レポート提出票

科目名: 情報工学実験2

実験テーマ: 実験テーマ2 Webアプリケーション

実施日: 2020年 10月 5,12日

学籍番号: 4619055

氏名: 辰川力駆

共同実験者:

1　実験の要旨

Web アプリケーションとは、インターネットなどのネットワークを介して、利用者に対して様々なサービスを提供するアプリケーションのことである。サービスの要求に対して Web サーバで処理した結果をクライアント側の Web ブラウザに表示するため、アプリケーションをインストールする必要はなく、クライアント側の環境に頼らずにクロスプラットフォームに対応しているという利点がある。

2　実験の目的

本実験では、Web アプリケーションの開発を通じて、Web アプリケーションの仕組み、オブジェクト指向によるソフトウェア開発の考え方、データベース管理を理解することを目的とする。

3　実験の原理

3.1　Web アプリケーションの仕組み

Web アプリケーションでは、アプリケーションに関する処理はほぼすべてサーバで行われる。このため、アプリケーションに新たな機能が追加された場合や、新たに別のアプリケーションのサービスを受ける場合などにおいても、サービスを提供する側のサーバソフトウェアを変更するだけで良く、利用者側は何も変更することなく新たなサービスが受けられる。

サーバはクライアントからの要求に従い、HTML や画像データなどを送信し、Web ブラウザはそのデータを表示する。サーバは、クライアントからのリクエストを待ち受けており、クライアントからのリクエストが到着すると処理を行い、レスポンスを返す。また、サーバはクライアントからのリクエストを解釈し、クライアントはサーバの応答を解釈する必要がある。この要求／応答のやり取りの方式や解釈の方式の取り決めを HTTP と呼ぶ。以下に処理の概要を示す。

1. クライアントからサーバに対して接続を行う。すなわち、TCP によるコネクションの確立が行われ、双方のコンピュータ同士で通信が可能となる。

2. クライアントからサーバに対し、「指定した URL の内容を送れ」という意味の要求が送られる。URI とは、インターネット上のリソースを示す記述法である。

3. サーバは要求された URI に対するデータや、サーバプログラムによって処理された結果をクライアントに対して送信する。クライアントである Web ブラウザでは、返ってきたデータを画面に表示する。

Web アプリケーションの機能を提供するサーバを Web アプリケーションサーバと呼ぶ。Web アプリケーションサーバは、その上で複数のサーバプログラムを動作させ、様々なサービスを提供する。Web アプリケーションサーバは、通常の Web サーバと同じ HTTP を用いてブラウザと通信を行うため、Web アプリケーションは通常の WWW のコンテンツの閲覧と同様にブラウザを用いて利用することができる。

3.2 URL の構成

URL の構造は主に 4 つに分けられる。例えば URL が、

http://www.riku.co.jp/hoge/tatsukawa.html

の場合について下記に示す。

1. http: という部分がプロトコル名である。これはインターネットでホームページにアクセスする際に必要な記号である。SSL 暗号化通信でアクセスする際には、http の後ろに「s」が付く。

2. www.riku.co.jp という部分がホスト名+ドメイン名である。アクセスするサーバを指定するために必要である。ホスト名とは、ネットワーク上のコンピューターにつける識別用の文字列のことで、一般的にホームページを管理するコンピューターには「www」が用いられる。ドメイン名とは、インターネット上のネットワークを特定するための文字列のことで、他のホームページと重複することはない。

3. hoge という部分がディレクトリ名である。サーバ内のフォルダ名とそのフォルダの位置を表す文字列のことである。

4. tatsukawa.html という部分がファイル名である。

   今回の場合は hoge フォルダの中の tatsukawa.html を用いていることを示している。

3.3 HTTP リクエスト/レスポンス

HTTP とは、Web サーバと Web クライアントの間でデータの送受信を行うために用いられるプロトコルである。HTTP のメソッドについては、その用途によって GET と POST で使い分ける。

GET メソッド

HTTP 通信でサーバから情報を取得する時に使用する。GET でのサーバへのデータの送信方法として、リクエスト URL の後に?に続いて情報を送信する方式であるクエリストリングがある。URL の末尾に「?」マークを付け、続けて「名前=値」の形式で記述する。値が複数あるときは「&」で区切り記述する。

```
http://example.com/index.html?name1=value1&name2=value2
```

