

# Web プログラミング期末レポート

202210016 高橋伊織

## 金銭管理 Web アプリケーション Manay

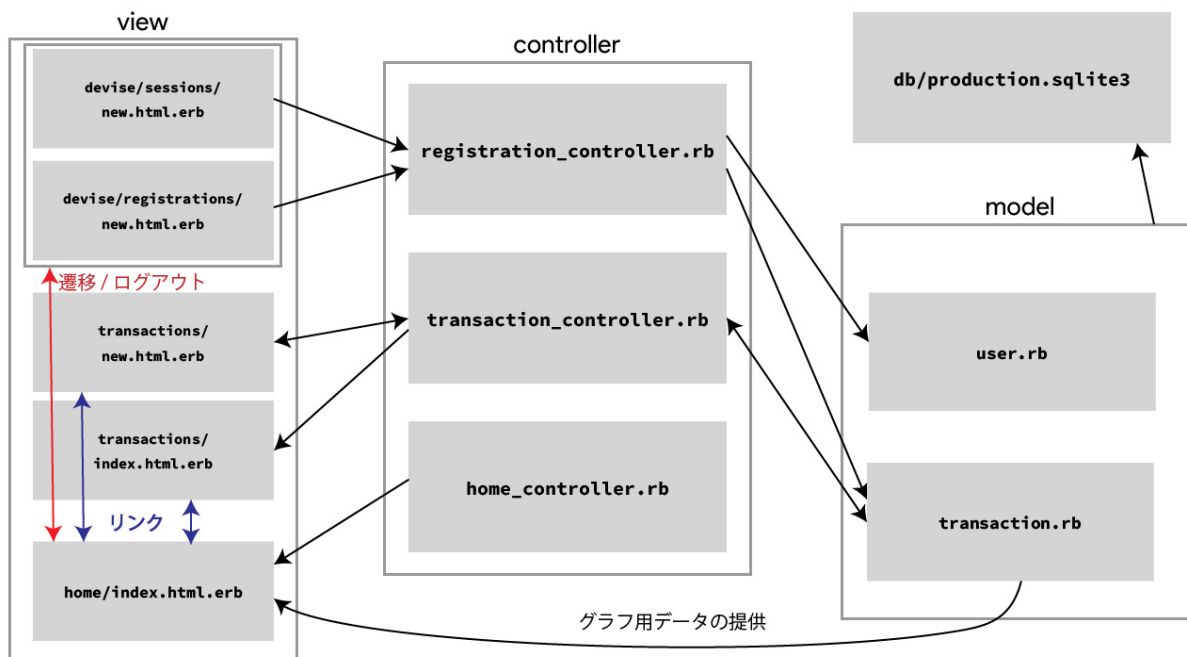
### 1 概要

Manay は、普段の金銭管理を簡単に行うことができる Web アプリである。

このアプリケーションでは、ユーザが自分の収入と支出を追跡・管理することができる。簡単な操作を通して、入出金の登録・履歴の閲覧・残高の推移の確認が行える。

このアプリケーションは、次の URL からアクセスできる。<http://20.243.83.92:3000/>

このアプリケーションは以下のような構造になっている。



### 2 特徴と工夫した点

## 2.1 フレームワーク

今回のアプリケーションでは、開発をより簡単・効率化するため、Ruby on Rails というフレームワークを使用した。MVC モデルを利用することで、頭の中を整理しながらコーディングをすることができた。

## 2.2 モデル

このアプリケーションでは、以下のようにモデルを定義した。

- User
  - email:string
  - encrypted\_password:string
  - current\_balance:decimal
- Transaction
  - user\_id:integer
  - amount:decimal
  - transaction\_type:string
  - created\_at:datetime
  - after\_balance:decimal

