your choice, then hit enter. When you are finished, type 'done'
DONE
```

## A highly-customisable menu and code to allow quick implementation for a large number of businesses:

I will separate the code into classes to help quick installation over a variety of businesses. Menu, Restaurant, Order & MenuItem won't require any change to the code. When installing the Ordering App in a new restaurant it will only take a minute or two to set-up by simply opening the main.rb file and changing the name of the restaurant, and replacing the items and prices in the 'menu'. Your new restaurant is now ready to go!

The screenshot shows the VS Code interface with the following details:

- Explorer:** Shows a file tree with a terminal application open. The terminal directory is `ordering-app`, which contains `spec`, `ascii_goodbye_art.txt`, `ascii_welcome_art.txt`, `daily_orders.txt`, `Gemfile`, `Gemfile.lock`, `menu_item.rb`, `menu.rb`, `order.rb`, `README.md`, `restaurant.rb`, and `ruby_ordering.rb`.
- Terminal:** The command `ruby ruby_ordering.rb` is run, displaying a ASCII art logo of a person holding a tray with food items.
- Output:** Text indicating "At Sam's Covid-Safe Restaurant Ordering Is Easier Than Ever!" followed by "Sam's Totally Trustworthy Sustenance Options for Humans". A table of food items and their prices is shown:

Item	Price
RED CURRY	..... 15.0
GREEN CURRY	..... 15.0
BEEF MASSAMAN	..... 15.5
PAD THAI	..... 12.5
PAD SEE EW	..... 12.5
PAD KEE MAO	..... 12.5
CASHEW NUT	..... 12.0
CHILLI BASIL	..... 12.0
TOM YUM SOUP	..... 5.0
COKE	..... 3.5
ICE TEA	..... 3.5
WATER	..... 3.0

**Bottom Status Bar:** master\* | Live Share | Ln 31, Col 17 | Spaces: 4 | UTF-8 | LF | Ruby | 🔍 | 📈

The screenshot shows the VS Code interface with the following details:

- Explorer:** Shows a file tree with a terminal application open. The terminal directory is `ordering-app`, which contains `spec`, `ascii_goodbye_art.txt`, `ascii_welcome_art.txt`, `daily_orders.txt`, `Gemfile`, `Gemfile.lock`, `menu_item.rb`, `menu.rb`, `order.rb`, `README.md`, `restaurant.rb`, and `ruby_ordering.rb`.
- Terminal:** The command `ruby ruby_ordering.rb` is run, displaying a ASCII art logo of a person holding a tray with drink items.
- Output:** Text indicating "At Sam's Covid-Safe Bar Ordering Is Easier Than Ever!" followed by "Sam's Totally Trustworthy Sustenance Options for Humans". A table of drink items and their prices is shown:

Item	Price
BEER	..... 6.5
SPIRITS	..... 9.0
WINE	..... 7.5
COCKTAILS	..... 18.0

**Bottom Status Bar:** master\* | Live Share | Ln 30, Col 50 | Spaces: 4 | UTF-8 | LF | Ruby | 🔍 | 📈

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure under "TERMINAL\_APP" with files like "ascii\_goodbye\_art.txt", "ascii>Welcome\_art.txt", "daily\_orders.txt", "Gemfile", "Gemfile.lock", "menu\_item.rb", "menu.rb", "order.rb", "README.md", and "restaurant.rb".
- Terminal View:** Displays the contents of "ruby\_ordering.rb" and "restaurant.rb".
- Status Bar:** Shows "master\*", file status icons (0 changes), "Live Share" button, line 31, column 50, spaces: 4, UTF-8 encoding, LF line endings, Ruby language, and a few other icons.

