コロナ予防アプリの制作

松田　悠斗

１．研究概要

Webアプリケーションフレームワーク「Ruby on Rails」（略称 Rails）を使用してコロナ予防促進 Web アプリを制作した。機能は、１日１回の「健康観察」とコロナ対策に効果的な「項目チェック」、集めたポイントで遊ぶ「ミニゲーム」を制作した。

２．研究の具体的内容

（１）Rails とは

プログラミング言語「Ruby」で作られた、Web アプリ開発を簡単に行えるフレームワークで、MVC モデルを基礎としている（図１）。

・M（Model）

データベースを操作する仕組み。

・V（View）

画面表示を取り扱う。

・C（Controller）

Model と View に指示を出す司令塔で、アプリの機能を制御する。

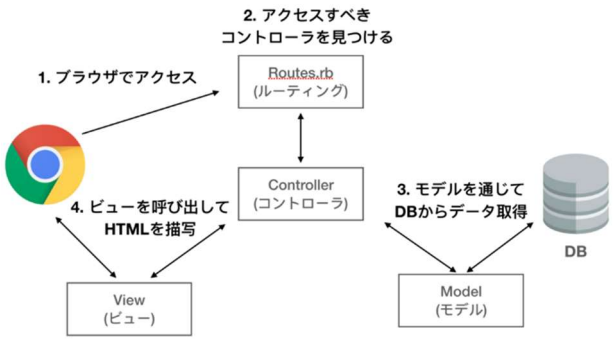

図１　MVC モデルのイメージ

（２）環境構築

・AWS Cloud9 ： ブラウザから使える IDE。

・Heroku ： Web サーバを提供する PaaS。

・GitHub ： ソフトウェアのバージョン管理。

これらのサービスのユーザ登録を行い Rails の初期設定とアプリのデプロイを行った。

（３）アプリの機能紹介

今回制作したアプリ主な機能を４つ紹介する。

（ア） 健康観察機能

１日１回体温を入力し、体調不良のチェックを行う機能である。健康観察を行うことで一定のポイントが貰える（図２，３）。

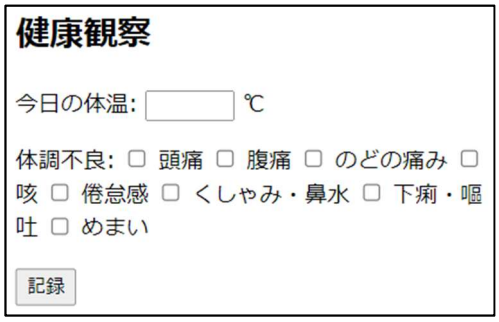

図２　アプリ画面

```
def create
  @temperature = Temperature.new(temperature_params)
  @temperature.date = Time.zone.today
  if @temperature.save
    redirect_to root_path
  else
    render 'temperatures/new'
  end
end

private
def temperature_params
  params.require(:temperature).permit(:user_id, :taion, :health)
end
```

図３　Controller（一部抜粋）

（イ） 項目チェック機能

コロナ予防に効果的な行動をチェックする機能である。具体的には、「マスクをつけて外出したか」,「手指消毒をしたか」,「３密を避けて行動したか」等の項目をクリアしたかをチェックする。達成度によって貰えるポイントが変化する（図４，５）。

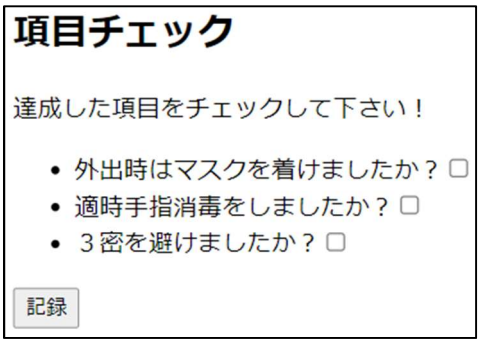

図４　アプリ画面

```ruby
def create
  @check = Check.new(check_params)
  @check.date = Time.zone.today
  if @check.save
    redirect_to root_path
  else
    render 'checks/new'
  end
end

private
def check_params
  params.require(:check).permit(:user_id, :achieve)
end
```