このように送信するデータがアドレスバーに表示されてしまうため、他人に見られる可能性があるので、他人に見られたくない情報は GET では送らない。

POST メソッド

HTTP 通信で、サーバへ情報を登録する時に使用する。データ量が多い場合、バイナリデータを送信したい場合、他の人に見られたくない情報を送る場合に利用する。POST でサーバにデータを送信する際は、メッセージボディにデータが記述される。

以下に、POST でのサーバからクライアントへのリクエスト例を示す。

```
POST /test/ HTTP/1.1
Host: localhost
Content-Length: 25
Content-Type: application/x-www-form-urlencoded

name1=value1&name2=value2
```

上記のように、リクエストヘッダの後に一行、空行が入り、その後 POST で送信したクエリストリングが、リクエストボディとしてクライアントからサーバへと送信される。リクエストボディの長さは、「Content-Length:」という項目で表される。

```
HTTP/1.1 200 OK
Content-Type: text/html
Date: Mon, 3 Sep 2017 00:00:00 GMT
Last-Modified: Fri, 1 Sep 2017 00:00:00 GMT
Accept-Ranges: bytes
ETag: "011be787c2f41"
Server: Apache/2.4.27(Unix)

<!DOCTYPE html>
<html>
<head>
<metacharset="utf-8">
<title>Sample</title>
</head>
<body>
<p>Hello!</p>
</body>
</html>
```

レスポンスメッセージのメッセージヘッダの１行目はステータスラインであり、HTTP のバージョン、ステータスコードを表している。ステータスコードを以下の表に示す。

表 1: ステータスコード

ステータスコード	内容
1xx	情報
2xx	成功
3xx	リダイレクション
4xx	クライアントエラー
5xx	サーバエラー

残りのヘッダ部分にはリクエスト先のウェブサーバの情報等が記述されている。メッセージボディ部分が、クライアント側で表示される内容となっている。

3.4 SQL

JIS では、「複数の独立した利用者に対して、要求に応じてデータを受け入れ、格納し、供給するためのデータ構造」と定義されている。この「複数の独立した利用者に対して、データを受け入れ、格納し、供給する」を行うシステムをデータベース管理システム (DBMS) と呼ぶ。大量のデータを扱う際に、高速に効率よくデータを操作 (追加、更新、削除) したり、検索したりすることができる。

現在利用されている多くのデータベース管理システムはリレーショナルデータベース管理システム (RDBMS) である。リレーショナルデータベースは１つ以上のテーブルで構成され、テーブルは１つ以上の項目 (列) で構成される。各項目についてデータ型を定義する。データは行単位で扱う。

SQL は、リレーショナルデータベースの定義・操作・検索などをするためにデータベース言語であり、多くの RDBMS がこの言語に対応している。下記に本実験で使用する SQL の操作一覧を示す。

表 2: SQL 操作一覧

操作	SQL 文
テーブルの作成	**CREATE TABLE** テーブル名 (列名 データ型, 列名 データ型, ...)
データの検索	**SELECT** 列名 **FROM** テーブル名 **WHERE** 条件式;
データの追加	**INSERT INTO** テーブル名 (列名, 列名...) **VALUES** (値, 値, ...);
データの更新	**UPDATE** テーブル名 **SET** 列名 = 値 **WHERE** 条件式;
データの削除	**DELETE FROM** テーブル名 **WHERE** 条件式;

3.5 MVC モデルの仕組み

MVC は、ユーザインターフェースをもつアプリケーションを実装するためのデザインパターンである。アプリケーションの内部データをユーザが直接参照・編集する情報から分離する。そのためにアプリケーションを以下の 3 つの部分に分割する。

Model アプリケーションデータ、ビジネスロジック、アプリケーションが扱う領域のデータと手続きを表現する要素である。ここで、ビジネスロジックとは、ビジネスオブジェクトをモデル化したデータベース上のデータに対する処理手順を指す。

View グラフや図などの任意の情報表現モデルのデータを取り出してユーザが見るのに適した形で表示する要素である。すなわち UI への出力を担当する。Web アプリケーションでは HTML 文書を生成して動的にデータを表示するための部分にあたる。