```

ruby_ordering.rb X
ordering-app > ruby_ordering.rb
23   exit
24 end
25
26 # menu = {"RED CURRY" => 15.00, "GREEN CURRY" =>
27 #           15.00, "BEEF MASSAMAN" => 15.50, "PAD THAI" => 12.
28 #           50, "PAD SEE EW" => 12.50, "PAD KEE MAO" => 12.
29 #           50, "CASHEW NUT" => 12.00, "CHILLI BASIL" => 12.
30 #           00, "TOM YUM SOUP" => 5.00, "COKE" => 3.50, "ICE
31 #           TEA" => 3.50, "WATER" => 3.00}
32
33 menu = {"BEER" => 6.50, "SPIRITS" => 9.00, "WINE"
34 => 7.50, "COCKTAILS" => 18.00}
35
36 # restaurant = Restaurant.new("Sam's Covid-Safe
37 # Restaurant", menu)
38 restaurant = Restaurant.new("Sam's Covid-Safe
39 Bar", menu)
40 puts
41 puts restaurant.welcome_art
42 restaurant.intro

```

## Creating txt files for kitchen orders, total amounts, and survey results:

.txt files will be created for the orders, cash sales, and survey. This will help management keep track of the daily sales and help me refine later versions with the feedback given in the survey. To do this for the total amount & survey I will define a method that utilises the 'File' class in Ruby and call it towards the end of the program. Orders and amount will go to the kitchen as soon as the customer orders the item.

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure under "TERMINAL\_APP" with files like "daily\_orders.txt", "Gemfile", "Gemfile.lock", "menu\_item.rb", "menu.rb", "order.rb", "README.md", and "restaurant.rb".
- Terminal View:** Displays the contents of "restaurant.rb".
- Status Bar:** Shows "master\*", file status icons (0 changes), "Live Share" button, line 74, column 5, spaces: 4, UTF-8 encoding, LF line endings, Ruby language, and a few other icons.

```

ruby_ordering.rb X
restaurant.rb X
menu.rb
ordering-app > restaurant.rb
65 def daily_orders
66   File.write("daily_orders.txt", "$%.2f" % "#{order_total}",
67             mode: "a")
68 end
69
70 def review_app
71   prompt = TTY::Prompt.new(active_color: :cyan)
72   puts "Before You Go!".colorize(:magenta).bold
73   puts
74   review = prompt.select("How easy was Sams Terminal App to use?".
75                         colorize(:magenta), %w(Easy-Peasy Normal Hard Impossible))
76   File.write("daily_orders.txt", " - #{review}\n", mode: "a")
77 end
78
79
80 end

```

The screenshot shows the VS Code interface with the terminal tab active, displaying the contents of the file `daily_orders.txt`. The file contains a list of 13 orders, each with a number, price, and difficulty level. The terminal status bar indicates the file is 14 lines long.

Line Number	Description
1	\$37.50 - Hard
2	\$42.00 - Normal
3	\$0.00 - Easy-Peasy
4	\$85.00 - Easy-Peasy
5	\$30.00 - Easy-Peasy
6	\$12.00 - Easy-Peasy
7	\$15.50 - Easy-Peasy
8	\$0.00 - Easy-Peasy
9	\$0.00 - Normal
10	\$77.50 - Hard
11	\$0.00 - Easy-Peasy
12	\$0.00 - Normal
13	\$150.00 - Impossible
14	

The screenshot shows the VS Code interface with the code editor tab active, displaying the file `order.rb`. The code defines a `Order` class with methods for initializing, adding items, and writing order items to a file. The line `File.write("user_order_items.txt", "#{name}" + "..." + "#{amount}, ", mode: "a")` is highlighted with a blue selection.

