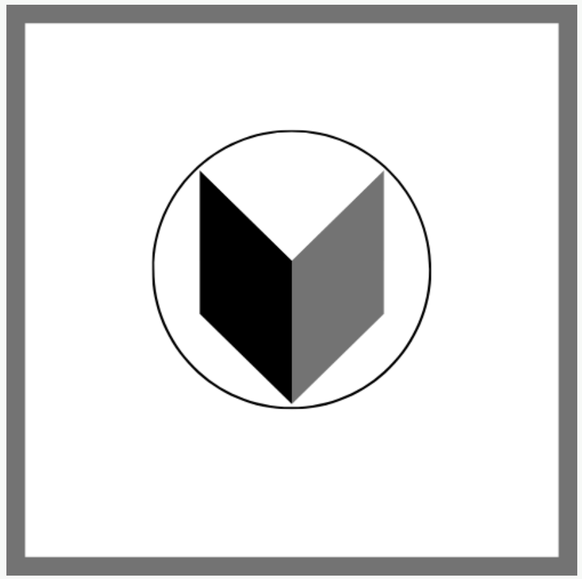
Isaac Cavallaro Terminal App T1A3



- **Coder Academy**
- **Term 1**
- **Assessment 3**

Purpose/scope

- **BeatTheBlues is a terminal app which generates a major 12 bar blues chord progression relative to user input.**

What problem does Beat the blues solve?

1. **The first aim of BeatTheBlues is to help beginning musicians learn three variations of the major 12 bar blues; in all twelve keys.**

What problem does Beat the blues solve?

2. The second aim of the app is to help beginning musicians decide what to practice in a given session. In short, alleviate decision fatigue.

Target Audience

2. The target audience of BeatTheBlues is beginner musicians who have a basic understanding of music theory.

Specifically, how to build major and minor chords from a root note.

Target Audience

- **The app is particularly useful for people wanting to learn music as a hobby and for those who don’t have access to a private tutor.**

Key Feature 1.1

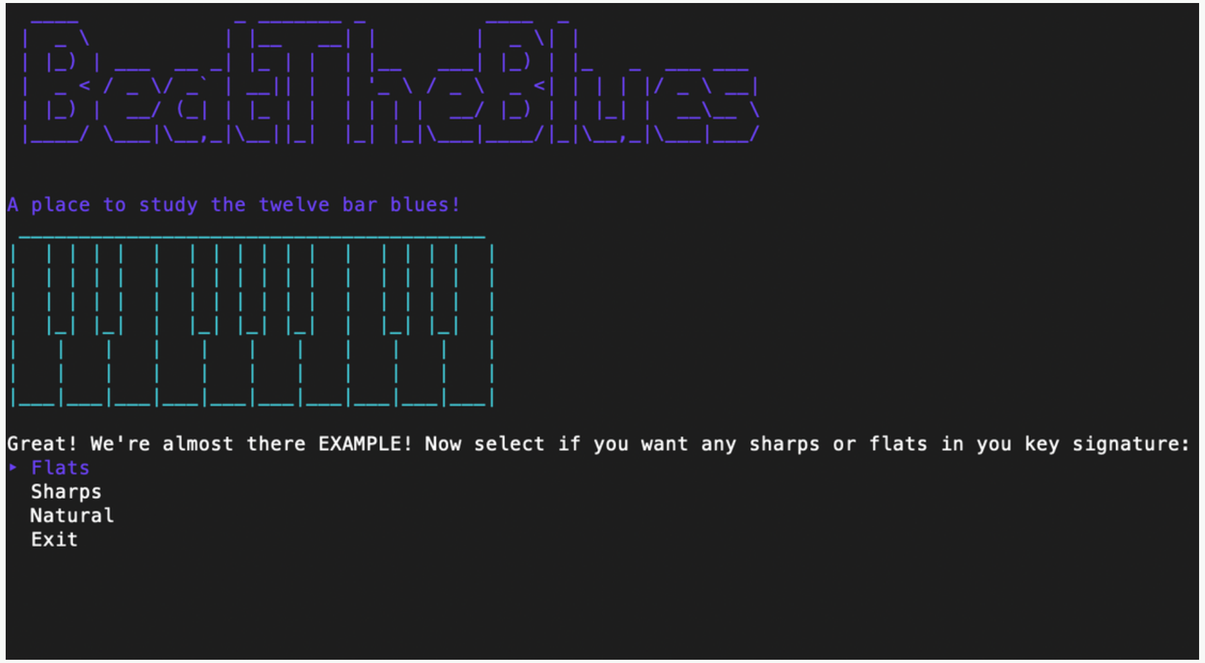
- **Users can specify the complexity (the number of chord changes) in the suggested progression.**

- BeatTheBlues will display the progression to the user and check if they are ready to move on.
- This way the user can easily reselect a new level if they wish; without having to exit the application.
  - This is the 12 bar Roman Numerals for Level One.

- This is the 12 bar Roman Numerals for Level Two.
- Added chords to challenge user.

- This is the 12 bar Roman Numerals for Level There.
- Added chords to challenge user.

Key Feature 1.2



- **Users can specify if they want their selected progression to include flats, sharps or neither (natural).**

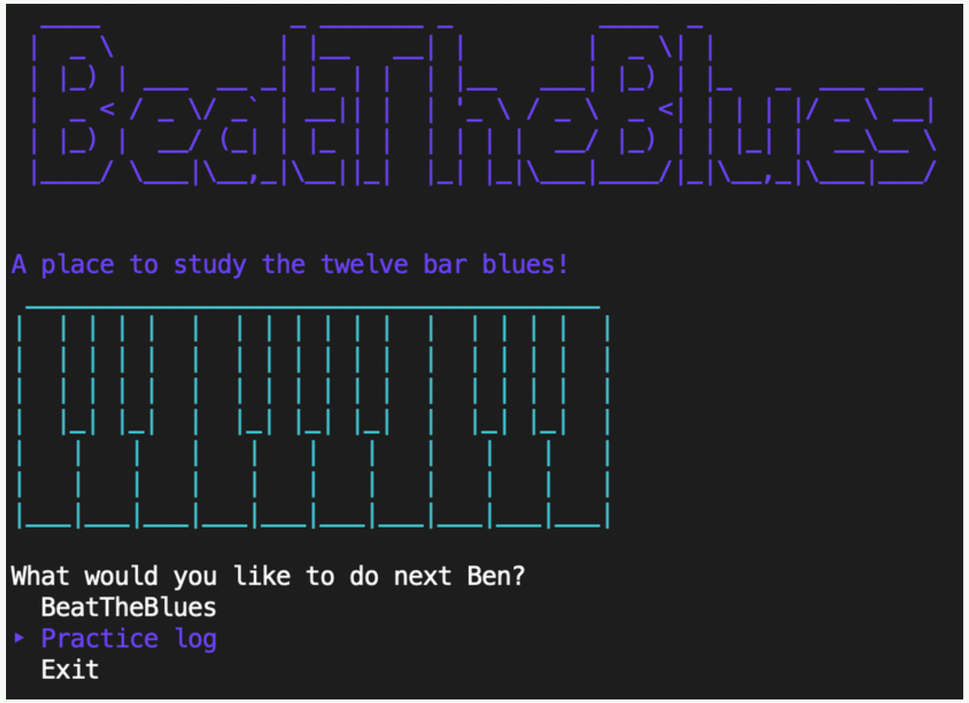
Key Feature 1.2

- **BeatTheBlues will confirm the selection with the user and check if they are ready to move on.**
- **Again, the user can easily reselect if they wish; without having to exit the application.**