Controller 入力を受け取り model と view への命令に変換するユーザからの入力をモデルへのメッセージへと変換してモデルに伝える要素である。すなわち UI からの入力を担当する。モデルに変更を引き起こす場合もあるが、直接に描画を行ったり、モデルの内部データを直接操作したりはしない。

4　実験装置

- MacBook Pro(16-inch,2019)
  - ProductName: Mac OS X
  - ProductVersion: 10.15.5
  - BuildVersion: 19F101
  - プロセッサ: 2.6 GHz 6 コア Intel Core i7
  - メモリ: 16 GB 2667 MHz DDR4
- ブラウザ : Google Chrome 86.0.4240.80

5　実験結果・考察

5.1　課題 1

Slim framework と Twig を利用して，以下に示す要件を満たす消費税を自動計算する Web アプリケーションを作成する。以下に示す２つのファイルを作成する。

- index.php
- templates/tax.html

動作

1. 金額入力画面を表示 (結果の表は表示しない)
2. 金額と消費税の有無の入力
3. 金額と消費税の有無を判定し、判定結果を表として表示する

ルーティング
GET /tax 金額入力画面表示
POST /tax 消費税計算処理、および、結果表示

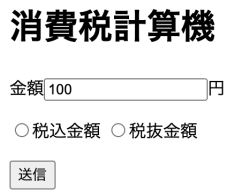

図 1: 課題 1 の GET の表示画面

図 1 は初期画面である。tax.html の 15 行目,17 行目で input type=radio にすることにより、ラジオボダンを実装している。

図 2: 2000 円の税抜金額を入力

図 2 は、税抜金額 2000 円を入力した場合の結果を示す。index.php の 25 行目から 28 行目では、税抜金額が入力されたならば (data['type'] が'extax' ならば) 実行をしている。data['price'] で来たものを data['extax'] として消費税を計算している。

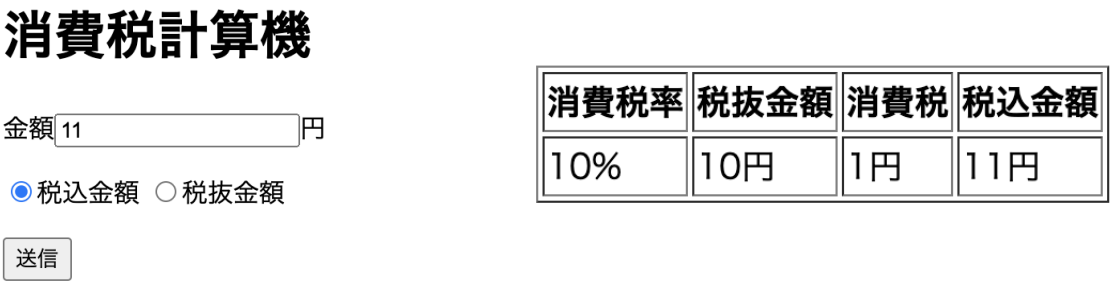

図 3: 11 円の税込金額を入力

図 3 は、税込金額 11 円を入力した場合の結果を示す。index.php の 21 行目から 24 行目では、税込金額が入力されたならば (data['type'] が'intax' ならば) 実行をしている。data['price'] で来たものを data['intax'] として消費税を計算している。

また、GET と POST で結果を表示させるかどうかを判定するのは、tax.html の 26 行目で taxrate が存在 ($\neq 0$) すれば、表を出力している。

5.2 課題 2

以下に示す要件を満たす自動販売機の Web アプリケーションを作成する。アプリケーションの作成にあたり Slim と Twig を利用し、以下に示す 2 つのファイルを作成する。

- index2.php
- templates/machine.html

- データベース (**Model**)
  列名は，“ID，商品名，価格，数量”とする．
  データベースファイル名は、“vending.db”とする
- 動作
  1. 商品一覧を表示
  2. 投入金額、商品の購入個数の入力
  3. 購入金額と投入金額を判定し、判定結果に応じた処理を行う
     - (購入可能)
       (a) データベースを更新 (b) 結果表示領域にレシート (購入商品、合計金額、おつり) を表示
     - (購入不可) 結果表示領域に「投入金額不足」を表示