```
1  class Order
2    def initialize
3      @order_items = Hash.new(0)
4    end
5    def add_item(name, amount)
6      @order_items[name] += amount
7      #   ^ key      ^ value
8      File.write("user_order_items.txt", "#{name}" + "..." + "#{amount}, ", mode: "a")
9      # return
10   end
11   def find_items
12     return @order_items
13   end
14 end
```

The screenshot shows the VS Code interface with a terminal window open. The terminal output is as follows:

```

ordering-app > src > user_order_items.txt
1 PAD THAI - 3, GREEN CURRY - 1, CHILLI BASIL - 4,
TOM YUM SOUP - 9, COKE - 9,
TERMINAL ... 1: zsh + ×
Your order has been processed!
Total: $192.00
Thanks for using:

```

## Total calculated and printed on receipt:

The item price will be multiplied by the quantity and then added to the total. The bill will then be printed-out once the user types 'done'!

The screenshot shows the VS Code interface with a terminal window open. The terminal output is as follows:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: ruby + ×
RED CURRY ..... 15.0
GREEN CURRY ..... 15.0
BEEF MASSAMAN ..... 15.5
PAD THAI ..... 12.5
PAD SEE EW ..... 12.5
PAD KEE MAO ..... 12.5
CASHEW NUT ..... 12.0
CHILLI BASIL ..... 12.0
TOM YUM SOUP ..... 5.0
Your order has been processed!
Total: $282.00
Thanks for using:
Before You Go!
How easy was Sams Terminal App to use? (Press ↑/↓ arrow to move and Enter to select)
> Easy-Peasy
Normal
Hard
Impossible

```

The screenshot shows a code editor interface with a dark theme. The top navigation bar includes tabs for 'main.rb', 'restaurant.rb' (which is the active file), and 'main\_spec.rb'. On the left side, there are several icons with blue circular badges indicating notifications: a file icon (1), a magnifying glass icon (1), a users icon (1), and a gear icon (1). The main workspace displays the 'restaurant.rb' file content:

```
51
52     def order_total
53         total = 0
54         @order.find_items.each do |item,amount|
55             total += @menu.find_price(item)* amount
56         end
57         return total
58     end
59
60     def print_order
61         puts
62         puts "Your order has been
63             processed!\n\nTotal:      $%.2f".colorize(:cyan).
64             bold % order_total
65     end
```

## **Outline of User Interaction & Experience:**

There will be two different experiences which depends on whether the user is a customer or an employee.

## **Customer User:**

The customer will be greeted with a welcome message that is more than just a title, followed by a print-out of the menu available to them. They will find out how to interact with the app by following the simplified instructions that prints-out underneath the menu. If the user follows the first instruction correctly, they will input their order into the command line and push 'enter'. The same method is then used to get how many of that item they would like. The customer will keep interacting with the loop feature until they complete their order, once complete, the simple instructions make it clear to type 'done'. This will clear the screen and print the receipt, with some art created with a Ruby gem. The customer will then be asked to answer a one-question survey with a list of 4 options they can use the arrow keys to cycle through if they have the time.

