7-1 システム自作

バージョン	1.5
作成日	2022/01/26
作成者	k.eguchi
更新日	2023/04/23
更新者	k.eguchi

◆ 改訂履歴

No	バージョン	改訂日	改訂者	備考
1	1.1	2022/01/26	k.eguchi	新規作成
2	1.2	2022/05/12	r.shimada	設計書詳細追加
3	1.3	2022/08/31	k.eguchi	設計書サンプル追加、提出物関連追加
4	1.4	2022/10/20	k.eguchi	課題条件追加修正
5	1.5	2023/04/23	k.eguchi	課題フローを変更
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

目次

◆イントロダクション

○課題について

今までの総復習として、Laravelを使用したWebシステムの自作を行います。
過去の内容を振り返りながら、自身で設計･製造を進めていきましょう。

また、このセクションでは現場において必須知識となる**Git**, **Docker**も使用しつつ進めていきます。

この章での目標は以下の3点です。

・提示された要件※を満たすシステム構成を考え、設計する
(※要件書をお渡しします)

・設計の内容に従ってシステムを製造する

・DockerやGitに慣れる

◆課題の進め方

○課題条件

1. 要件定義書に従ったシステムを実装すること

標題の通りです。
現場では要望を受けてシステムを作成します。
　要件定義書からどのようなテーブルやデータが必要かを考え、設計を行ってください。

2. LaravelのResourceControllerについて調べて使用すること

　LaravelにはCRUD処理を簡単に行えるようにする「リソースコントローラー」という機能が存在します。
Laravelを使用する多くの現場でほぼ必ずと言っていいほど使用されているメジャーな機能になります。
公式のマニュアルは以下のURLです。
　https://readouble.com/laravel/6.x/ja/controllers.html
うまく活用して効率的に製造を進めてください。

3. Bootstrap等を活用しレイアウトを工夫すること(Laravel課題のレイアウト転用は不可)

　標題の通りです。(→Bootstrap公式レイアウト集: https://themes.getbootstrap.com/)
レイアウトはユーザーの使いやすさにも大きく影響します。
全てにおいて言えることですが、「自分がこのシステムのユーザーだったら」という視点を持って、
使いやすい・見やすいレイアウトを目指して製造してください。
尚、見た目にこだわりすぎて機能が間に合わない、ということは無いよう気を付けてください。

◆課題の進め方

○課題開始：要件定義書

課題スタート時に要件定義書をお渡しします。

要件定義書には「システムとしての目的や実装する大まかな機能」が示されたものです。
次の設計のフェーズでこの大まかな機能をシステムとしてどのように実装するかを細かく決めていきます。

○課題中：納期について ※重要

設計 ･･･ 6章完了後、**1週間以内**に提出。
レビュー、修正指示が出た場合は**5**日以内を目処に再提出してください。
添付の資料(画面設計書, テーブル定義のサンプル)や以降の課題参考，6-Cの設計資料を活用して進めてください。
設定テーマと似たサイトを参考に進めてみましょう。

製造 ･･･ 毎週金曜に進捗報告(次の[製造時に行うこと]を参照)。**1**ヶ月を目処に完了。

発表の3日前までに自作のレビューを実施できるよう製造を完了させてください。
製造状況の共有については、後述の補足の項をご確認ください。

現場では常に納期が存在します。納期意識を持って課題に取り組むようにしてください。

◆課題の進め方

○製造時に行うこと

①gitへのpush

日々の作業完了時にgitへpushを行ってください。(コードを安全に保管する意味も兼ねています)
コミット時のコメントは**[** 日付 **-** 実装機能 **-** 実装中 **or** 完了 **]**のルールに従って行ってください。
例: [20230101 - ログイン機能 - 実装中] , [20230101 - ユーザーCRUD, 権限分け - 完了] etc

②Slackにて進捗報告

週次の進捗をフォーマットに従ってSlackにて報告してください。(フォーマットはslackの報告チャンネルにて共有します)
その際には、**DB**データのエクスポートを行い、SQLファイルの共有を行ってください。(p.15参照)