Key Feature 2

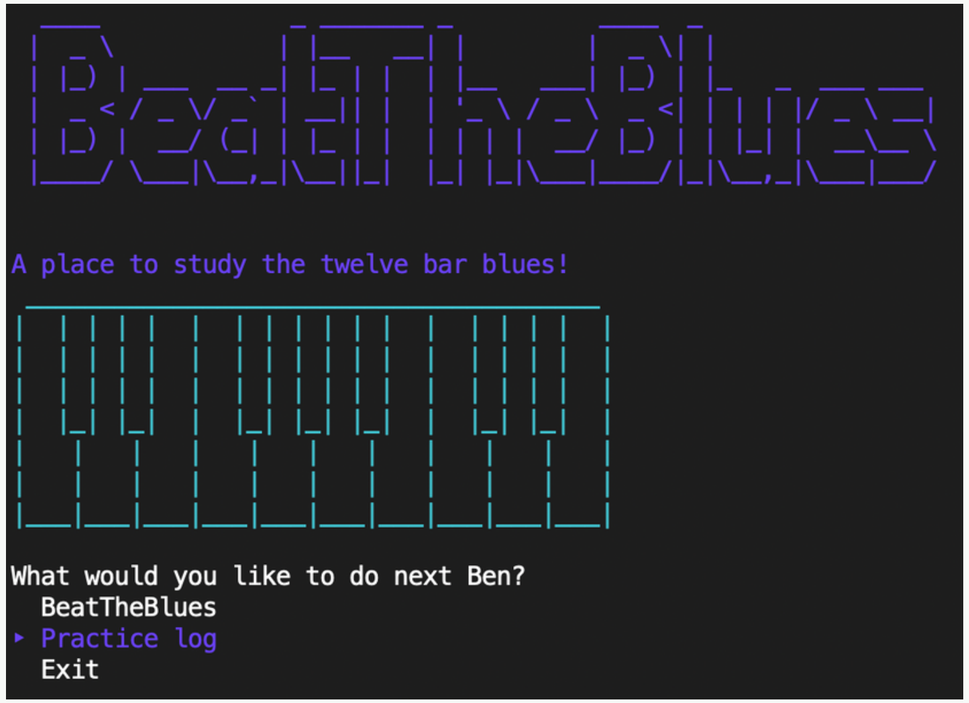
- **Users can select the lucky dip option.**
- **This will output a progression that could be from either level one, level two or level three difficulty and could be from any of the twelve keys.**

Key Feature 3



- Users can save their progress under their username.
- Multiple usernames can be stored and retrieved.
- Data can be stored and view at the start of the app and after a progression has been displayed.

Key Feature 3 (cont)



- **View and log at the start.**

Key Feature 3 (cont)

- **View and log at the end.**

Overview of Code

src > classes > chord_progression.rb > ...

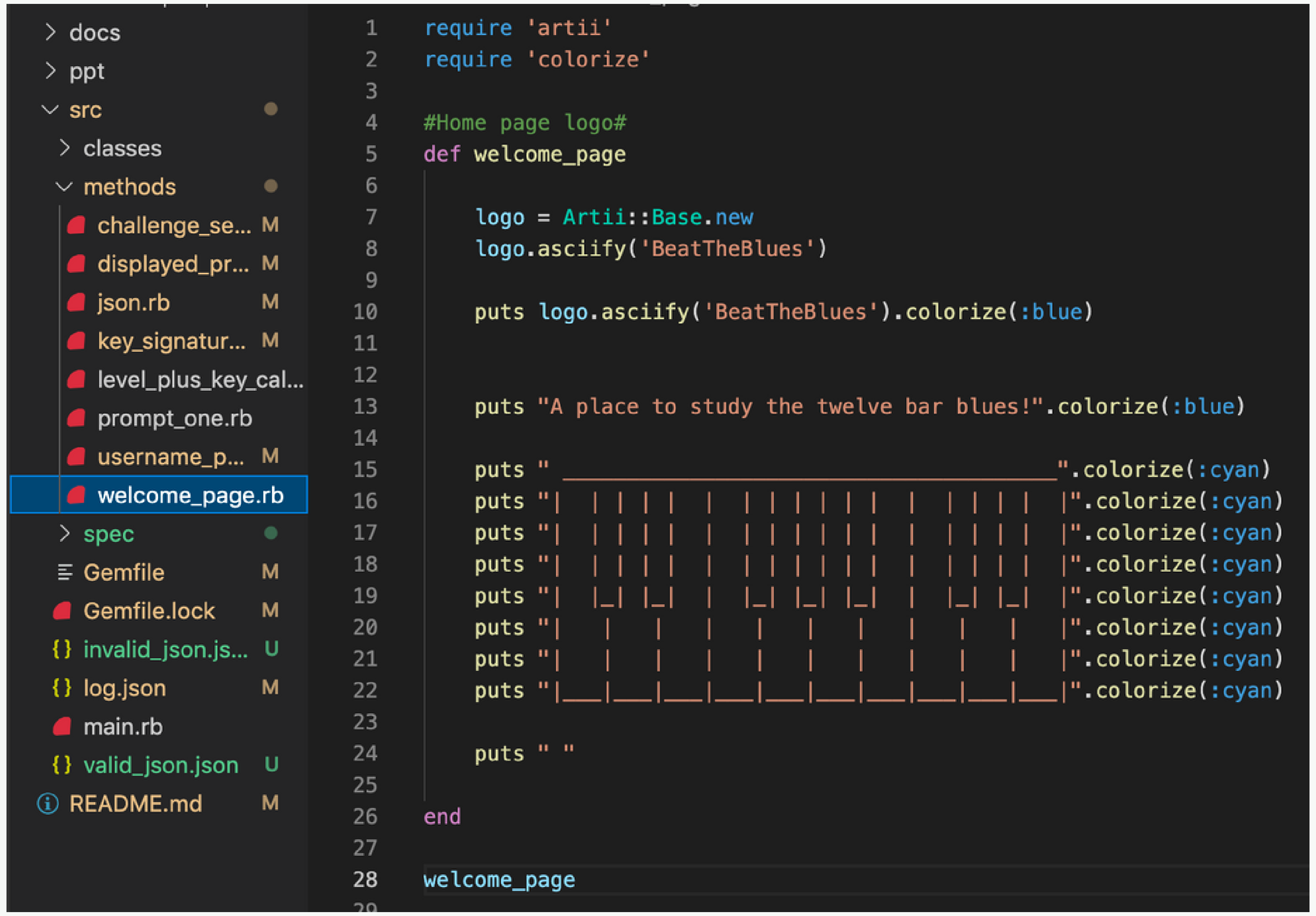
```ruby
########################################  CHORD PROGRESSION CLASS  ########################################
class ChordProgression

    attr_reader :progression, :key, :chords

    def initialize(progression, key, chords)
      @progression = progression
      @key = key
      @chords = chords
    end

    def to_s
        return "This progression #{@progression} in the key of #{@key}, translates to these chords #{@chords}"
    end

end

```

- **The project uses only one class.**
- **Which is responsible for displaying the progression (roman numeral), key (flats, sharps, natural) and the chords (translation) to the user.**

Example

- **In this example the user-selected Level One and Flats.**
- **Then BeatTheBlues displayed a random Level One flats progression and its translation (Cb Major in this example).**