図５　Controller（一部抜粋）

（ウ）レポート機能

「健康観察」と「項目チェック」で入力したデータを確認する機能である。月別の一覧表のような形で確認できるほか、任意の日付の健康データを検索できる（図６，７）。

図６　アプリ画面

```erb
<h1>健康レポート</h1>
<h2 class="report-title">体温データ</h2>
<table class="report-table">
  <tr>
    <th>日付</th>
    <th>体温</th>
    <th>体調不良</th>
  </tr>
  <% @user.temperatures.each do |t| %>
    <tr>
      <td><%= t.date %></td>
      <td><%= t.taion %>℃ </td>
      <td><%= t.health %></td>
    </tr>
  <% end %>
</table>

<h2 class="report-title">チェックデータ</h2>
<table class="report-table">
  <tr>
    <th>日付</th>
    <th>達成項目</th>
  </tr>
  <% @user.checks.each do |c| %>
    <tr>
      <td><%= c.date %></td>
      <td><%= c.achieve %></td>
    </tr>
  <% end %>
</table>
```

図７　View

（エ）ミニゲーム

ポイントを使って遊ぶコレクションゲームを制作した。ガチャで入手したアイテムは図鑑に登録され、合成で新たなアイテムを作ることができる（図８，９，１０）。

図８　アプリ画面（図鑑ページ）

図９　アプリ画面（合成ページ）

```ruby
def get_gacha
  use_point = 5
  point = Profile.find_by(user_id: current_user.id).point
  if point >= use_point
    items = ItemMaster.where(gacha: true)
    random = rand(100)
    if random <= 2
      items = items.where(rare: 5)
    elsif random <=12
      items = items.where(rare: 4)
    else
      items = items.where(rare: 3)
    end
    item = items.sample
    item_save(item)
    profile = Profile.find_by(user_id: current_user.id)
    new_point = point - use_point
    profile.update(point: new_point)

    @result = ItemMaster.find(item.id).name + 'をGET！'
  else
    @result = 'ポイントが足りません！'
  end
end
```

図１０　Controller（ガチャ部分）

（４）プログラミング

（ア）データベース設計と Model の実装

まずテーブル間のリレーションやカラムを整理し、E-R 図を作成した（図１１）。

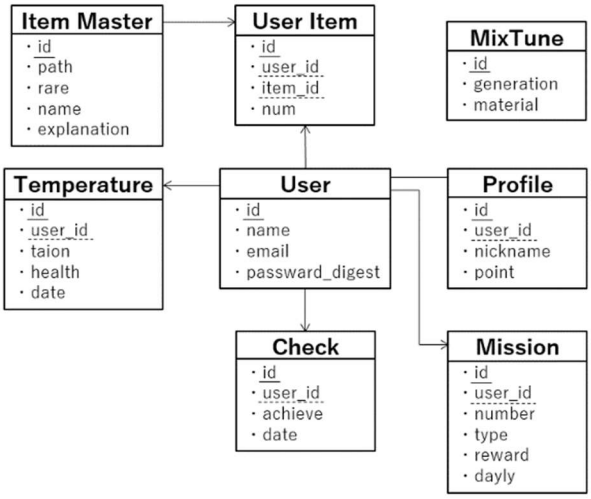

図１１　E-R 図

次に Model を作成し、ユーザの操作に応じてデータベースを扱えるようにした（図１２，１３）。Model によって、SQL を書くことなく Ruby のコードで簡単にデータベースを扱うことができる他、SQL インジェクションを意識することなく防ぐことができる。

```ruby
class User < ApplicationRecord
  validates :name, presence: true
  validates :email, presence: true, uniqueness: true

  has_secure_password

  has_many :temperatures, dependent: :destroy
  has_many :checks, dependent: :destroy
  has_many :missions, dependent: :destroy
  has_many :user_items, dependent: :destroy
  has_many :items, through: :user_items, source: :item_master
  has_one :profile, dependent: :destroy
end
```

図１２　作成したモデル（編集後）

```
10|岡山工業|okako@net|$2a$10$JzNxcyYBeAm7FSYWmY141uD0Hfse
21|テスト|test@gmail.com|$2a$10$KbGyBUhEm7pmFFmOeutZfee7r
22|熊沢蕃山|ijin-1@gmail.com|$2a$10$q1Opd/2bAqb7KyAq.GgNc
23|中江藤樹|ijin-2@gmail.com|$2a$10$O/riQBQUF17sGPSHbp6G8
24|上杉鷹山|ijin-3@gmail.com|$2a$10$/i59GynWh9gfrC5SDpEML
25|森信三|ijin-4@gmail.com|$2a$10$YSHHH3lhbB0ur8FDj9Jbbeg
26|文谷校長|top@net|$2a$10$PWoFy5Md6jU8eoszPGU6vemMZXsIWJ
```