◆課題参考

○権限について

権限は「管理者と一般ユーザー」や「店舗側と利用者側」のようにアクセスできる権利が異なるユーザーのことです。
例えば
・このテーブルへのアクセスは管理者しかできない
・このデータの削除は管理者しかできない
・商品情報の登録/編集/削除は店舗側ユーザしかできない
等です。

やり方としては
・Userテーブルに例えばrole(役割)というカラムを持たせ、条件分岐を行う
・テーブルやWeb.phpでのアクセス先を分ける
等があります。

上記の方法を参考に実装を行ってみてください。

また、アクセス制御や機能一覧は右のような形で、
誰が、どこにアクセスが可能かを示す資料を作成してください。

【機能】

	一般ユーザ	管理ユーザ
ログイン	●	●
パスワードリセット	●	
投稿	●	
投稿の編集・削除	●	
足跡（いいね）	●	
コメント	●	
フォロー	●	
投稿検索	●	●
投稿削除	●	●
ユーザ検索		●
ユーザ削除		●

◆課題参考

○Ajax / APIについて

AjaxやAPIは新機能への挑戦として、自身で調べながら自作サービス内に取り入れてください。

AjaxはJavaScriptを使用した非同期通信を行います。
非同期通信といってもイメージは湧きづらいかと思いますが、イメージとしてはSNS等の「いいね！」機能や、
一番下までスクロールすると更に過去のコンテンツが出てくる「無限スクロール」等がそれにあたります。

フロント側(JavaScript等)からバックエンド側(Laravel)にアクセスし、処理を行ってもらう通信の仕方です。
今までのカリキュラムでは、画面遷移と同時に処理が走る同期通信しか扱っていませんでした。

非同期では画面遷移なくバックエンドのみで処理を走らせる実装が可能です。

それによって、画面遷移をしないまま様々な処理、
例えばDBへのいいねの登録や無限スクロール等の過去情報の取得を行うことが可能になります。

APIはApplication Programming Interfaceの略で、外部アプリケーションとの連携を行うことです。
例えばレビューサイトのアクセス画面でGoogleMapを表示する、等がAPIにあたります。

これら機能のいずれかを盛り込んだ内容を製作してみてください。(両方が必須ではありません)