ここで、balance（残高）属性が User/Transaction のどちらにもあることに疑問を持つかもしれない。

これは、以下の機能を実装するためである。

- 現在の残高を表示する
- これまでの残高の推移をグラフで表示する

current\_balance のみ存在する場合は、1 ユーザにつき 1 つの残高しか持てないため、残高の推移をグラフで表示することができない。また、after\_balance のみ存在する場合は、残高を表示するために全ての Transaction を走査し、計算する必要がある。

そのため、どちらのモデルにも現在の残高を保持する実装をすることにした。以下は、`app/models/transaction.rb` のコードの一部である。

```
1 class Transaction < ApplicationRecord
2   # 省略
```

```

3   after_save :update_user_balance
4
5   private
6
7   def update_user_balance
8
9     #省略
10
11    new_balance = transaction_type == 'payment' ? user.current_balance - amount : user.current_balance + amount
12    user.update(current_balance: new_balance)
13    update_columns(after_balance: new_balance)
14  end
15
16 end

```

`after_save` コールバックを使用し、`transaction` によって増減した `balance` を、`current_balance` ・ `after_balance` の両方に格納している。これにより、上記要件が実装できた。

## 2.3 アカунト作成 / ログイン

このアプリケーションは、ユーザごとにサービスを提供する必要があった。今回は、`devise` という gem を使用した。`devise` を用いることで、ユーザ登録・認証系の機能を簡単に実装することができた。

特に、`app/controllers/application_controller.rb` に以下のコードを追加することで、アプリケーションにおける全てのアクションに対して、ログイン状態を要求することができた。

```

1  class ApplicationController < ActionController::Base
2    before_action :authenticate_user!
3  end

```

また、アカウント作成時に現在所持している金額を入力する機能を実装した。これは、以下の2点を狙ったものである。

- アカウント作成時に `after_balance` にデータを入れることで、トップページの折れ線グラフでレンダリングエラーを発生させない
- 現在の所持金を、入出金取引と同様に登録するのは直観的ではない

`current_balance` の初期登録は `app/views/devise/registrations/new.html.erb` によって実装されている。

```
1 # 省略
2 <div class="field">
3   <%= f.label :current_balance, "残高登録" %>
4   <%= f.number_field :current_balance, min: 0 %>
5 </div>
6 # 省略
```

また、`after_balance` の初期登録は `app/controllers/transactions_controller.rb` によって実装されている。

```
1 # 省略
2 def create
3   super do |user|
4     if user.persisted?
5       initial_balance = params[:user][:current_balance].to_
6         ↳ f
7       user.transactions.create(amount: initial_balance, des
8         ↳ cription: "初期登録", transaction_date: Time.current
9         ↳ , after_balance: initial_balance)
10    end
11  end
12 end
13 # 省略
```

この時、`after_balance` の登録より前に `current_balance` の登録が行われるため、例外処理を行う必要があった。以下は `app/models/transaction.rb` での例外処理の内容である。

```

1  # 省略
2  def update_user_balance
3
4      return if description == '初期登録'
5
6      new_balance = transaction_type == 'payment' ? user.current_balance - amount : user.current_balance + amount
7  # 省略

```

## 2.4 入出金処理

このアプリケーションでは、入出金処理を行うために、Transaction モデルを用いた。以下は、`app/controllers/transactions_controller.rb` のコードの一部である。

```

1  # 省略
2  def new
3      @transaction = Transaction.new
4  end
5
6  def create
7      @transaction = current_user.transactions.new(transaction_params)
8      if @transaction.save
9          redirect_to root_path
10     else
11         # 失敗時の処理
12         Rails.logger.debug @transaction.errors.full_messages
13         render :new
14     end
15 end
16
17 private
18
19 def transaction_params
20     params.require(:transaction).permit(:amount, :description)

```

```

    ↳ on, "transaction_date(1i)", "transaction_date(2i)", ↳
    ↳ "transaction_date(3i)", "transaction_date(4i)", "tra ↳
    ↳ nsaction_date(5i)", :transaction_type)
21   end
22   # 省略