  ルーティング

  GET /vending 商品一覧表示

  POST /vending 商品購入処理、および、結果表示

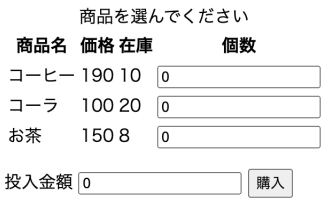

図 4: 課題 2 の GET の表示画面

図 4 は初期画面である。初期の machine.html とほぼ変わりはない。ただし、細かいことだが投入額を投入金額に変えている。

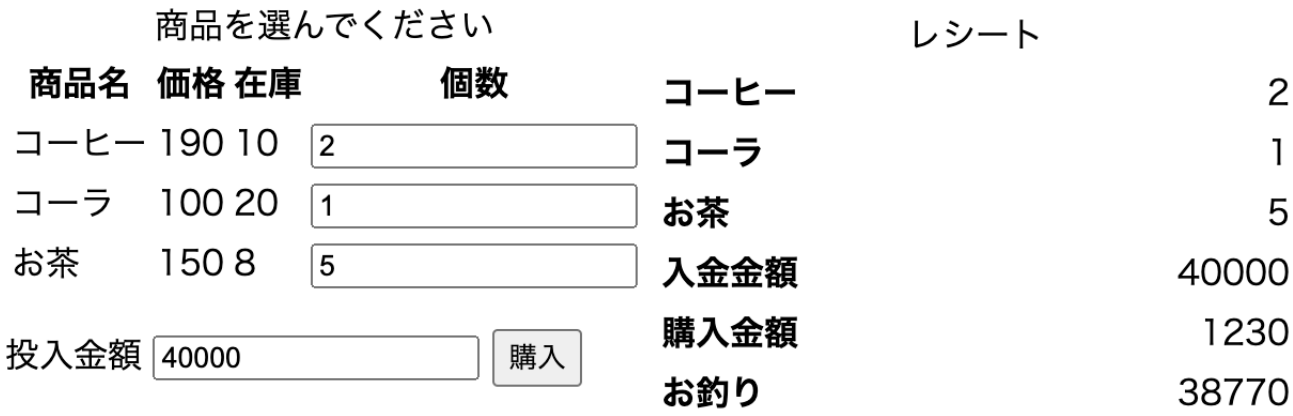

図 5: 正しい入力がされた場合の結果

図 5 は正しい入力がされた場合の結果を表示している。index.php の 78 行目でエラーが 1 つもない場合は、79 行目で data['result'] を'Finish' にして、80 行目以降でデータベースを更新している。

data['result'] を'Finish' にすることにより、machine.html の 37 行目の if 文で'Finish' ならばレシートを表示している。

図 6 以降は、エラー処理についてまとめたものである。また、エラーの出力は machine.html の 63 行目で data['result'] が'Error' ならば、for 文でエラーを一つ一つ表示している。

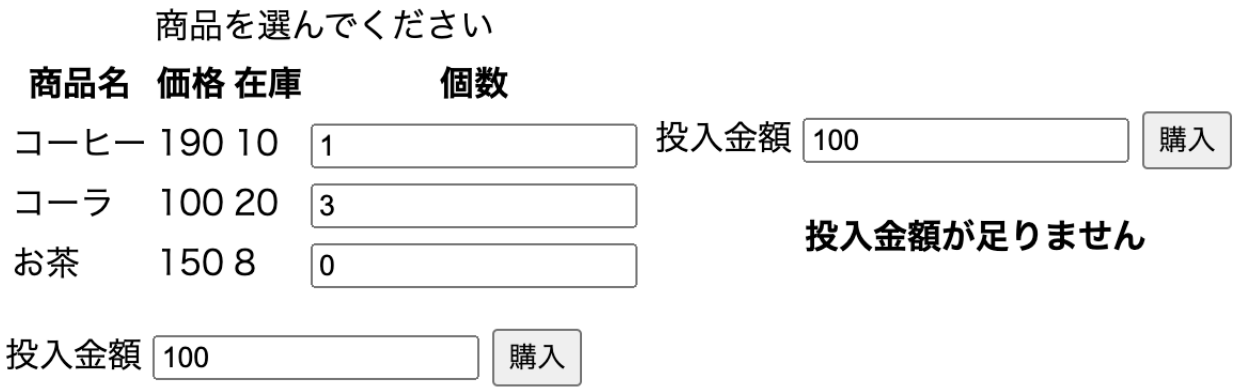

図 6: 投入金額が足りなかった場合の結果

図 6 は、投入金額が足りなかった場合の結果を表示している。課題 2 の index.php の 72 行目から 76 行目で実装している。入力された金額 – 支払総額 が負の場合はエラー処理をしている。