◆設計

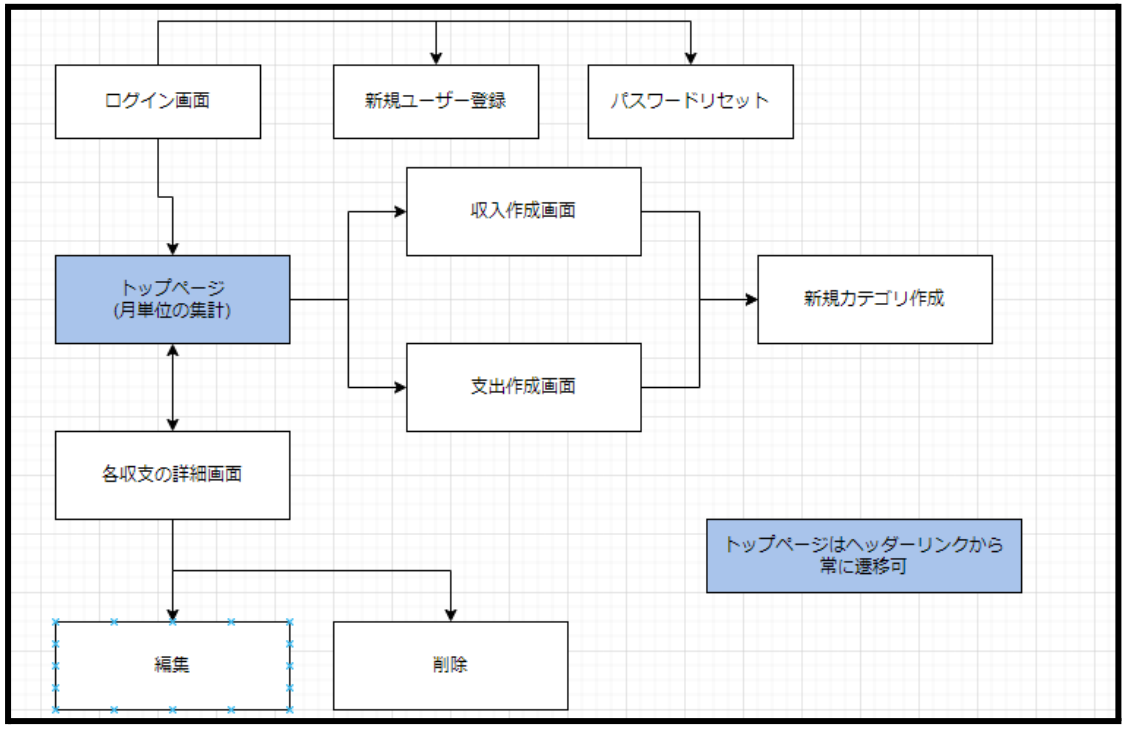
○設計：外部設計

要件定義書からシステムの大枠のイメージがついたら、次は画面を設計していきます。

画面に関する設計は主に①画面遷移図と②画面仕様書の2点です。

①画面遷移図

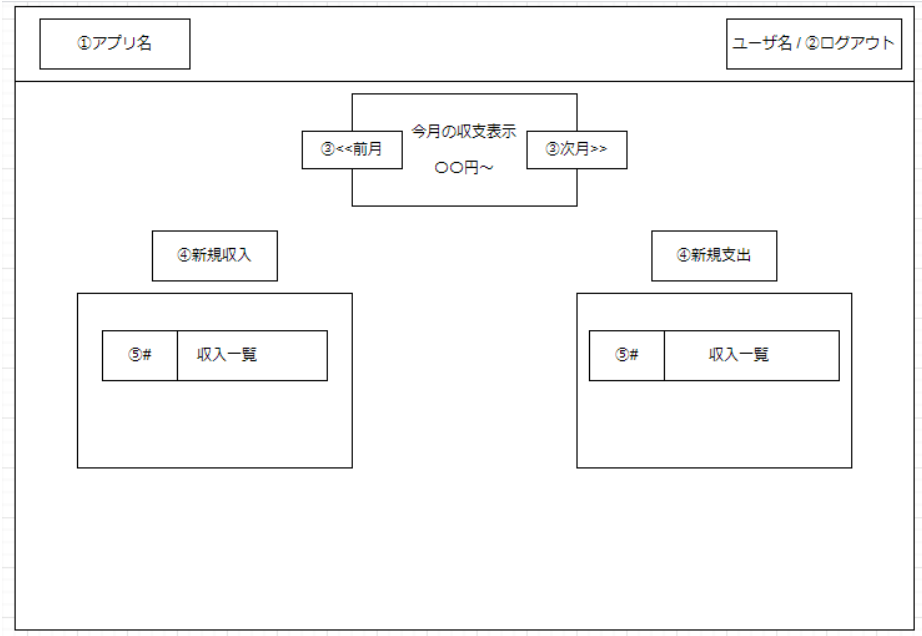
・・・どの画面からどの画面へ遷移することができるか。以下のように関連性を示した図のことです。
下の例では前章の家計簿アプリの画面遷移図を示しています。



②画面仕様書

・・・画面上に置くボタン等の要素や、クリックしたときの動作を説明する図、資料のことです。

下の例では先ほど同様家計簿アプリから、新規収入作成画面の画面仕様書を作成しています。



No.	type	機能	備考
①	リンク	トップページに遷移する	
②	リンク	ログアウトし(/login)に遷移する	
③	リンク	次月、先月のページに遷移する	
④	リンク	新規作成画面(create_xx)に遷移する	
⑤	リンク	各投稿の詳細画面へ遷移する	

これ以外にも例えば、フォームはテキストか、プルダウン式か等、表記すべき点は多数あります。
簡素な例でしか示していないので、具体的な書き方などを調べて作成してみてください。

