



CODER DETOX SPA

RUBY TERMINAL APPLICATION

by Natalie Bottema



What is Coder Detox Spa?

- The coder detox spa is a two-part application
- The first part of the application allows the user to complete a quiz on programming in order to earn points for their spa wallet
- The second part of the application allows users to cash in their points to 'purchase' spa treatments

DESIGN PROCESS & DECISIONS

APPLICATION OBJECTIVES

1. Create an application that provides the user with an interactive and engaging way to revise programming concepts

- Implemented through the quiz, which aims to give the user a different approach to studying

2. Provide a virtual reward system for completing the quiz

- Implemented through the spa, which aims to be amusing and somewhat satirical

DESIGN PROCESS & DECISIONS

CONTROL FLOW OF APPLICATION

THE ORIGINAL CONTROL FLOW

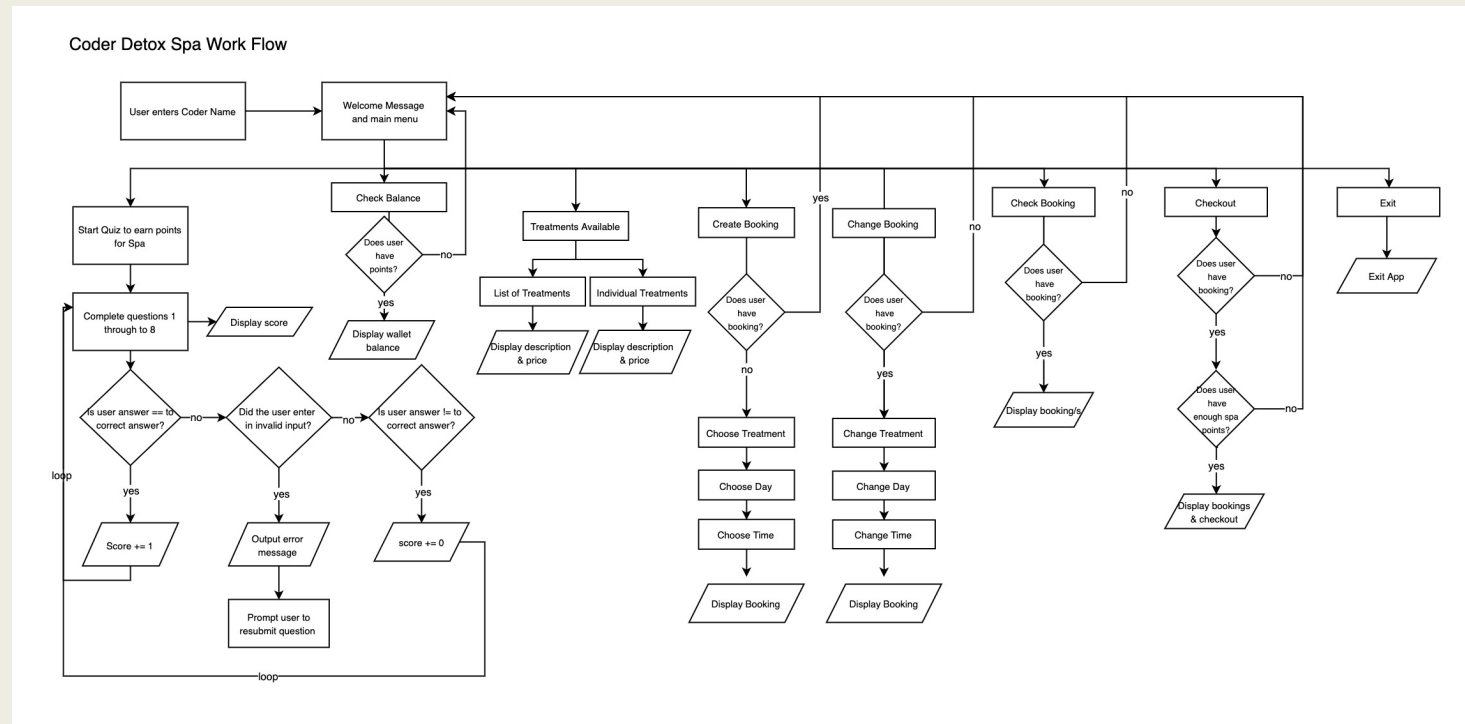
Three menu items in the main menu:

- Start Quiz
- Check Balance
- Enter Spa

THE UPDATED CONTROL FLOW

Spa Menu consolidated into main menu:

- Start Quiz
- Check Balance
- View Treatments
- Create Booking
- Change Booking
- Check Booking
- Checkout



OVERVIEW

APP FEATURES

- Menu
- Quiz
- Treatments
- Create a Booking
- Change a Booking
- Display Booking
- Check Spa Points
- Checkout

RUBY GEMS UTILISED

What would you like to do?