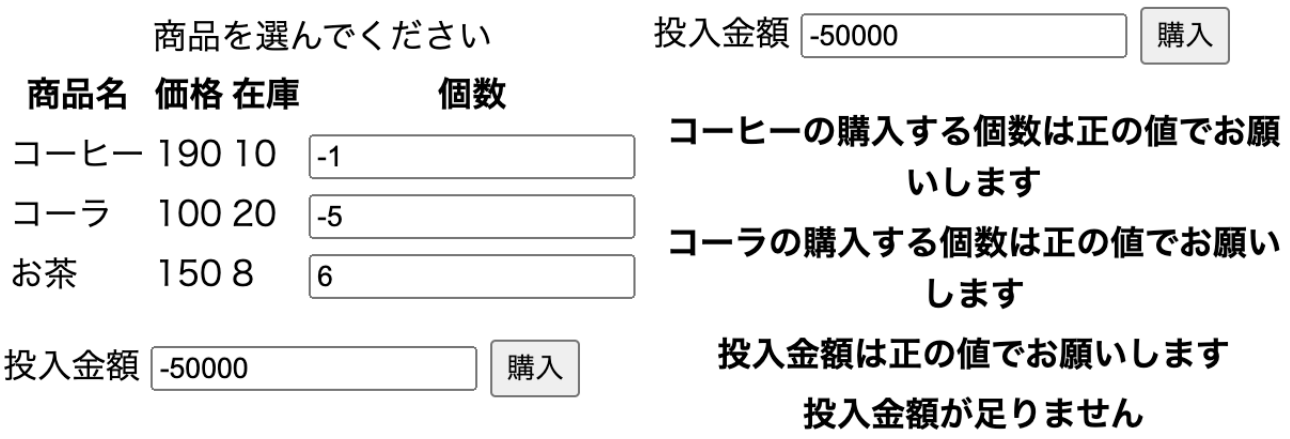

図 7: 負の値が入力された場合の結果

図 7 は、負の値が入力された場合の結果を表示している。課題 2 の index.php の 56 行目から 60 行目では、購入する個数が負の値の場合を実装している。支払う金額が負の値の場合の実装は 64 行目から 68 行目である。どちらも、0 より少なければエラー処理を施しただけである。

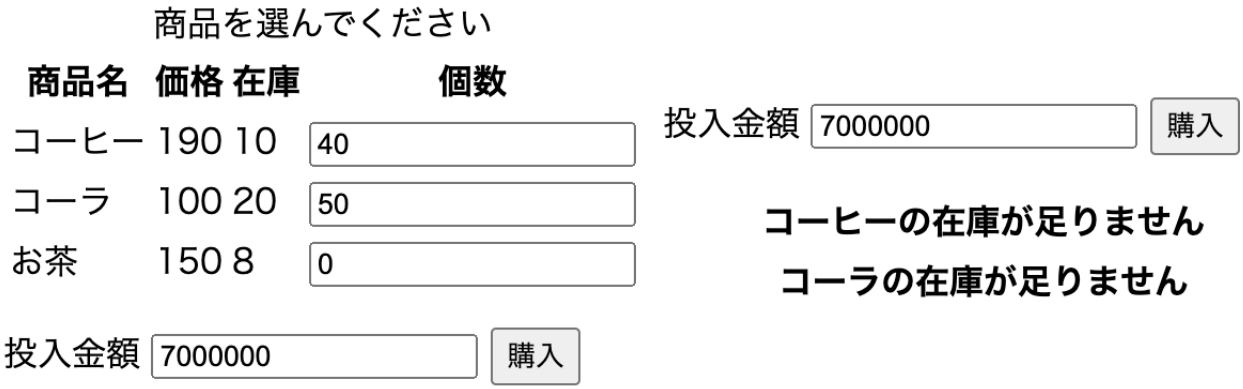

図 8: 在庫が足りない場合の結果

図 8 は、在庫が足りない場合の結果を表示している。課題 2 の index.php の 51 行目から 55 行目で実装している。入力された個数 > 在庫 ならばエラー処理をしている。