また、上記の例はPDFとして入れてあるので確認してみてください。

◆設計

○設計：内部設計

DBに関する設計を作成します。

ER図とテーブル定義書を作成してください。
ER図作成時にはこちらのサイトが便利です(https://app.diagrams.net/) 提出時には必ず**PDF**化してください。

DB設計をする際には扱う値の大きさを検討し、格納データに適したデータ型を選択してください。

例えば･･･
　権限分けをroleカラムを作成し、0,1,2の3パターンで行いたい。 → **tinyint**型(上限255)を使う
　int型では40億程の数値まで対応できますが、扱うデータが小さい場合は無駄になってしまいます。

数値を格納する型だけでも**int, float, tinyint**など様々あります。

どのような型が存在し、扱いたいデータを格納するにはどれが適しているか。
現場においても重要な知識ですので、しっかりと調べてから作成してください。

前章6-Cの資料等も参考に作成してください。

◆設計

○設計：内部設計 - 補足　外部キーについて

外部キーとは、異なるテーブルを紐づけるためのカラムになります。
6章の課題にあった、支出とカテゴリの関連を例に考えてみます。

6章では「とある支出」に対して「特定のカテゴリ」が紐づく構成でした。
とある支出投稿に紐づく特定のカテゴリは、カテゴリの**id**があれば判断することが可能です。

このspendingsテーブルが持つカテゴリidのような、他テーブルと紐づけるためのカラムのことを外部キーと呼びます。

外部キーは [テーブル名_id]というカラム名であることが一般的です。
6章の内容を振り返って、外部キーがどういう働きをしているか確認してみてください。

自作サービスにおいても、様々なテーブルの関連が発生します。

誰が行った投稿か、のような形です。
(この場合は投稿に特定のユーザーが紐づくため、ユーザーidを投稿テーブルの情報として持っておけばよさそうです。)