Methods

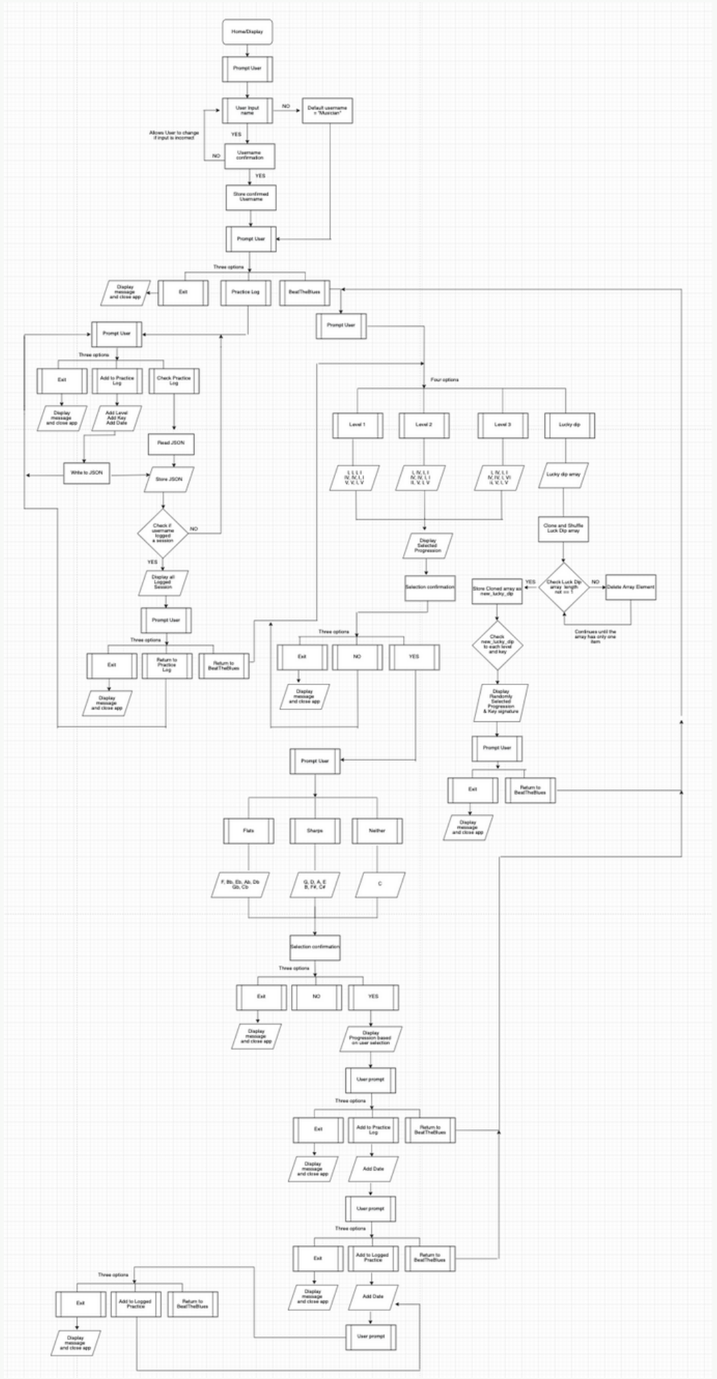


- The rest of the app is created with methods.
- Methods are located in their own files depending on their functions.

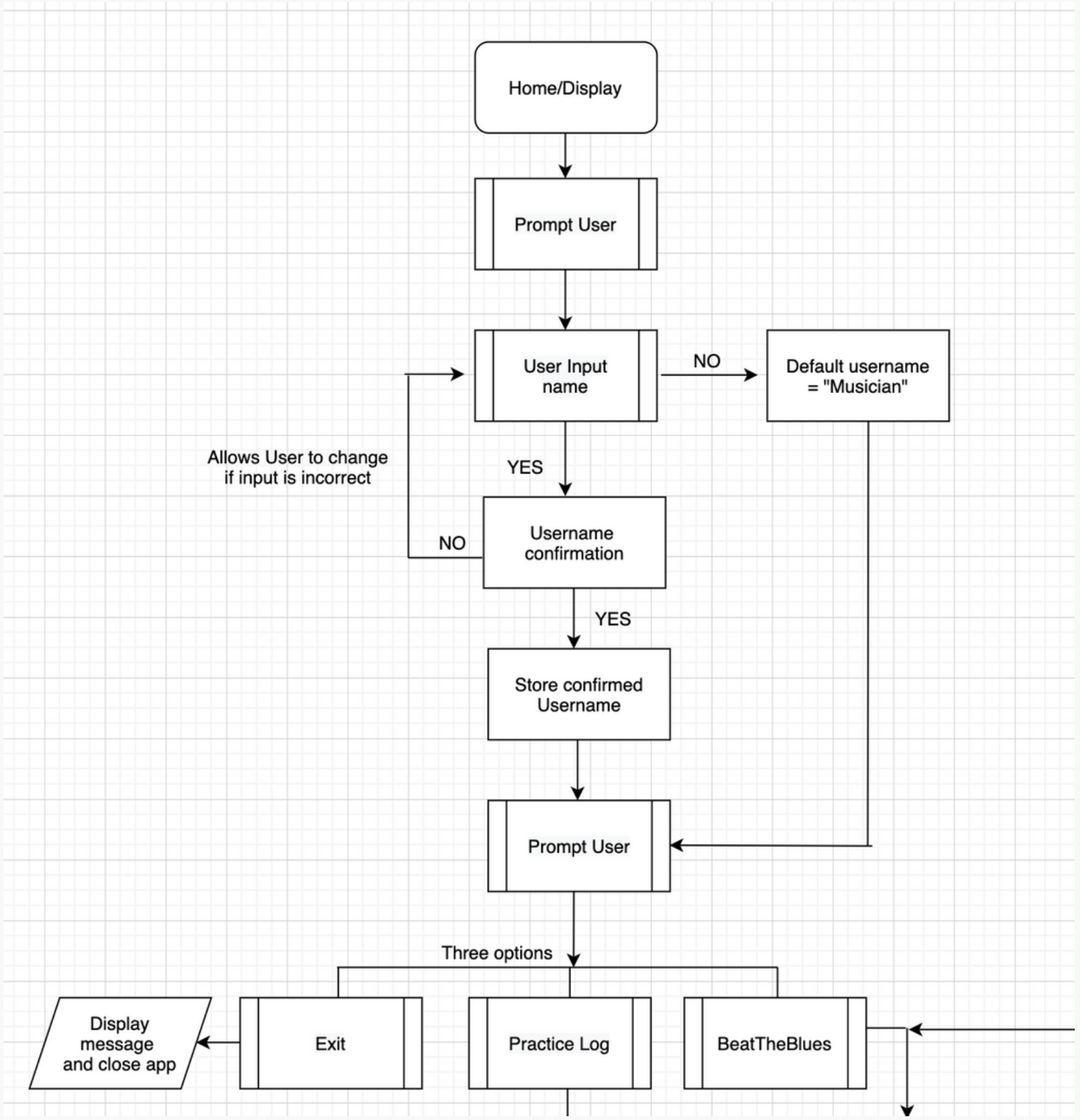
Gems

```
1   # frozen_string_literal: true
2
3   source "https://rubygems.org"
4
5   git_source(:github) { |repo_name| "https://github.com/#{repo_name}" }
6
7   # gem "rails"
8
9   gem "tty-prompt", "~> 0.23.1"
10
11  gem "rspec", "~> 3.10"
12
13  gem "colorize", "~> 0.8.1"
14
15  gem "artii", "~> 2.1"
16
17  gem "json", "~> 2.5"
18
```

- **In addition to the class and methods, the app makes use of a number of gems.**