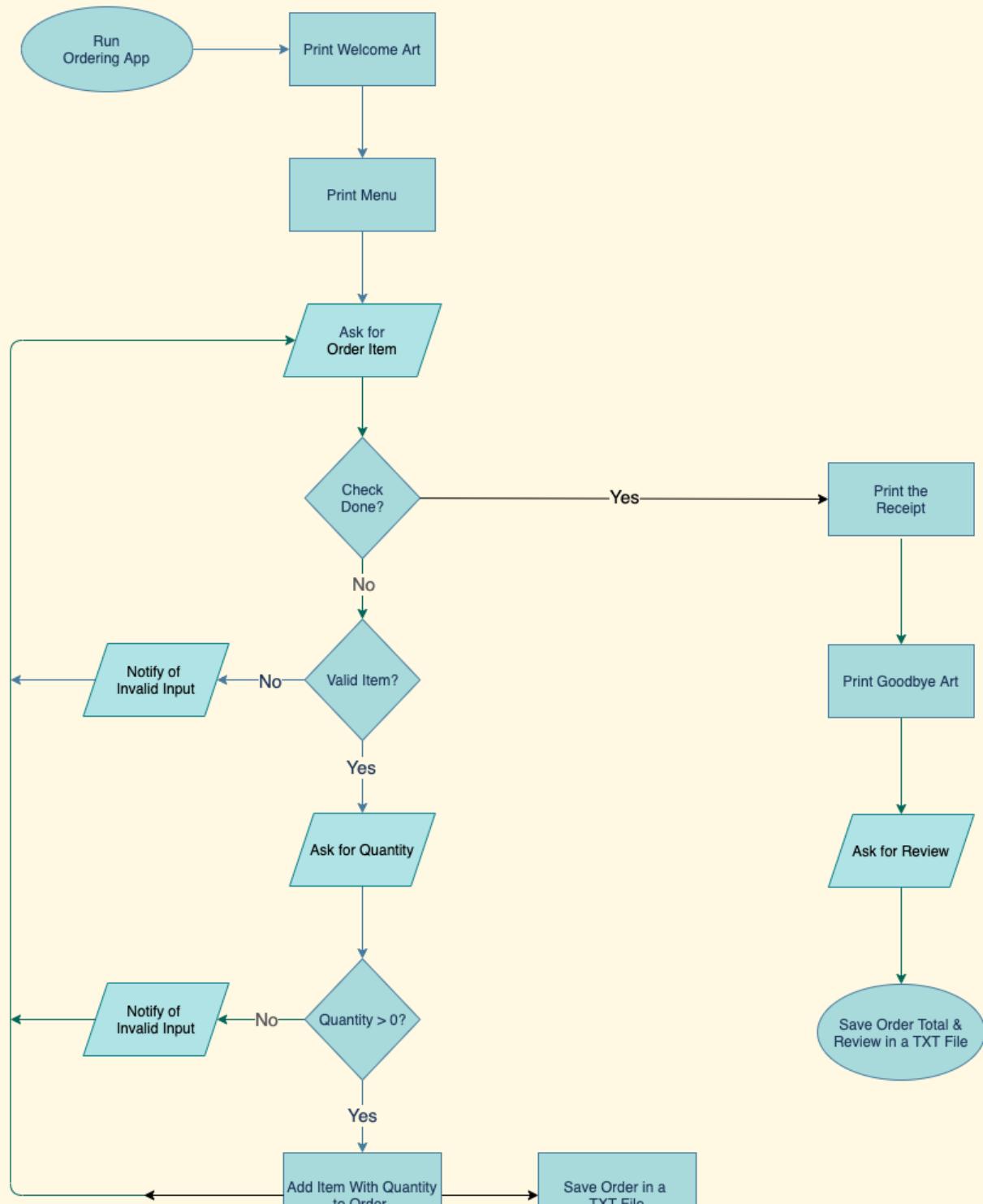
If the customer ever types an invalid input they will receive a friendly message asking them to please try again. This includes ordering '0' of something.

The customer can then leave the restaurant and head to the bar on the corner, which will be interacted with in the same way as the restaurant. Only the items to order and name showcased will be different.

## **Employee User:**

An employee will be able to interact with customer orders, amount sold in the day so far, and the survey answer by opening the .txt files in the app.