(Press ↑/↓/←/→ arrow to move and Enter to select)

► Start Quiz
Check Balance

You will be presented with a statement on a number of programming concepts, in which you will have provide a response.

- tty-prompt
- colorize
- Artii
- tty-font
- tty-progressbar
- tty-box
- pastel

Congratulations
You have earned 400 spa points.

CURRENT BOOKING

Check Balance

Checking your spa points balance...[=====

]■

Full List of Treatments

.....

OVERVIEW

MENU FEATURE

The Coder Detox Spa has three menus, which include:

- A Main Menu
- A Treatments Menu
- A Checkout Menu

CODE IMPLEMENTATION

MENU FEATURE

- Each menu utilises TTY-Prompt, which enables the user to select an option without the user entering invalid input
- All three menu's are structured utilising a while loop, which breaks when the user selects exit

FINAL DESIGN

MENU FEATURE

Welcome to Coder Detox Spa!

Please enter your name:

Natalie

Hello Natalie, welcome to the Coder Detox Spa! You can earn points to use at the Coder Detox Spa by completing a quiz on programming 🧐.

What would you like to do?

(Press ↑/↓/→ arrow to move and Enter to select)

- Start Quiz
- Check Balance
- Treatments Available
- Create Booking
- Change Booking
- Display Booking

Available Treatments!

Which treatment would you like to view?

(Press ↑/↓/→ arrow to move and Enter to select)

- Full List of Treatments
- A Tasty Treat
- Detox Facial
- New Hair, Who Dis?
- Coder Special
- Stack Overflow Enlightenment

FINALISE YOUR BOOKING

Oh no, Natalie. Before you exit, you need to finalise your account!!!
Please select from the following options:

(Press ↑/↓ arrow to move and Enter to select)

- Check Balance
- Amount Owning
- Checkout
- Exit

OVERVIEW

QUIZ FEATURE

- The user has the option to take a test on programming
- The user is presented with 5 statements, in which the user is required to answer with either “true” or “false”.
- When the user answers a question correctly they receive 50 points
- After finishing the quiz the user is presented with a final score

CODE IMPLEMENTATION

QUIZ FEATURE

A Quiz Class

- Attributes of question, answer, incorrect and score

An Array of Questions

- holds all new quiz class objects

A For loop

- loops through all questions.
- When the User answers correctly a score is allocated to a score variable

A SpaPoints Class

- stores the users final score from the quiz as a SpaPoints object

```
class Quiz
  attr_accessor :question, :answer, :incorrect, :score
  def initialize(question, answer, incorrect, score)
    @question = question
    @answer = answer
    @incorrect = incorrect
    @score = score
  end
end
```

```
# Question variables

q1 = "Ruby is an object-oriented programming language.".green
q2 = "HTML stands for HyperTyped Markup Language".green
q3 = "In FlexBox, Justified Content defines how to position elements vertically".green
q4 = "The Git Command 'git remote show origin' allows you see more information about a remote repo".green
q5 = "Subresource Integrity is a security feature that prevents files from being manipulated".green
q6 = "The comparable mixin '<=>' compares values on either side of it and can be used to sort values".green
q7 = "In Ruby, you can access variables in a method from outside of that method".green
q8 = "In HTML, the textarea element is used to create a checkbox in a form".green
```

```
def Questionnaire(questions)
  answer = ""
  score = 0
  attempts = 0
  begin
    attempts = attempts || 0
  end
  for question in questions
    puts question.question
    answer = gets.strip.downcase.to_s
    if answer == question.answer
      score += question.score
    elsif answer == question.incorrect
      score += 0
    else
      raise InvalidInputError.new "Invalid input entered. Please use 'true' or 'false' answers."
    end
  end
end
end
return score
end
```

```
class SpaPoints
  attr_accessor :wallet
  def initialize(wallet)
    @wallet = wallet
  end
end
```

FINAL DESIGN

QUIZ FEATURE

Programming Quiz

This is a quiz that you can take to test out your programming knowledge and earn valuable spa points, which can be used at the Coder Detox Spa.

You will be presented with a statement on a number of programming concepts, in which you will have provide a response.

If you believe a statement is correct, please type in 'true'.

On the other hand, if you believe a statement to be incorrect, please type in 'false'.