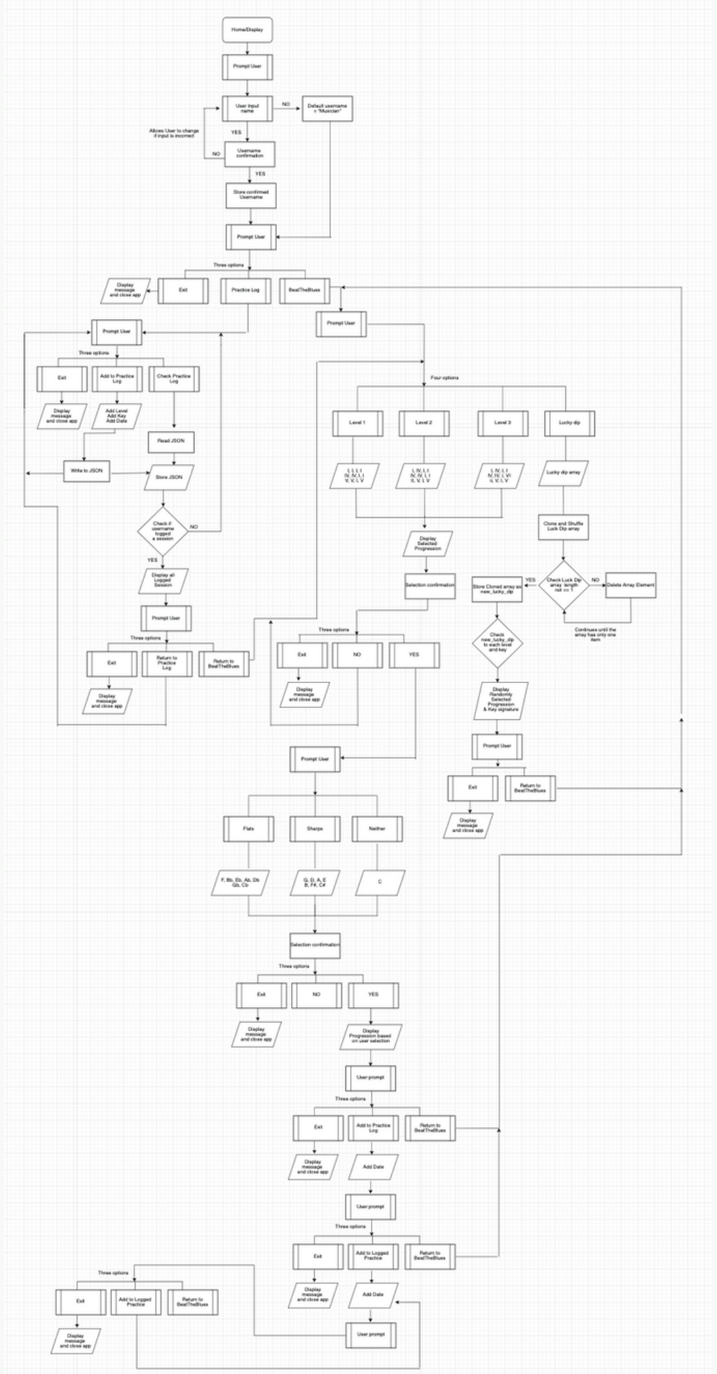


- **Here is the flow of the app from the start**

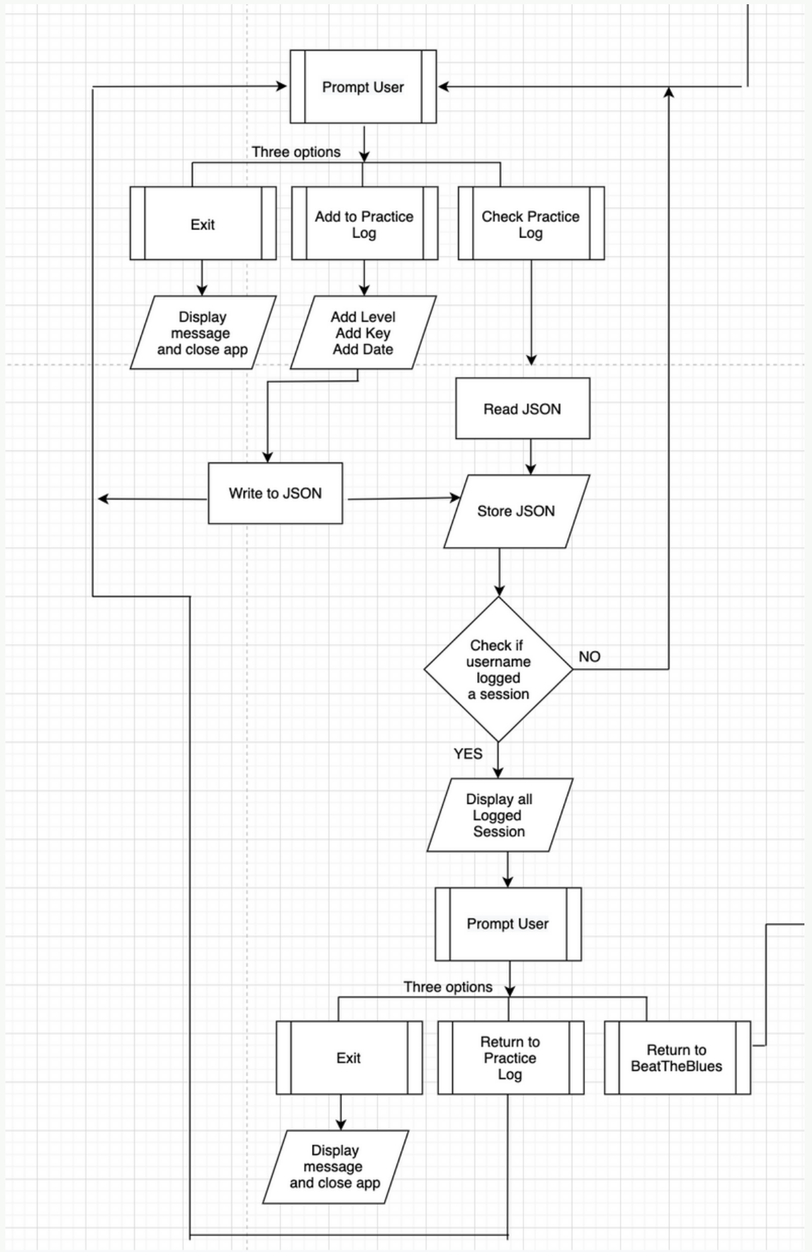


Prompting user, confirming and storing username.

Then selecting either Practice log or BeatTheBlues.