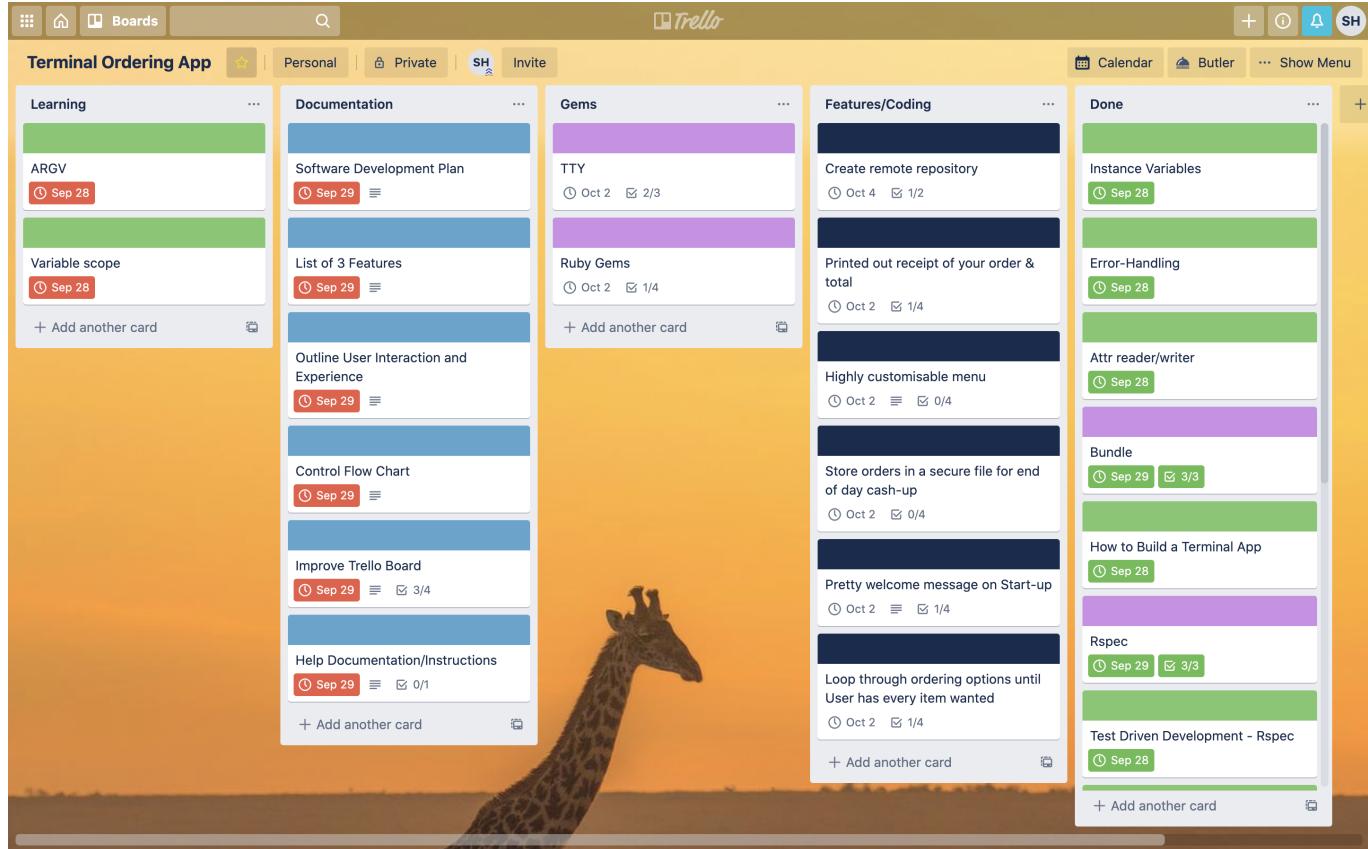
## Control Flow Chart:



## Link to My Implementation Plan

[Trello link](#)

I regret not taking a screenshot when I first created my plan, but below is a screenshot of where I was up to with 5 days left until my deadline:



## Instructions for Installation and Use

- Once published, you can clone or download the Ordering-App from my [Github page](#)
- You will need ruby installed on your computer. You can download it by clicking this [link](#)
- You will also need the Bundler gem installed. It should come installed with Ruby but you can also install it with the following input into your terminal: `gem install bundler`
- To install the gems, go to the src location of the app on your terminal and use the command: `bundle install`
- Run the app from the same location with the bash script: `./run_app.sh`
- The app should run on all 21st Century computers
- For more information about the app before running it type `ruby main.rb` followed by:
  - a or -all ... list all commands
  - h or -help ... show help
  - v or -version ... show which version is installed

**Link to my slide deck for the W.I.P presentation as the videos won't work in a pdf:**

[LINK](#)