Ruby is an object-oriented programming language.

true

HTML stands for HyperTyped Markup Language

false

In FlexBox, Justified Content defines how to position elements vertically

false

The Git Command 'git remote show origin' allows you see more information about a remote repo

true

Subresource Integrity is a security feature that prevents files from being manipulated

■

OVERVIEW

TREATMENT FEATURE

- Allows the User to view either a full list of treatments or individual treatments
- Each treatment includes a name, a description and a price.
- This provides the user with what is involved in the treatment and how many spa points they need to book the treatment.

CODE IMPLEMENTATION

TREATMENT FEATURE

The code implemented in the treatment feature includes:

- A treatment class with the attributes of name, description and price
- THE FULL LIST OF TREATMENTS OPTION iterates over all individual treatments and puts to the screen for the user to view
- THE INDIVIDUAL TREATMENT OPTION utilises a case statement to determine what individual treatment to print and accesses that treatment's corresponding class attributes
- The user accesses this information through the treatment menu

```
class Treatment
  attr_accessor :name, :description, :price
  def initialize(name, description, price)
    @name = name
    @description = description
    @price = price
  end
end
```

```
def display_all_treatments(treatments)
  for treatment in treatments
    puts treatment.name
    puts "\n"
    puts treatment.description
    puts "\n"
    puts treatment.price
    puts "\n"
  end
end
```

```
def display_individual_treatments(treatment)
  puts "Treatment: #{treatment.name}\n\n".magenta
  puts "The Package: #{treatment.description}\n\n"
  puts "The Cost: #{treatment.price} spa points\n\n\n".green
end
```

FINAL DESIGN

TREATMENT FEATURE



OVERVIEW

CREATE A BOOKING FEATURE

The Create a Booking Feature allows users to create a booking for a treatment

Through the menu prompts the user will be able to:

- Select a treatment
- Select a day
- Select a time

After the user has selected all the above information, they will receive a message stating that they have secured a booking for their chosen treatment, day and time

CODE IMPLEMENTATION

CREATE A BOOKING FEATURE

The code implemented in the create a booking feature includes:

- Storing each piece of user input into a hash called booking. Storing the information in a hash will allow the program to easily display the booking details when the user requests them.
- Furthermore, a booking class will also be implemented to store the booking in a secondary location as booking1. When a hash value pair is produced, the values are stored as a new booking class object.

```
class Booking
  attr_accessor :treatment, :day, :time
  def initialize(treatment, day, time)
    @treatment = treatment
    @day = day
    @time = time
  end
end
```

```
when "Create Booking"
  if owing != nil
    # Create a booking heading
    create_booking_heading

    # User is prompted to select a treatment, answer is stored in booking hash
    answer = prompt.select("Please select your treatment:\n\n", %w(A\ Tasty\ Treat Detox\ Facial New\ Hair\ Who\ Dis? Coder\ Special Stack\ Overflow\ Enlightenment))
    booking[:treatment] = answer

    # User is prompted to select a day, answer is stored in booking hash
    answer = prompt.select("Please select a day you would like your treatment:\n\n", %w(Sunday Monday Tuesday Wednesday Thursday Friday Saturday))
    booking[:day] = answer

    # User is prompted to select a time, answer is stored in booking hash
    answer = prompt.select("Please select a time you would like to have your treatment:\n\n", %w(9:00 10:00 11:00 12:00 13:00 14:00 15:00 16:00 17:00))
    booking[:time] = answer

    # Display to user their booking details
    print "You have now secured a booking for #{booking[:treatment]} on #{booking[:day]} at #{booking[:time]}.\n\n"

  else
    puts "Thank you for waiting #{name}, it appears you already have a booking.\n"
    If you would like to change your booking, please select Change Booking from the main menu."
  end
end
```

```
# Booking details are stored in a new class object called booking1
booking1 = Booking.new(booking[:treatment], booking[:day], booking[:time])

if booking[:treatment] == "A Tasty Treat"
  owing << treatment1.price
elsif booking[:treatment] == "Detox Facial"
  owing << treatment2.price
elsif booking[:treatment] == "New Hair, Who Dis?"
  owing << treatment3.price
elsif booking[:treatment] == "Coder Special"
  owing << treatment4.price
elsif booking[:treatment] == "Stack Overflow Enlightenment"
  owing << treatment5.price
end
```

FINAL DESIGN

CREATE A BOOKING FEATURE

CREATE A BOOKING

Please select your treatment:

A Tasty Treat

Please select a day you would like your treatment:

Tuesday

Please select a time you would like to have your treatment:

13:00

You have now secured a booking for A Tasty Treat on Tuesday at 13:00.