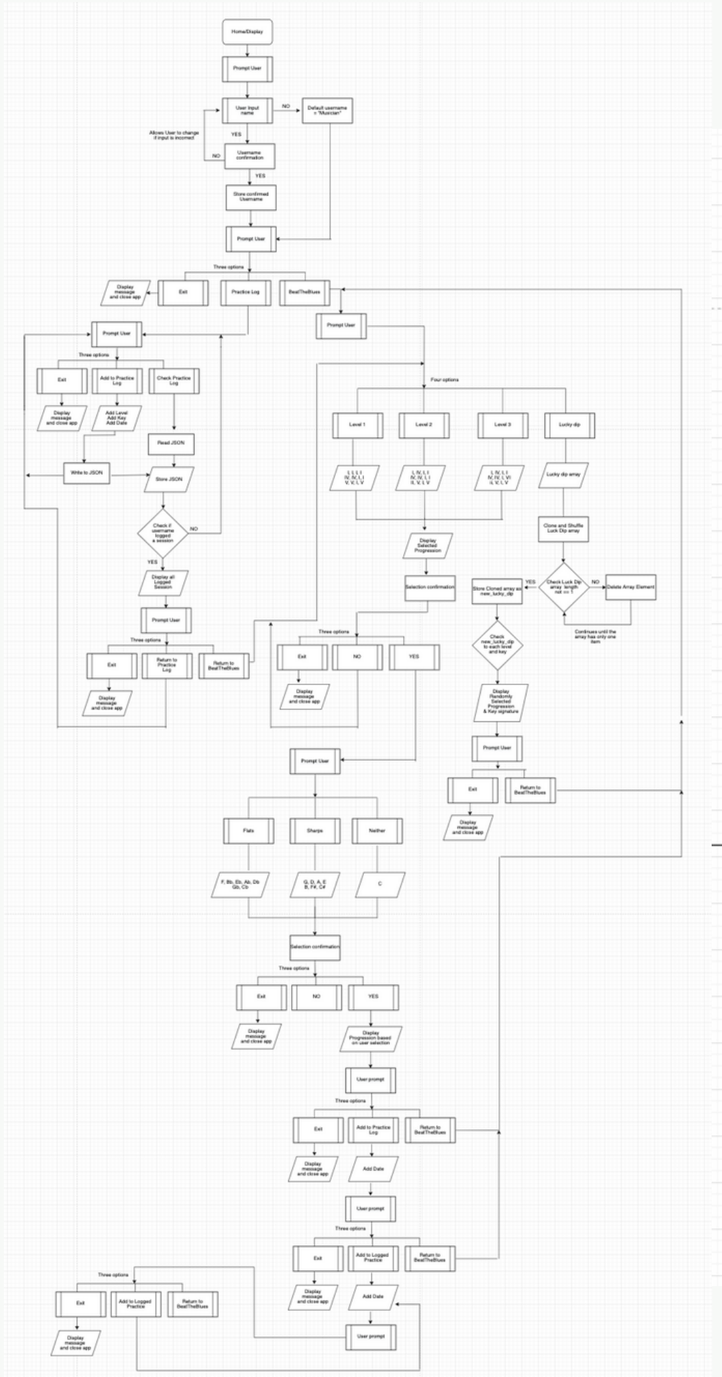
- **This is from Practice Log.**

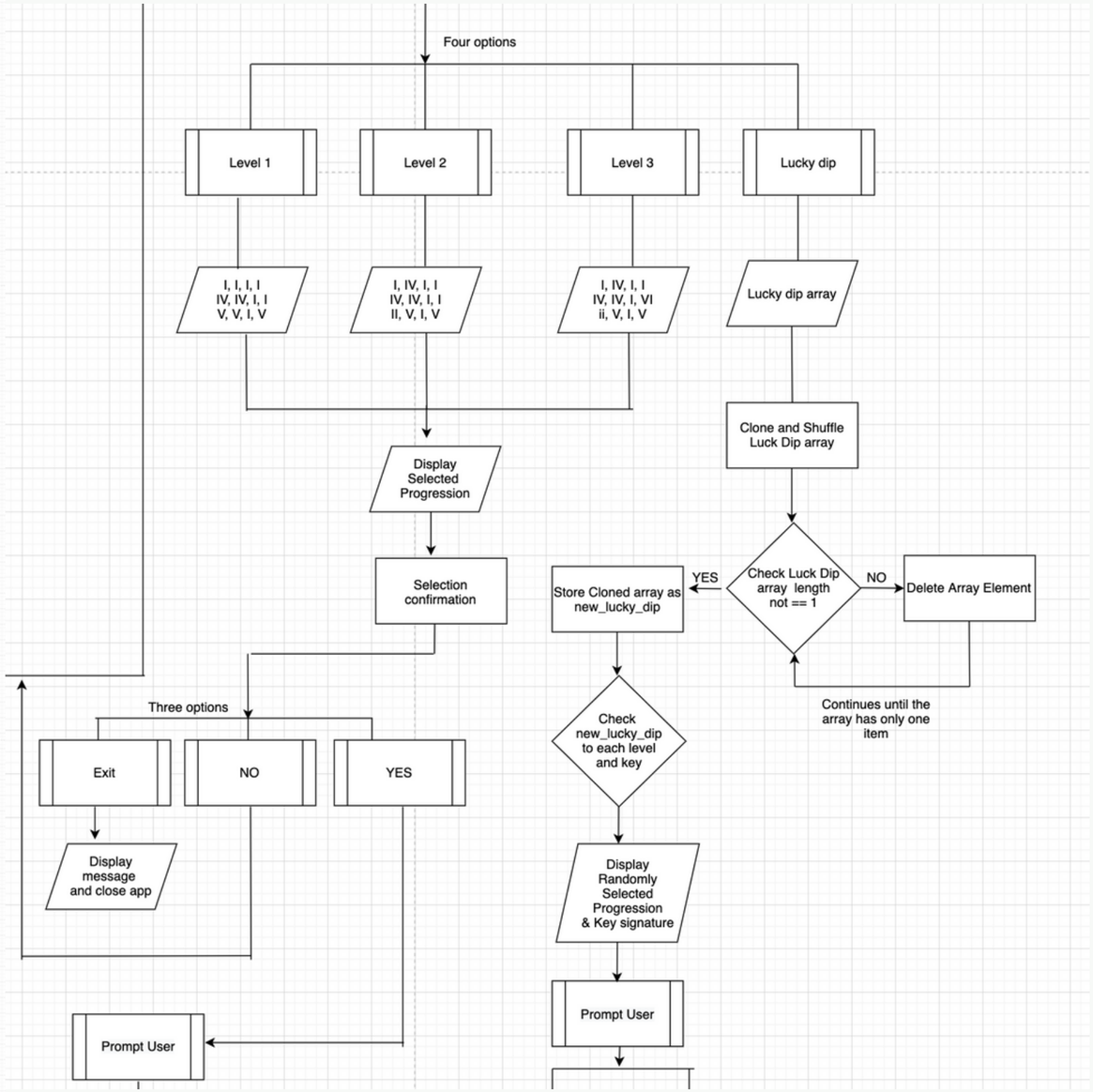


Allowing users to add the key, level and date to their logged practice.

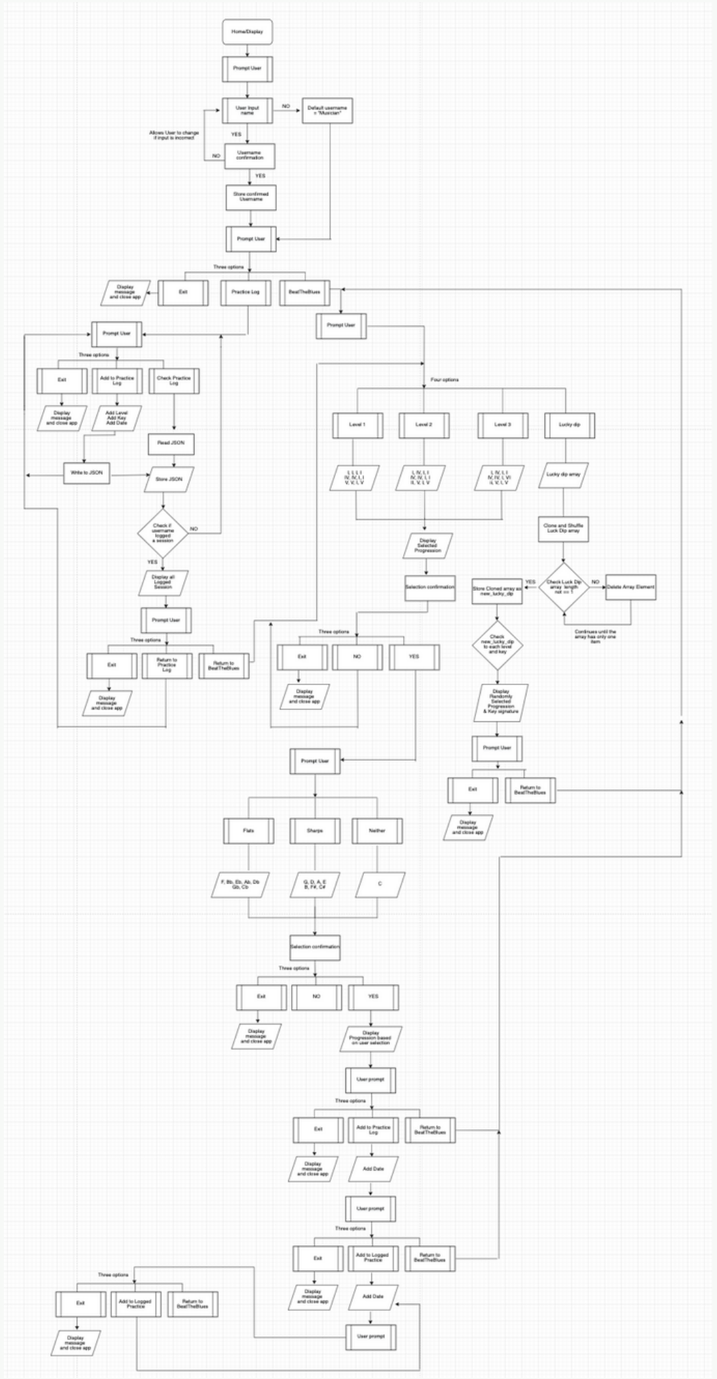
If their username is already logged in the json file, they can view all of their logged sessions .