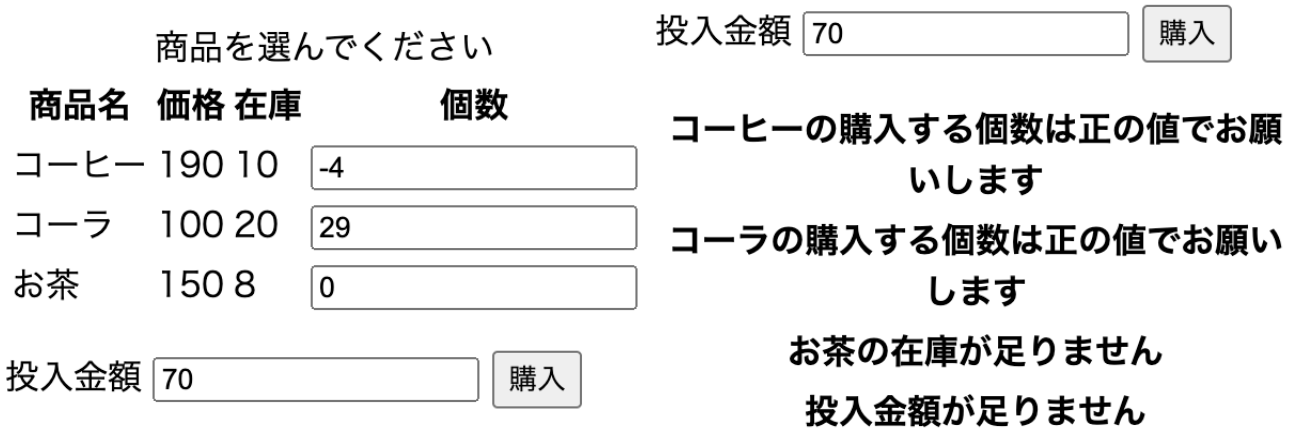

図 9: 複数のエラーが出力された場合の結果

図 9 は、複数のエラーが出力された場合の結果を表示している。これは、いままでのエラーハンドリングの実装のまとめである。複数のエラーを出力できているのは、index.php の 47 行目で用意したエラーの数を数える変数 data['count'] を用意することで、エラーが発見される度に data['errorlog'][data['count']] にエラーを入れて count をプラス 1 しているからである。

5.3 フレームワークのメリット・デメリット

メリット

機能やデザインのカスタマイズが容易にできる。また、短時間で高品質なアプリケーションを開発することが可能である。さらに、コードの統一性が生まれ、コードが見やすくなる。

デメリット

言語の勉強以外に、その言語のフレームワークも勉強しないといけない。また、費用がかかるものも存在する。

フレームワークそれぞれにもメリット・デメリットが存在する

自分が知っているもので例をあげると、Go 言語のフレームワークには有名なもので Echo,Gin,Beego などがあるが、Beego は簡単に Web アプリ開発ができ初学者におすすめで、Gin は歴史が古く軽量であり、Echo は REST の原則に則っているなどの個別の特徴があります。

6 まとめ

今実験では、html,php,SQLite を用いて Web アプリケーション開発を行った。Web アプリケーションの開発を通じて、Web アプリケーションの仕組み、オブジェクト指向によるソフトウェア開発の考え方、データベース管理を理解することができた。