OVERVIEW

CHANGE A BOOKING FEATURE

The Change a Booking Feature allows the user to change their original treatment booking

Like the Create a Booking feature, the Change a Booking menu prompts the user to:

- Select a treatment
- Select a day
- Select a time

After the user has selected all the above information, they will receive a message stating that they have secured a new booking for their chosen treatment, day and time. The user is also told that their original booking has been deleted.

CODE IMPLEMENTATION

CHANGE A BOOKING FEATURE

The code implemented in the Change a Booking Feature includes:

- Deleting the previous bookings key values from the bookings hash
- Storing each new piece of user input into the booking hash.
- A new booking class object will also be created as booking2, which utilises the values in the bookings hash

```
when "Change Booking"
  # Change booking heading
  self.clear
  change_booking_heading
  if owing != nil
    puts "You currently have a booking for #{booking[:treatment]} on #{booking[:day]} at #{booking[:time]}.\n\n"
    puts "This booking will be deleted.\n\n"

    # Items from hash are deleted
    booking.delete(:treatment)
    booking.delete(:day)
    booking.delete(:time)

    # User is prompted to reselect a new treatment
    answer = prompt.select("Please select a new treatment:\n\n", %w(A\ Tasty\ Treat Detox\ Facial New\ Hair\ Who\ Dis? Coder\ Special Stack\ Overflow\ Enlightenment))

    # New treatment is stored in the bookings hash
    booking[:treatment] = answer

    # User is prompted to reselect a new day
    answer = prompt.select("Please select a new day that you would like your treatment:\n\n", %w(Sunday Monday Tuesday Wednesday Thursday Friday Saturday))

    # New day is store in the bookings hash
    booking[:day] = answer

    # User is prompted to reselect a new time
    answer = prompt.select("Please select a new time that you would like to have your treatment:\n\n", %w(9:00 10:00 11:00 12:00 13:00 14:00 15:00 16:00 17:00))

    # New time is store in the bookings hash
    booking[:time] = answer

    # Message to user confirming the change in their booking
    puts "You have now have a new booking for #{booking[:treatment]} on #{booking[:day]} at #{booking[:time]}.\n\n"
    puts "This booking is secured, we look forward to seeing you on #{booking[:day]} at #{booking[:time]}."
```

FINAL DESIGN

CHANGE A BOOKING FEATURE

CHANGEBOOKING

You currently have a booking for Coder Special on Wednesday at 12:00.

This booking will be deleted.

Please select a new treatment:

(Press ↑/↓ arrow to move and Enter to select)

- ▶ A Tasty Treat
- Detox Facial
- New Hair, Who Dis?
- Coder Special
- Stack Overflow Enlightenment

OVERVIEW & CODE IMPLEMENTATION

DISPLAY BOOKING FEATURE

- The Display Booking Feature allows users to view their future treatments
- Display Booking feature code implementation accesses the treatment, description and price information by calling on the values stored in the bookings hash
- This information is then puts to the screen for the user to view

```
when "Display Booking"
  self.clear
  display_booking_heading
  if owing != nil
    puts "You currently have a booking for #{booking[:treatment]} on #{booking[:day]} at #{booking[:time]}.\n\n"
  else
    puts "Sorry #{name}, it appears that you do not have a booking yet. Select create a booking to change that!"
  end
```

FINAL DESIGN

DISPLAY BOOKING

CURRENT BOOKING

You currently have a booking for A Tasty Treat on Tuesday at 13:00.

What would you like to do?

(Press ↑/↓/←/→ arrow to move and Enter to select)

- Start Quiz
- Check Balance
- Treatments Available
- Create Booking
- Change Booking
- Display Booking

OVERVIEW & CODE IMPLEMENTATION

CHECK SPA POINTS FEATURE

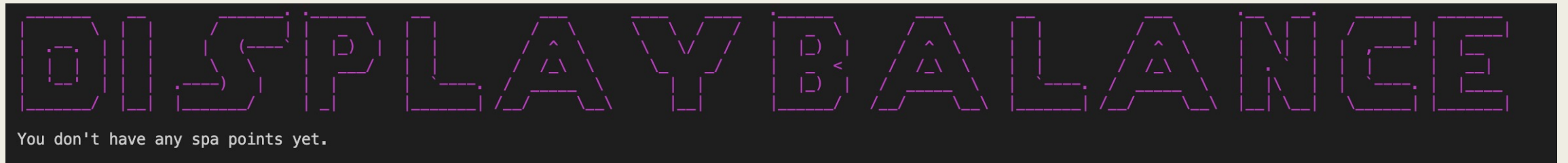
- The Check Spa Points Feature allows users to view how many spa points they have received
- The Check Spa Points Feature accesses the users spa points via the objects stored in the Spa Points Class
- This information is then puts to the screen for the user to view

```
when "Check Balance"
  loading_balance
  self_clear
  display_balance_heading
  if $wallet == nil
    puts "You don't have any spa points yet.\n\n"
  else
    puts "You currently have #{ $wallet.wallet } spa points.\n\n"
  end
```