- **This is from BeatTheBlues.**

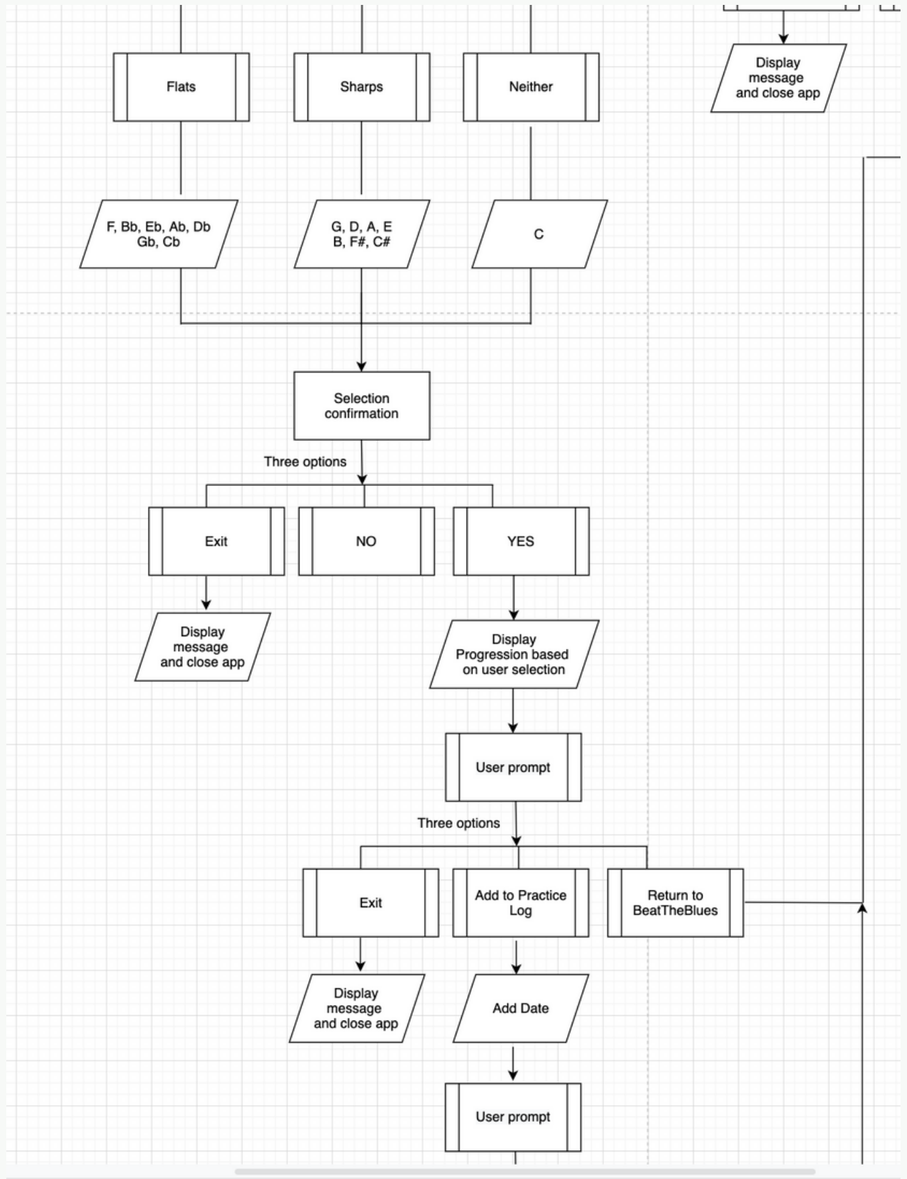




Allowing users to select level or lucky dip.



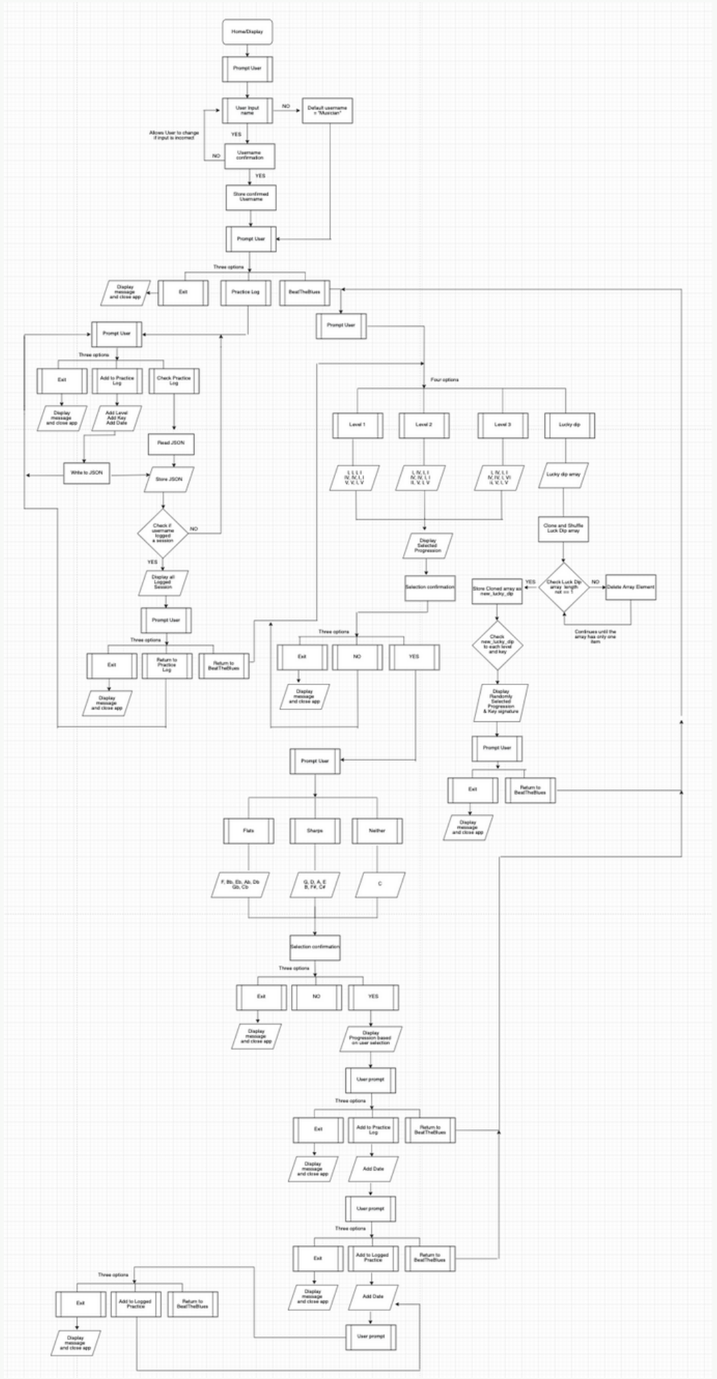
- **This is from BeatTheBlues (cont).**

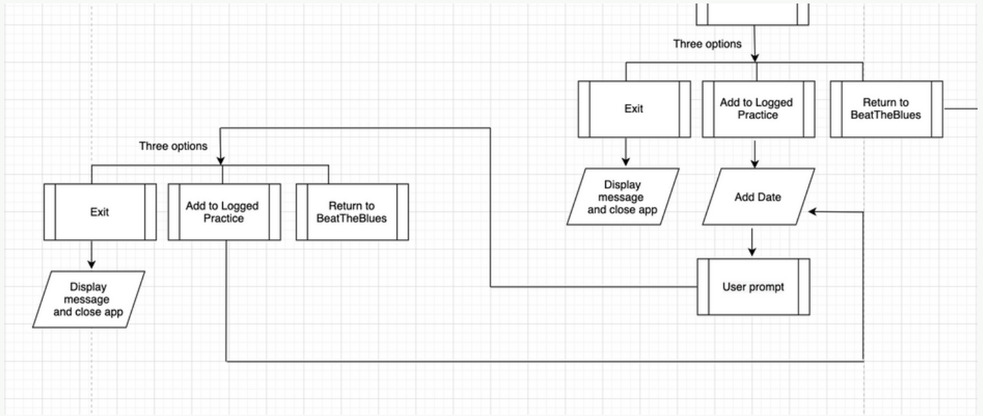


Displaying progression and allowing users to store their progression.

This time users only need to add a date as input given the key and level are saved then automatically added to the json file.

- **This is from BeatTheBlues (cont).**





From there user can return to BeatTheBlues.

Important parts of code

```ruby
require_relative '../classes/chord_progression'
require 'tty-prompt'
require 'colorize'

########################################  LEVEL ONE PROGRESSIONS  ############################################

########  Level One Flat Progressions - global variables #######
$level_one_f_blues = ChordProgression.new([['I','I','I','I'],['IV','IV','I','I'],['V','V','I','V']],"F Major",[['F','F','F','F'],['Bb','Bb','F','F'],['C','C','F','C']])
$level_one_b_flat_blues = ChordProgression.new([['I','I','I','I'],['IV','IV','I','I'],['V','V','I','V']],"Bb Major",[['Bb','Bb','Bb','Bb'],['Eb','Eb','Bb','Bb'],['Eb','Eb','Bb','Eb']])
$level_one_e_flat_blues = ChordProgression.new([['I','I','I','I'],['IV','IV','I','I'],['V','V','I','V']],"Eb Major",[['Eb','Eb','Eb','Eb'],['Ab','Ab','Eb','Eb'],['Bb','Bb','Eb','Bb']])
$level_one_a_flat_blues = ChordProgression.new([['I','I','I','I'],['IV','IV','I','I'],['V','V','I','V']],"Ab Major",[['Ab','Ab','Ab','Ab'],['Db','Db','Ab','Ab'],['Eb','Eb','Ab','Eb']])
$level_one_d_flat_blues = ChordProgression.new([['I','I','I','I'],['IV','IV','I','I'],['V','V','I','V']],"Db Major",[['Db','Db','Db','Db'],['Gb','Gb','Db','Db'],['Ab','Ab','Db','Ab']])
$level_one_g_flat_blues = ChordProgression.new([['I','I','I','I'],['IV','IV','I','I'],['V','V','I','V']],"Gb Major",[['Gb','Gb','Gb','Gb'],['Cb','Cb','Gb','Gb'],['Db','Db','Gb','Db']])
$level_one_c_flat_blues = ChordProgression.new([['I','I','I','I'],['IV','IV','I','I'],['V','V','I','V']],"Cb Major",[['Cb','Cb','Cb','Cb'],['Fb','Fb','Cb','Cb'],['Gb','Gb','Cb','Gb']])

# Array of level one flats progressions
$level_one_flats_array = [$level_one_f_blues, $level_one_b_flat_blues, $level_one_e_flat_blues, $level_one_a_flat_blues, $level_one_d_flat_blues, $level_one_g_flat_blues, $level_one_c_flat_blues]

#Method to display a random level 1 progression with flats
def display_level_1_flats_progression(username, selected_level, selected_key, user_progression_check)

    puts $level_one_flats_array.sample
    check_in_prompt(username, selected_level, selected_key, user_progression_check)

end

```

All progressions are stored in global variables.