参考文献

[1] 東京理科大学工学部情報工学科 情報工学実験 2 2020 年度東京理科大学工学部情報工学科出版

[2] 分かりやすい URL 構造を設計しよう「ディレクトリ名・ファイル名の付け方」：ビジネスと IT 活用に役立つ情報

https://www.asobou.co.jp/blog/web/url-optimisation

最終閲覧日 2020/10/18

A 付録

ソースコード 1: tax.html

```
<!DOCTYPE html>
<html lang="ja">

<head>
    <meta charset="utf-8">
    <title>消費税計算機</title>
</head>

<body>
    <h1>消費税計算機</h1>
    <form action="" method="POST">
        <p><label>金額<input type="number" name="price" value={{price}}>円
        </label></p>
        <p>
            <label><input type="radio" name="type" value="intax">税込金額
            </label>
            <label><input type="radio" name="type" value="extax">税抜金額
            </label>
        </p>

        <p><input type="submit" value="送信"></p>
    </form>


    <!--$data['taxrate']=10(%)が存在するなら表を表示-->
    {% if taxrate %}
    <table border="1">
        <tr>
            <th>消費税率</th>
            <th>税抜金額</th>
            <th>消費税</th>
            <th>税込金額</th>
        </tr>
        <tr>
            <td>{{taxrate}}%</td>
            <td>{{extax}}円</td>
            <td>{{tax}}円</td>
            <td>{{intax}}円</td>
        </tr>
    </table>
    {% endif %}


</body>

</html>
```

ソースコード 2: 課題 1 の index.php

```php
<?php
require 'vendor/autoload.php';

$app = new \Slim\App();

$loader = new Twig_Loader_Filesystem('templates');
$twig = new Twig_Environment($loader);

$app->get('/tax', function ($request, $response, $args) use ($twig) {
    $data = [];
    $data['price'] = 100;
    $html = $twig->render('tax.html', $data);
    return $response->write($html);
});

//POST "/tax" 税金を計算
$app->post('/tax', function ($request, $response, $args) use ($twig) {
    $data = $request->getParsedBody();
    $data['rate'] = 0.10;

    if ($data['type'] == 'intax') { //入力されたものが税込金額のとき
        $data['intax'] = $data['price'];
        $data['extax'] = floor($data['intax'] / (1 + $data['rate']));
        $data['tax'] = $data['intax'] - $data['extax'];
    } elseif ($data['type'] == 'extax') { //入力されたものが税抜金額のとき
        $data['extax'] = $data['price'];
        $data['intax'] = floor($data['price'] * (1 + $data['rate']));
        $data['tax'] = $data['intax'] - $data['extax'];
    } else { //エラー
        echo "Error...";
    }

    $data['taxrate'] = $data['rate'] * 100;

    $html = $twig->render('tax.html', $data);
    return $response->write($html);
});

$app->run();
```

ソースコード 3: machine.html

```
<!DOCTYPE html>
<html lang="ja">

<head>
    <meta charset="utf-8">
    <title>自動販売機</title>
</head>

<body>
    <form action="" method="POST">
        <table>
            <caption>商品を選んでください</caption>
            <tr>
                <th>商品名</th>
                <th>価格</th>
                <th>在庫</th>
                <th>個数</th>
            </tr>
            {% for item in items %}
            <tr>
                <td>{{item.name}}</td>
                <td>{{item.price}}</td>
                <td>{{item.num}}</td>
                <td><input type="number" name="buyitem[{{item.id}}]" value
                    ="0"></td>
            </tr>
            {% endfor %}
        </table>
        <p>
            <label>
                投入金額
                <input type="number" name="pay" value="{{pay}}">
                <input type="submit" value="購入">
            </label>
        </p>
    </form>

    {% if result=='Finish' %}
    <table width="300">
        <caption>レシート</caption>
        {% for item in items %}
        {% if buyitem[item.id] != 0 %}
        <tr>
            <th align="left">{{item.name}}</th>
            <td align="right">{{buyitem[item.id]}}</td>
        </tr>
        {% endif %}
        {% endfor %}
        <tr>
            <th align="left">入金金額</th>
            <td align="right">{{pay}}</td>
```

```
51              </tr>
52              <tr>
53                  <th align="left">購入金額</th>
54                  <td align="right">{{sum}}</td>
55              </tr>
56              <tr>
57                  <th align="left">お釣り</th>
58                  <td align="right">{{change}}</td>
59              </tr>
60          </table>
61          {% endif %}
62
63          {% if result==’Error’ %}
64          <table width="300">
65              {% for log in errorlog %}
66              <tr>
67                  <th>{{log}}</th> <!--エラーを出力-->
68              </tr>
69                {% endfor %}
70          </table>
71          {% endif %}
72      </body>
73
74      </html>
```