FINAL DESIGN

CHECK SPA POINTS FEATURE

Before completing the quiz



After completing the quiz



OVERVIEW & CODE IMPLEMENTATION

CHECKOUT FEATURE

The Checkout Feature allows users to finalise payment for their booked treatment

The code implemented in this feature is:

- A transaction method that calculates the treatment price subtracted from their spa points
- If the treatment price is more than the user's spa points the application will prompt the user to either retake the quiz or select another treatment

```
when "Checkout"
  sum = buy($wallet.wallet, owing[0])
  if sum < 0
    puts "Im sorry #{name}, it appears that you do not have enough spa points for this treatment.\n\n"
    puts "Remember, practice makes perfect #{name}. You can complete the quiz again to earn more spa points."
  else
    puts "Thank you for choosing Coder Detox Spa, #{name}. We look forward to seeing you on #{booking[:day]} at #{booking[:time]}."
  end
```

DEVELOPMENT & BUILD PROCESS

APPROACHING THE TASK

Planning : -

- Working off my control flow chart to understand what classes and methods were needed to build my application

Time Management & Prioritisation: -

- A Trello board was utilised to prioritise tasks and complete application requirements

GitHub: -

- Initiating a remote repository and committing regularly

DEVELOPMENT & BUILD PROCESS

APPROACHING THE TASK - TRELLO

The screenshot shows a Trello board titled "Terminal Application" with a background image of a mountain range. The board is organized into several columns, each containing task cards. Each card has a title, a description, a size label (e.g., "size - large", "size - medium", "size - small"), a priority label (e.g., "High Priority"), a due date (e.g., "27 May", "24 May - 27 May"), and a checklist of items to be completed. The columns are: "Code Requirements", "Software Development Plan", "Slide Deck", "Completed Tasks", and "Implementation plan".

- Code Requirements**
 - size - large High Priority: Implement features in the software development plan you have designed. (Due: 27 May)
 - size - medium High Priority: Apply DRY (Don't Repeat Yourself) coding principles to all code produced. (Due: 27 May)
 - size - medium High Priority: Apply all style and conventions for the programming language consistently to all code produced. (Due: 27 May)
 - size - large High Priority: Creates an application which runs without error and has features that are consistent with the development plan. (Due: 27 May)
 - size - medium High Priority: Design TWO tests which check that the application is running as...
- Software Development Plan**
 - size - large High Priority: README Documentation (Due: 24 May - 27 May)
 - size - small High Priority: Design help documentation which includes a set of instructions which accurately describe how to use and install the application. (Due: 27 May)
- Slide Deck**
 - + Add a card
- Completed Tasks**
 - 24 May - 25 May: An overview of your code
 - 24 May - 25 May: Develop a diagram which describes the control flow of your application. (Due: 19 May)
 - size - medium: Develop an outline of the user interaction and experience for the application. (Due: 26 May)
 - size - medium: Develop a statement of purpose and scope for your application. (Due: 19 May)
 - size - medium: Develop a list of features that will be included in the application. It must include: - at least THREE features - describe each feature (Due: 18 May - 19 May)
- Implementation plan**
 - size - large High Priority: Quiz feature (Due: 19 May - 20 May, 7/7)
 - size - large High Priority: Treatments feature (Due: 19 May - 20 May, 6/6)
 - size - large High Priority: Create booking feature (Due: 20 May - 21 May, 6/6)
 - size - large High Priority: Change booking feature (Due: 20 May - 21 May, 7/7)
 - size - large High Priority: Display future bookings (Due: 20 May - 21 May, 4/4)
 - Checkout feature (Due: 22 May - 23 May, 6/6)
 - size - large High Priority: Main application file (Due: 20 May - 26 May, 4/4)

- Trello Board to keep track of tasks and software implementation
- All task were given a size label and if they were deemed a priority they were given a high priority label
- Due dates were also assigned to keep to the strict deadline
- All features had a checklist of things to complete

DEVELOPMENT & BUILD PROCESS

CHALLENGES

- Not having the experience to know how to fix a bug
- Working across local files with “require_relative”
- Amending a line of code to fix a bug, which ultimately breaks something else in the program
- Time management

DEVELOPMENT & BUILD PROCESS

FAVOURITE PARTS

- Incremental success – when you change something in your code and it finally works
– the eureka moment!
- The learning process – I have a much better understanding of Ruby after completing this project
- The satisfaction of the final product