These variables are then stored in an array depending on the level and key.

Important parts of code (cont)

```ruby
require_relative '../classes/chord_progression'
require 'tty-prompt'
require 'colorize'

########################################  LEVEL ONE PROGRESSIONS  ############################################

########  Level One Flat Progressions - global variables #######
$level_one_f_blues = ChordProgression.new([['I','I','I','I'],['IV','IV','I','I'],['V','V','I','V']],"F Major",[['F','F','F','F'],['Bb','Bb','F','F'],['C','C','F','C']])
$level_one_b_flat_blues = ChordProgression.new([['I','I','I','I'],['IV','IV','I','I'],['V','V','I','V']],"Bb Major",[['Bb','Bb','Bb','Bb'],['Eb','Eb','Bb','Bb'],['Eb','Eb','Bb','Eb']])
$level_one_e_flat_blues = ChordProgression.new([['I','I','I','I'],['IV','IV','I','I'],['V','V','I','V']],"Eb Major",[['Eb','Eb','Eb','Eb'],['Ab','Ab','Eb','Eb'],['Bb','Bb','Eb','Bb']])
$level_one_a_flat_blues = ChordProgression.new([['I','I','I','I'],['IV','IV','I','I'],['V','V','I','V']],"Ab Major",[['Ab','Ab','Ab','Ab'],['Db','Db','Ab','Ab'],['Eb','Eb','Ab','Eb']])
$level_one_d_flat_blues = ChordProgression.new([['I','I','I','I'],['IV','IV','I','I'],['V','V','I','V']],"Db Major",[['Db','Db','Db','Db'],['Gb','Gb','Db','Db'],['Ab','Ab','Db','Ab']])
$level_one_g_flat_blues = ChordProgression.new([['I','I','I','I'],['IV','IV','I','I'],['V','V','I','V']],"Gb Major",[['Gb','Gb','Gb','Gb'],['Cb','Cb','Gb','Gb'],['Db','Db','Gb','Db']])
$level_one_c_flat_blues = ChordProgression.new([['I','I','I','I'],['IV','IV','I','I'],['V','V','I','V']],"Cb Major",[['Cb','Cb','Cb','Cb'],['Fb','Fb','Cb','Cb'],['Gb','Gb','Cb','Gb']])

# Array of level one flats progressions
$level_one_flats_array = [$level_one_f_blues, $level_one_b_flat_blues, $level_one_e_flat_blues, $level_one_a_flat_blues, $level_one_d_flat_blues, $level_one_g_flat_blues, $level_one_c_flat_blues]

#Method to display a random level 1 progression with flats
def display_level_1_flats_progression(username, selected_level, selected_key, user_progression_check)

    puts $level_one_flats_array.sample
    check_in_prompt(username, selected_level, selected_key, user_progression_check)

end

```

This image illustrates all the level one flats objects, the array of level one flat objects and the method that is called if the user selects level one and flats.

Important parts of code – Lucky Dip

```
########################################  LUCKY DIP ##############################################
def display_lucky_dip_progression(username)

    #Array of global variables which each hold their own array
    $lucky_dip_array = [$level_one_flats_array, $level_one_sharps_array, $level_one_c_blues, $level_two_flats_array, $level_two_sharps_array, $level_two_c_blues, $level_three_flats_array, $level_three_sharps_array, $level_three_c_blues]

    #Copy the array
    $clone_lucky_dip_array = $lucky_dip_array.clone

    #Reorder the array
    lucky_dip = $lucky_dip_array.shuffle
```

1. Clones the array of objects

(lucky_dip_array)

2. Shuffles the array and stores in a new variable.

(lucky_dip)

Important parts of code – Lucky Dip

3. Deletes array elements until there is only one item left in the array. Stores in new variable.

```
#Loop to delete elements from array
until lucky_dip.length == 1

    lucky_dip.shift
    new_lucky_dip = lucky_dip.shift

end
```

```ruby
######################################  LUCKY DIP ##########################################
def display_lucky_dip_progression(username)

    #Array of global variables which each hold their own array
    $lucky_dip_array = [$level_one_flats_array, $level_one_sharps_array, $level_one_c_blues, $level_two_flats_array, $level_two_sharps_array, $level_two_c_blues, $level_three_flats_array, $level_three_sharps_array, $level_three_c_blues]

    #Copy the array
    $clone_lucky_dip_array = $lucky_dip_array.clone

    #Reorder the array
    lucky_dip = $lucky_dip_array.shuffle

    #Loop to delete elements from array
    until lucky_dip.length == 1

        lucky_dip.shift
        new_lucky_dip = lucky_dip.shift

    end

    if new_lucky_dip == $level_one_flats_array

        final_lucky_dip = $level_one_flats_array.sample

        puts final_lucky_dip

        check_in_prompt_lucky_dip(username)

    elsif new_lucky_dip == $level_one_sharps_array

        final_lucky_dip = $level_one_sharps_array.sample

        puts final_lucky_dip

        check_in_prompt_lucky_dip(username)

    elsif new_lucky_dip == $level_two_flats_array

        final_lucky_dip = $level_two_flats_array.sample

        puts final_lucky_dip

        check_in_prompt_lucky_dip(username)

    elsif new_lucky_dip == $level_two_sharps_array

        final_lucky_dip = $level_two_sharps_array.sample

        puts final_lucky_dip

        check_in_prompt_lucky_dip(username)

    elsif new_lucky_dip == $level_three_flats_array

        final_lucky_dip = $level_three_flats_array.sample

        puts final_lucky_dip

        check_in_prompt_lucky_dip(username)

    elsif new_lucky_dip == $level_three_sharps_array

        final_lucky_dip = $level_three_sharps_array.sample

        puts final_lucky_dip

        check_in_prompt_lucky_dip(username)

    else puts "This lucky dip was not so lucky! Better luck next time"

        check_in_prompt_lucky_dip(username)

    end

end
```

4. Checks the variable against each array and then displays a sampled element from array.

Important parts of code

```
    else puts "This lucky dip was not so lucky! Better luck next time"

        check_in_prompt_lucky_dip(username)

    end

end
```

Sometimes the lucky dip is not very lucky!

```ruby
# WRITE TO JSON FILE FROM DISPLAYED_PROGRESSION

def displayed_progression_write_json_file(username, selected_level, selected_key, user_progression_check)

    system("clear")

    welcome_page

    file =  File.read(File.expand_path( "../log.json", __dir__))

    json = JSON.parse(file)

    log_hash = Hash.new

    log_hash["Name"] = username

    #checking stored user level
    if selected_level == 1
        log_hash ["Level"] = "Level One"
    end

    if selected_level == 2
        log_hash ["Level"] = "Level Two"

    end

    if selected_level == 3
        log_hash ["Level"] = "Level Three"
    end

    #checking stored user key
    if selected_key == 1
        log_hash ["Key"] = "Flats"
    end

    if selected_level == 2
        log_hash ["Key"] = "Sharps"

    end

    if selected_level == 3
        log_hash ["Key"] = "Natural"
    end

    puts "Add a date to your logged session".colorize(:blue)
    log_hash ["Date"] = gets.chomp

    json.push(log_hash)

    File.open('./log.json', 'w') do |f|
        f.puts JSON.pretty_generate(json)
    end
```

This is the first half of the method that allows the user to add to their practice log after completing BeatTheBlues.

