インプット

※有料のものに関してはスキップしていただいて問題ないです！

1. HTML/CSS/Javascript/PHP/Laravelの学習(2w)
    a. Web教材(Progate/ドットインストール)で学習(5日)
        i. Progate：Command Line(1h)
        https://prog-8.com/courses/commandline
        ii. Progate：Git(1h)
        https://prog-8.com/courses/git
        ※Gitはツールを使わずにCommand Lineを利用すること
        iii. Progate：HTML/CSS(12h)
        https://prog-8.com/courses/html
        iv. Progate：Javascript(12h)
        https://prog-8.com/courses/es6

        v. ドットインストール：PHP(24h)
            1. 開発環境を整えよう
                a. Dockerを導入しよう [Windows 10版]全6回
                b. Dockerを導入しよう [macOS版]全3回

            2. **PHP**の雰囲気をつかもう
                a. はじめてのPHP全12回

            3. もっと詳しく**PHP**を学んでみよう
                a. 詳解PHP 基礎文法編全34回
                b. 詳解PHP ビルトイン関数編全32回
                c. 詳解PHP オブジェクト編全26回
                d. 詳解PHP ウェブ開発編全34回
                e. 詳解PHP データベース編全19回

            4. **PHP**でアプリケーションを構築しよう
                a. PHPでTodo管理アプリを作ろう 関数編全25回
                b. PHPでTodo管理アプリを作ろう クラス編全14回
                c. PHPでTodo管理アプリを作ろう 非同期通信編全30回

            5. フレームワークを使ってみよう
                a. Laravel 8入門 基本機能編全22回
                b. Laravel 8入門 データベース編全13回
                c. Laravel 8入門 CRUD処理編全20回
                d. Laravel 8入門 リレーション編全16回

b. PHP/Laravelは本でも学習(3日)
   PHPフレームワークLaravel入門 第2版
   https://www.amazon.co.jp/dp/4798060992/

c. PHP/LaravelでWebAPIの学習
   https://drive.google.com/file/d/1muV0vDYKn9elMKBBKOCw1CwXQA5zUId4/view?usp=sharing
   ※元記事

d. 上記をもう一度繰り返し学習する(2日)

はじめに

バックエンド（PHP） - デモアプリの開発 -

本内容は、既に配布されているかもしれませんが、
エンジニア育成プログラム　バックエンドコース2章 「デモアプリの開発」
の改訂版になります。

内容が完全に確定しているわけではないので、研修プログラムを受ける皆さんからのフィードバックもお待ちしてます。
研修実施中も内容に少し改訂が入るかもしれませんが、
その際は周知するようにします。内容面に関する質問・指摘もお待ちしてます。

また、本プログラムの目的はアウトプットであり、指示内容はあえて曖昧なままにしているところもあるので、記載されていることを踏まえたうえでのさらなる工夫や機能追加にも是非チャレンジしてみてください！

課題は「基本実装」と「応用実装」に分かれています。
応用実装は時間がかかるものもあり、必須というわけでもないので、まずは基本実装を一通り作り上げてから、応用実装に着手することをおすすめします。
基本実装は必須、応用実装は任意という位置づけにはしていますが、すべての課題に意図をもって作成しています。UZone開発に携わる上で必要になる技術に、ある程度触れるようにしているので、できる限り全ての課題に手を付けてほしいです（着手しなければ分からないことも分からないので…）。
Laravelに触るのが初めてで何もかも不安…という方は、一つの機能を実装するたびに適宜質問したり、レビューを依頼するようにしても全然OKです。ここに至るまでの面接等を通じて感じていると思いますが、重要視されているのは成果の出来ではなく、研修を通じてどれだけのことを習得できたか、そこに至るまでの姿勢になります。

コード（ファイル）やエラーの共有等、一般のコミュニケーションツールだと難しい点があるかと思い、Slackでこちらのスペースを開設しました。
招待リンク：LINK（2025/4/14まで有効）
質問などお待ちしております！

以下に開発の手助け・参考になるかもしれないワードをいくつか記載しておきますので、
是非これらを踏まえたうえで開発にチャレンジしてみてください！

RESTful、CRUD、MVC(S)、リポジトリパターン、FormRequest、Resource、Eloquent、
リレーション（DB）、
【TODO】 充実させる

環境構築

Laravel公式も推奨する「Laravel Sail」を使用して構築した環境を、パーツワンのPublic Repository（**https://github.com/Partsone/backend_tutorial_base**）に用意しました。
　今回は自分のGithubアカウントでここからソースをForkして、提出用のPublicレポジトリを作成するようにしてください。Fork後の環境構築手順は以下です。

1) （Docker Desktopをインストールし起動しておく
　　→ https://docs.docker.com/desktop/ ）

2) ターミナル上で（Forkしたuzone_be_pgディレクトリに移動し、）
```
./setup.sh
```
を実行する（本プロジェクトのセットアップ用に作成したスクリプトになります）