図１３　データベース（一部抜粋）

（イ）登録フォームのバリデーション

データの整合性を保つため、JavaScript でリアルタイムに入力チェックを行うプログラムを作成した。メールアドレスの一意性チェックはサーバへのリクエストが必要なため、Ajax で非同期処理を行った（図１４）。

```javascript
$.ajax({
  type: 'GET',              //GETでリクエスト
  url: '/mails/searches',   //参照先URL
  data: {mail: mail},       //送信データ（相手で使うパラメータ: 送信データ）
  dataType: 'json'          //レスポンスされるデータ型
})
.done(function(data) {      //data : レスポンスされたデータ
  if(!data) {
    $('.js-mail-error').append('このメールアドレスは登録済みです。');
  }
})
.always(function() {
  $('.js-form-btn').prop('disabled', false);
});
```

図１４　Ajax プログラム（一部抜粋）

（ウ）バッチ処理

Heroku のアドオン Heroku Scheduler を使用して、１日１回処理を行うバッチプログラムを実行させた。バッチプログラムを作成し、Heroku Scheduler（図１５）にコマンド、実行周期、実行開始時刻を入力するだけでバッチ処理を実装することができる。なお、時刻の指定は UTC（世界標準時）で行うため、日本時間より９時間遅らせて指定した。

図１５　Heroku Scheduler

（５）スタイリング

SCSS でアプリのスタイリングを行った。また、Google Chrome の拡張機能であるデベロッパーツール(図１６)を使用することで、スタイルの適用範囲が可視化される他、スタイルを手軽に試すことができるため、開発効率が飛躍的に向上した。

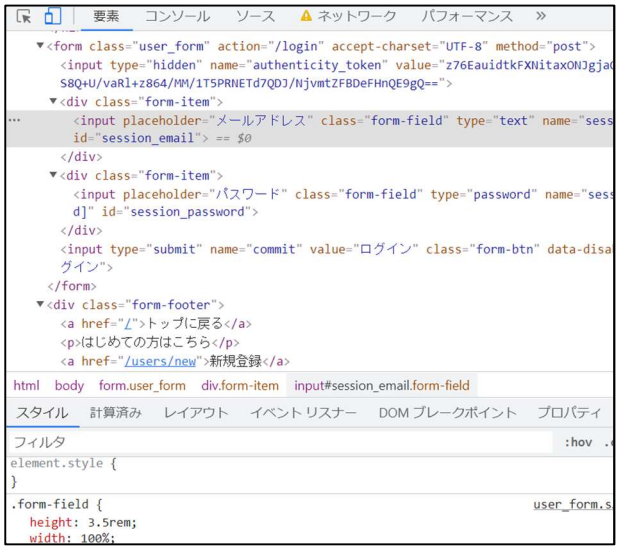

図１６　デベロッパーツール

（６）アプリの完成イメージ

最後に、アプリの QR コードと URL を紹介する（図１７）。

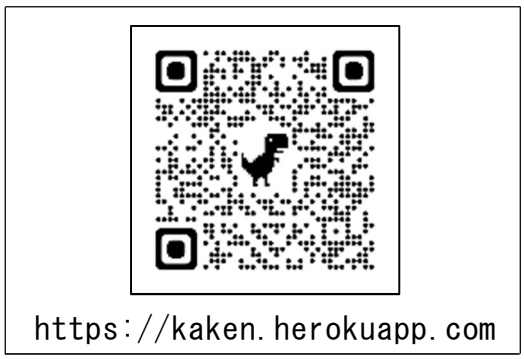

図１７　QR コードと URL

３．研究のまとめ

今回のアプリ制作を通して Rails の基礎をはじめ、アプリ開発のノウハウや Web の仕組みなどたくさんのことを学ぶことができた。特に開発では、コードの冗長性を極力なくすことでバグの抑制や保守性の向上に繋げることができた。反省点としては、設計通りの開発ができずプログラムを作り直すことがあったため、最初の設計を更にしっかりするべきだと感じた。今回の研究での経験をもとに今後は Web についての技術を高め、IoB(Internet of Behavior)を活用して技術的に人々の行動に働きかけるアプリを開発していきたい。

参考文献

・Ruby on Rails チュートリアル
https://railstutorial.jp/

・Ruby on Rails ガイド
https://railsguides.jp/

・Pikawaka（Ajax チュートリアル）
https://pikawaka.com/rails/ajax-jquery

・Qiita（【Rails】ログイン機能を実装する）
https://qiita.com/d0ne1s/items/7c4d2bef53e34a9dec7