```

この時、入金と支払いを正負の数で入力するシステムは、ユーザーフレンドリーではないと考えた。そのため、入金と支払いを区別するために、`transaction_type` 属性を用いた。この属性では、`payment` と `deposit` の2つの値を取る。erb ファイルでは、チェックボックスで選択してもらい、`after_balance` の計算は以下のように場合わけを行なった。

```

1   # 省略
2   new_balance = transaction_type == 'payment' ? user.current_b ↳
    ↳ alance - amount : user.current_balance + amount
3   # 省略

```

`app/views/transactions/new.html.erb` で実装した画面が以下である。

取引タイプ

☐ 入金 ☒ 支払い

金額

10000

説明

お年玉

取引日

2024

January

1

00

:

05

追加

## 2.5 残高推移グラフ

このアプリケーションでは、残高の推移をグラフで表示する機能を実装した。以下は、`app/controllers/home_controller.rb` のコードの一部である。

```
1  # 省略
2  <div id="balance_chart" style="width: 70dvw; height: 50dvh" <
  <h3></h3></div>
3  <script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
4  <script type="text/javascript">
5    google.charts.load('current', {'packages': ['corechart']});
6    google.charts.setOnLoadCallback(drawChart);
7  </script>
```

```

8     function drawChart() {
9         var data = google.visualization.arrayToDataTable([
10            ['日付', '残高'],
11            <% current_user.transactions.each do |transaction| ↵
12                ↳ %>
13                ['<%= transaction.created_at.strftime("%Y-%m-%d") ↵
14                ↳ %>', <%= transaction.after_balance %>],
15            <% end %>
16        ]);
17
18        var options = {
19            title: '残高の推移',
20            pointSize:8,
21            legend: { position: 'bottom' }
22        };
23
24        var chart = new google.visualization.LineChart(docume ↵
25        ↳ nt.getElementById('balance_chart'));
26
27        chart.draw(data, options)
28    }
29 </script>
30 # 省略

```

これによって、グラフを表示することができた。実際の例は以下の通りである。



ようこそ、hoge@fugaさん！

現在の残高: 193,900 円



## 2.6 使用ライブラリ

- devise (4.9.3)
- unicorn (6.1.0)
- Google Charts

## 3 感想

本授業は、非常に有意義な授業だったと感じている。Web 開発は多少ながら経験があったものの、バックエンドについてはほとんど触ったことがなかった。本授業を通して、バックエンドやデータベースとの連携について実践的に学ぶことができた。私は今まで

Ruby を扱ったことがなく、この授業がなければおそらくこの先も触ることがなかったため、貴重な機会だった。特に Ruby の「全てはオブジェクトである」という考え方が面白いと感じた。

## 4 参考文献

- Rails ガイド Active Record コールバック ([https://railsguides.jp/active\\_record\\_callbacks.html](https://railsguides.jp/active_record_callbacks.html))
- rails devise 完全入門！結局 devise って何ができるの？ (<https://www.sejuku.net/blog/13378>)
- devise のユーザー登録機能を拡張する (<https://qiita.com/krile136/items/e04181bb9e9dc0f37a79>)
- Rails devise の導入と View + Controller のカスタマイズ (sign\_in/out 後の path 指定など) (<https://qiita.com/mmmasuke/items/7d5c47b4a40f6912adf2>)
- devise を使ってユーザー管理機能を実装 (<https://qiita.com/mr0216/items/54bc1c5f9e673c8ea8>)
- 【Google Charts】 マーカー付き折れ線グラフ ([https://gldegg.com/google-charts\\_-line\\_pointsize/](https://gldegg.com/google-charts_-line_pointsize/))
- [Rails]CSS ファイルの適用について (<https://qiita.com/john-tarou/items/2eb178db48b3764cb8>)
- Rails アプリのデプロイ方法について (手動編) (<https://qiita.com/Tatsu88/items/7030fd72d0-ba714917fe>)