このように、必要な外部キーを検討して作成してみてください。
ER図を書きながらだとイメージしやすいかと思います。

◆製造

○製造

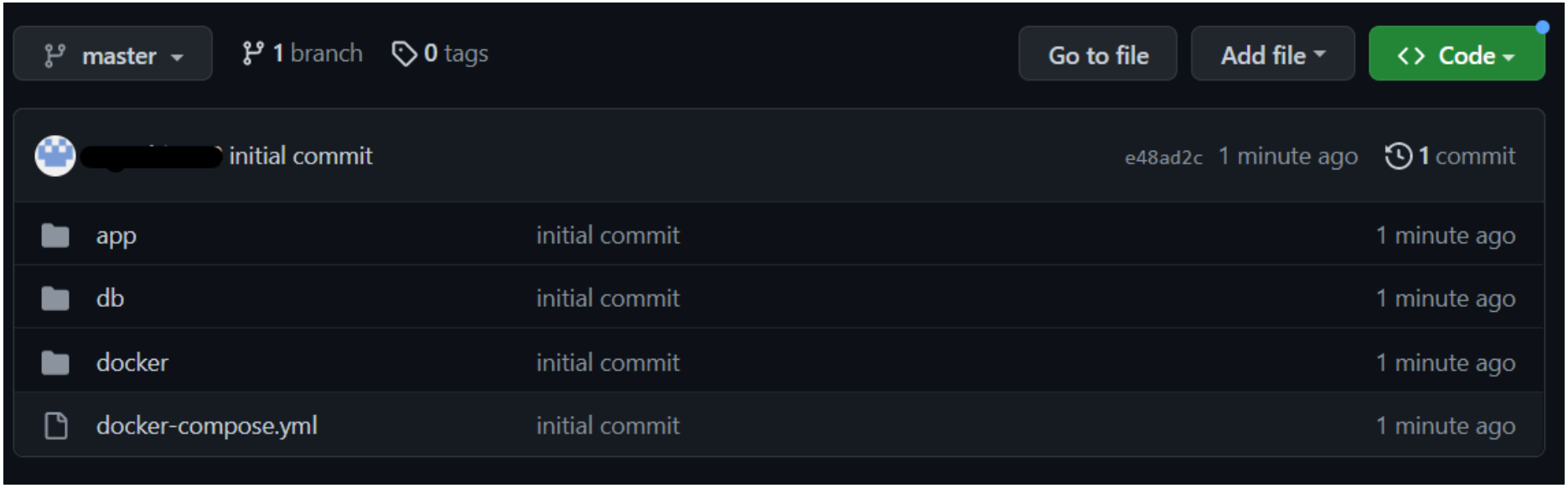
設計を提出しレビュー、修正を経て製造へのOKが出たら、製造です。

Dockerを使用して開発、Gitを使ってソースコードの管理を行って進めます。

また、製造の過程でも随時プッシュを行ってください。

プッシュの際は、環境ごとプッシュを行っていただきたいので、
docker-compose.ymlのあるディレクトリ(**docker-env**直下)でプッシュを行ってください。

以下のように、**app, db, docker**ディレクトリと**docker-compose.yml**があることを確認してください。



◆発表

○成果物発表

製造が完了したら成果物の発表会を行います。

その際には、

・参考にしたサイトとそのポイント
・システム説明
・実際の開発スケジュール
・課題や反省点

をまとめた発表資料を作成してください。

発表会の際には、その資料を使用しながらの説明と、画面共有を使用しての簡単なデモ操作を行っていただきます。

◆補足

○進捗報告 SQLファイルについて

製造の進捗報告の際には①**Git**への**Push** ②**SQL**ファイルの共有を行ってください。
各手順について以下で簡単にまとめていますので、参考にしてください。

まずは**SQL**ファイルの共有について

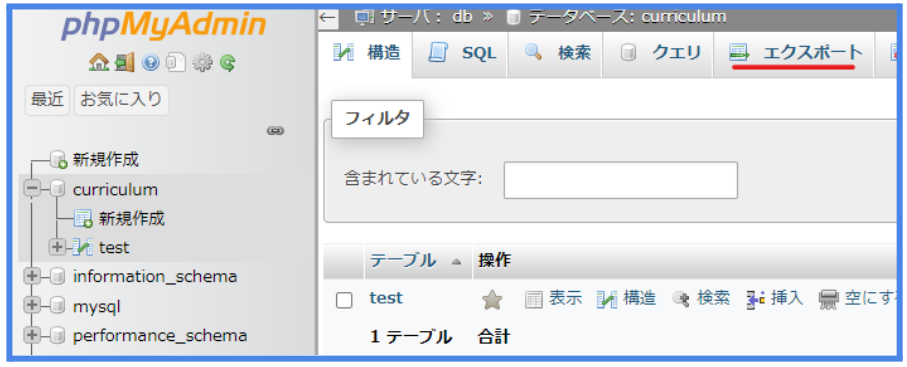
SQLファイルの取得は**phpMyadmin**から行えます。

まずは、phpMyAdminを開きます。

トップページから、今回使用するデータベースを選択します。
(今回の例ではcurriculumというDB名です)

選択後、上のタブから[エクスポート]を選択します。

その後、エクスポートボタンを押すと
SQLファイルがダウンロードできます。



ここでダウンロードしたファイルを共有するようにしてください。

◆補足

○進捗報告 Gitについて - 準備編

Gitは7-A, Git入門資料でも簡単に紹介した通り、バージョン管理を行うためのツールです。

ここではGitでPushを行う際の具体的な手順を紹介します。

まずは、プロジェクトを**Git**で管理するための準備を行います。

今回はdocker環境ごとGitに置きたいので、docker-compose.ymlがある階層(docker-env直下)で以下のコマンドを実行します。

```
PS C:\docker-env> git init
Initialized empty Git repository in C:/docker-env/.git/
```