ソースコード 4: 課題 2 の index.php

```
1   <?php
2   require 'vendor/autoload.php';
3
4   // データベース接続
5   try {
6       $pdo = new PDO('sqlite:vending.db');
7   } catch (PDOException $e) {
8       die("データベース接続失敗" . $e->getMessage());
9   }
10  // テーブル作成
11  $pdo->exec("create table if not exists items (
12      id    integer primary key autoincrement,
13      name  varchar,
14      price integer,
15      num   integer)");
16
17  // SLIM framework の準備
18  $app = new \Slim\App();
19
20  // twig の準備
21  $loader = new Twig_Loader_Filesystem('templates');
22  $twig = new Twig_Environment($loader);
23
24  //---------------------------------
25  // ルーティング
26  //---------------------------------
27  $app->get('/vending', function ($request, $response, $args) use ($twig, $pdo) {
28      $data = [];
29      $data['pay'] = 0;
30
31      $stmt = $pdo->prepare("SELECT * FROM items");
32      $stmt->execute();
33      $data['items'] = $stmt->fetchALL();
34
35      $html = $twig->render('machine.html', $data);
36      return $response->write($html);
37  });
38
39  $app->post('/vending', function ($request, $response, $args) use ($twig, $pdo) {
40      $data = $request->getParsedBody();
41
42      $stmt = $pdo->prepare("SELECT * FROM items");
43      $stmt->execute();
44      $data['items'] = $stmt->fetchALL();
45
46      $data['sum'] = 0; //購入する商品の合計金額
47      $data['count'] = 0; //エラーの数を数える
48
49
50      foreach ($data['items'] as $item) { //items 内の item それぞれに対して実行
51          if ($data['buyitem'][$item['id']] > $item['num']) { //在庫が足りない場合
```

```
52                  $data['result'] = 'Error';
53                  $data['errorlog'][$data['count']] = sprintf("%s の在庫が足りません",
                        $item['name']);
54                  $data['count']++;
55              }
56              if ($data['buyitem'][$item['id']] < 0) { //購入する個数が負の値の場合
57                  $data['result'] = 'Error';
58                  $data['errorlog'][$data['count']] = sprintf("%
                        s の購入する個数は正の値でお願いします", $item['name']);
59                  $data['count']++;
60              }
61
62              $data['sum'] += $item['price'] * $data['buyitem'][$item['id']];
63          }
64          if ($data['pay'] < 0) { //投入金額が負の値の場合
65              $data['result'] = 'Error';
66              $data['errorlog'][$data['count']] = sprintf("投入金額は正の値でお願いしま
                    す");
67              $data['count']++;
68          }
69
70          $data['change'] = $data['pay'] - $data['sum']; //お釣りを計算
71
72          if ($data['change'] < 0) { //お金が足りていない場合
73              $data['result'] = 'Error';
74              $data['errorlog'][$data['count']] = sprintf("投入金額が足りません");
75              $data['count']++;
76          }
77
78          if ($data['count'] == 0) { //エラーが
                1つもない場合はsql のデータベースを update する
79              $data['result'] = 'Finish';
80              foreach ($data['buyitem'] as $key => $num) {
81                  $sql = "UPDATE␣items␣SET␣num␣=␣num␣-␣:buy␣WHERE␣id␣=␣:idkey";
82                  $stmt = $pdo->prepare($sql);
83                  $params = array(':buy' => $num, ':idkey' => $key);
84                  $stmt->execute($params);
85              }
86
87              $stmt = $pdo->prepare("SELECT␣*␣FROM␣items");
88              $stmt->execute();
89              $data['items'] = $stmt->fetchALL();
90          }
91
92          $html = $twig->render('machine.html', $data);
93          return $response->write($html);
94      });
95
96
97      $app->get('/insert/{name}/{num}', function ($request, $response, $args) use ($twig,
            $pdo) {
98          $stmt = $pdo->prepare("insert␣into␣items␣(name,␣price,␣num)␣values␣
```

```
            (?,?,?)");
99          $stmt->execute([$args['name'], 0, $args['num']]);
100
101         return $response->write('新商品' . $args['name'] . 'を' . $args['num'] . '個追加しました');
102     });
103
104     $app->run();
```