This method is responsible for populating the JSON file with the level and key; without additional user input.

Challenges – JSON, testing.

```ruby
# WRITE TO JSON FILE FROM DISPLAYED_PROGRESSION

def displayed_progression_write_json_file(username, selected_level, selected_key, user_progression_check)

    system("clear")

    welcome_page

    file =  File.read(File.expand_path( "../log.json", __dir__))

    json = JSON.parse(file)

    log_hash = Hash.new

    log_hash["Name"] = username

    #checking stored user level
    if selected_level == 1
        log_hash ["Level"] = "Level One"
    end

    if selected_level == 2
        log_hash ["Level"] = "Level Two"

    end

    if selected_level == 3
        log_hash ["Level"] = "Level Three"
    end

    #checking stored user key
    if selected_key == 1
        log_hash ["Key"] = "Flats"
    end

    if selected_level == 2
        log_hash ["Key"] = "Sharps"

    end

    if selected_level == 3
        log_hash ["Key"] = "Natural"
    end

    puts "Add a date to your logged session".colorize(:blue)
    log_hash ["Date"] = gets.chomp

    json.push(log_hash)

    File.open('./log.json', 'w') do |f|
        f.puts JSON.pretty_generate(json)
    end
```

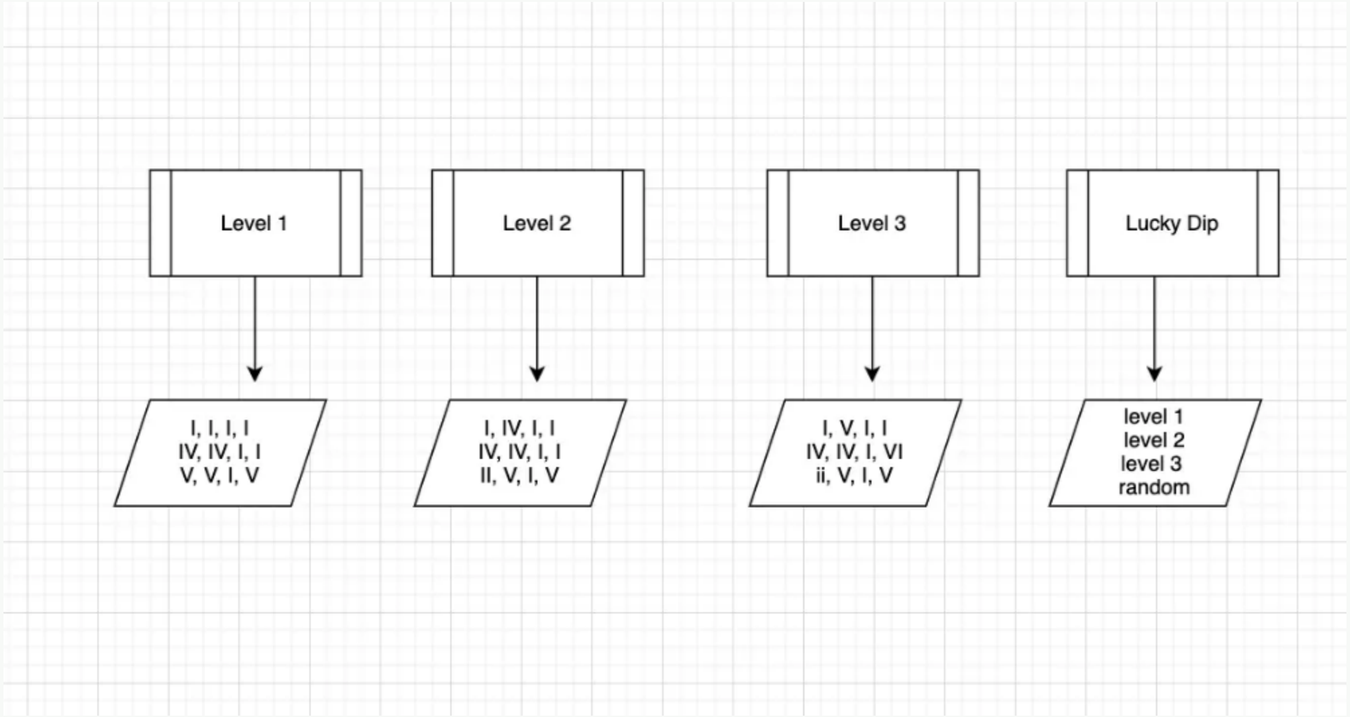
1.Creating JSON methods to work the way I want them to.

Reading and writing.

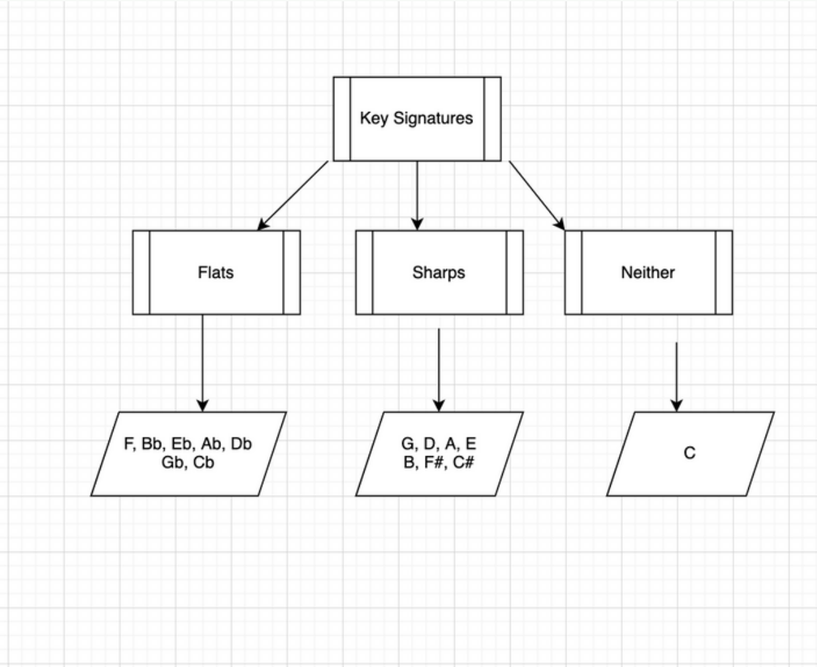
2. Getting the test to work.

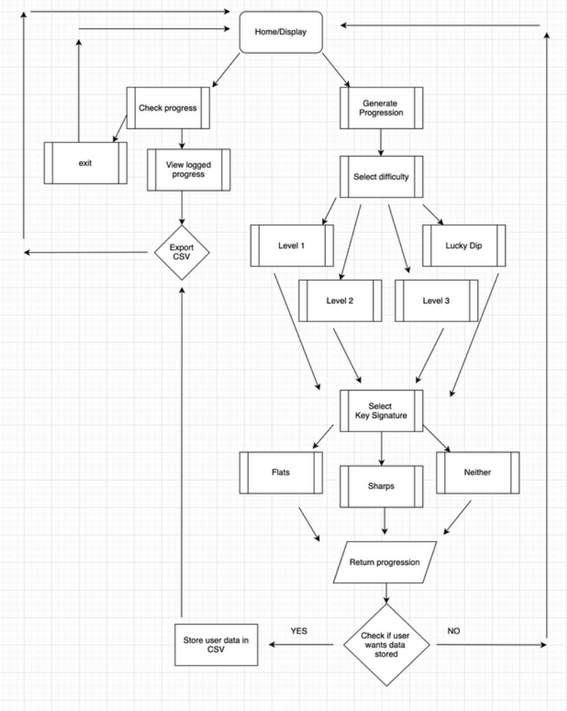
Build procress

1. **Identified a problem to solve.**
2. **Draft flow charts in chunks.**



Build process (cont)





Isaac Cavallaro – Coder Academy	May 2021

Ethical Issues

– Having a target audience kept me oriented on who the app was for (beginner musicians).

– While this was useful in helping direct my decisions, no ethical issues were present in the building of BeatTheBlues.

Thank you very much!

Thank you very much to Teej, Ash and my classmates for your continued support and sharing of resources.