git init コマンドでローカルリポジトリを作成します。

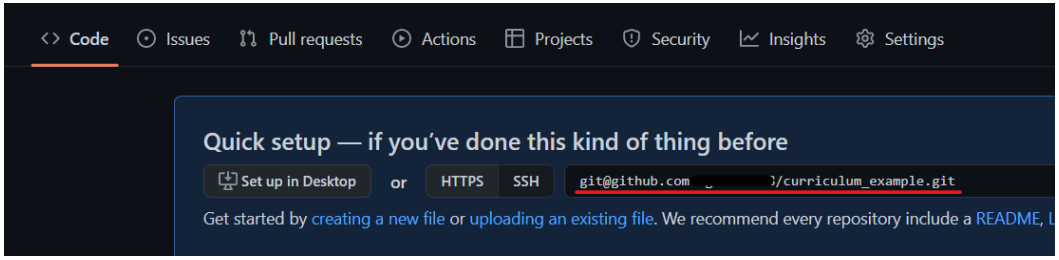
docker-env直下をローカルリポジトリとすることで、docker-env内の変更をGitで管理することが可能になります。

◆補足

○進捗報告 Gitについて - 準備編

git init が完了したら、次にどのリモートリポジトリと接続するかを登録します。

GitHub上でリポジトリを作成したら、下の赤線のようなSSHパスをコピーします。(リポジトリ名部分は黒塗りしてます)



このパスを使用して、どのリモートリポジトリ(Web上のコード保管場所)に変更を反映していくかを登録します。

先ほどgit init を行った場所で、リモートリポジトリを登録する [**git remote add origin ~パス~**]を実行します。

```
PS C:\docker-env> git remote add origin git@github.com.          /curriculum_example.git
PS C:\docker-env> git remote -v
origin  git@github.com:          /curriculum_example.git (fetch)
origin  git@github.com           /curriculum_example.git (push)
PS C:\docker-env>
```

コマンドを実行したら、[**git remote -v**]で登録されているリモートリポジトリの状況を確認できます。

ここまででローカルの準備とリモートへの繋ぎこみが完了しました。
次から、ローカルの変更をリモートに反映する手順を説明します。

◆補足

○進捗報告 Gitについて - Push編

リモートリポジトリの登録までが完了したら、リモートにローカルの状況を反映する作業を行います。

Gitへの反映時には、
①どのファイルを更新するか ･･･ [**git add ~ファイル名~**]
②ファイルの変更を確定 ･･･ [**git commit -m “コメント”**]
③リモートに反映 ･･･ [**git push origin ~ブランチ~**]
の3手順で行います。

まずは、**git add** についてです。

ローカル上で変更を加えたものの中で、どのファイルをリモートへの変更に反映するかを選択します。
ファイルA,B,Cとあった中で、Aだけをリモートにプッシュしたい場合、git add A とします。

今回は基本的に [**git add .**]で、全てのファイルの変更を反映する形で問題ないです 。
.(ドット)は階層にある全てのファイルを指します。

しかし、現場では勝手に変更を反映すると問題になるファイルがある場合があります。(環境の設定ファイル等)
今回は自身だけのプロジェクトなので問題ありませんが、現場では全てのファイルを**add**することはほぼあり得ません。

◆補足

○進捗報告 Gitについて - Push編

次に**git commit**です。

addで追加したファイルの変更を確定し、Gitに登録します。
コミット時には、どのような変更を加えたかを記述するコメントを付ける必要があります。

[**git commit -m “コメント内容”**]
でコミットを行います。

本課題ではコメントを以下のルールに従って行ってください。
[日付 **-** 実装機能 **-** 実装中**or**完了] 例: [git commit -m “20230101 - ログイン機能 - 実装中”]

最後に**git push**です。
コミットしたファイルをリモートリポジトリに反映します。

[**git push origin ~ブランチ名~**]

ブランチというのは、プロジェクト本体(**main**ブランチ)から枝分かれさせて作業を行うための場所です。
大規模なプロジェクトにおいては、実装する機能ごとにブランチを分けて作業を行うことがあります。

自作ではブランチ分けはしないので、[**git push origin main**]でmainブランチにプッシュを行って問題ありません。

尚、現場では**main**ブランチにプッシュする(プロジェクト本体を書き換える)ことはほぼありません。
間違ってもmainブランチを更新しないように気を付けてください。