3) ターミナル上で
```
./sail artisan migrate
```
を実行する（データベースのマイグレーション）

　以上の手順でエラーが出ず、Dockerコンテナを立ち上げたままの状態で
http://localhost/ や http://localhost:8002/ の画面が問題なく表示される場合は環境構築完了となります。
　各種設定、OSの違いなどで上記の手順を順守したとしても問題が起こる可能性は十分にありうるので、その際は遠慮なく共有をお願いします。特にこの部分でエラーが起きる場合は他の人もここでつまずいている可能性が非常に高い上に、環境構築で必要以上に時間をとられてしまうのはこちらとしても本意ではありません。

注1）ネットで調べると各種コマンドの最初のブロックが ”sail”, “php”, “docker compose” などから始まっていることがよくありますが、本ワークスペース内ではこれらは全て”./sail”で置き換えて実行するようにしてください。基本的にはLaravel Sailによるものなのですが、”sail”ではなく”./sail”で実行できるようになっているのはこちらの設定によるものなので、どこで設定されているのかを探ってみるのも勉強になるかもしれません。
　また、正式なコマンドである、”./vendor/bin/sail”でも実行は可能で、公式などではsailへのエイリアス設定が推奨されていますが、本ワークスペースでは不要（非推奨）です。

注2）エディターは（おそらく）VScodeかと思います。プログラミングに慣れていてこだわりがある人以外は基本的にVScodeを使用するようにしましょう。
　開発で必要と思われる拡張機能については、ソースを取り込んだ時点で自動で推奨される設定にしてありますので、適宜インストールをお願いします。
以下にそれらのリストと簡単な説明を記載しておきます。

- Intelephense : PHP用のインテリセンス
- PHP Tools for Visual Studio Code : PHP用の拡張機能パッケージ
- Github Pull Requests : Githubのプルリクエストの管理に便利
- Redocly OpenAPI : Swaggerでエラーを指摘してくれる

以上は今回の開発にフォーカスした拡張機能ですが、他にも普遍的に活用できるものがたくさんあるので、自身でカスタマイズしてみてください。

注3) データベース内のデータを確認するのに、CLIだと分かりづらいので、各人のOSにあったソフトを導入してアクセスしてみてください。
　MacだとSequel Ace、WindowsだとHeidiSQL、A5:SQL Mk-2、OSを問わずDBeaverなどがクライアントツールとして有名です。
　以下にデフォルトの接続情報も記載しておきます。
これらは.envファイルで定義されているので、編集する場合はそちらからお願いします。

```
Host = localhost  (127.0.0.1)
User = sail
Password = password
Database = laravel
Port = 3306
```

注4) コンテナはGUI（Docker Desktop）からスタート・ストップができますが、コマンドでも可能ですので、こちらにも慣れておいてください。

```
./sail up -d    # スタート（-d をつけることでバックグラウンドで起動する）
./sail stop     # ストップ（downでもよいが、基本的にstopでよいと思われる）
```

注5) Composerを使用して様々なパッケージをインストールできますが、最後の応用課題（認証機能）も含め、全てのタスクをパッケージをインストールせずともこなすことができるようになっているはずです。パッケージは非常に便利で、実業務を大いにサポートしてくれる強い味方となりますが、あまり仕組みを考えずとも機能を実現することができてしまうという点で、研修を通したLaravelの学習を妨げうると考えています。
　従って、本研修では**Composerを使用せずに**課題をこなしていくことを強く推奨します。

注6) **Windowsユーザーへ**
多くの開発ツールやライブラリは、ネイティブなLinux環境での動作を前提としているため、WSL (Windows Subsystem for Linux) 環境を利用することを推奨します。
- Windows Subsystem for Linux とは
- WSLを理解する

そのため、ソースコードについてもWindowsの C: ドライブ（例：C:\Users\YourName\Documents など）には配置しないでください。

配置場所の例 (WSL内)
- /home/username/project_name
- /mnt/wsl/project_name設計

設計

今回の課題では以下のシチュエーションを想定して開発を行うこととします。
この仕様に従っていればそのほかの設定・プロパティの追加は自由（推奨）です。本プログラムは研修終了後のチーム開発を見据えているので、「人に理解してもらう」・「実際にサービスを利用してもらう」といった点を踏まえた上で自身の学びにつながるような成果物を期待しています。

ユーザー（**user**）・記事（**article**）・コメント（**comment**）の3つのオブジェクトがあります。Yahooニュースや Reditの簡易版（返信機能がない等）を想定してもらうと分かりやすいかもしれません。

以下が仕様の詳細です。

1) ユーザーは記事に対してコメントを投稿することができます
    a) コメントは10文字以上100文字以下のテキストを指すこととします
    b) コメントの内容は自由です
2) ユーザーは記事を選択することで過去に（自分以外も含む）ユーザーによって投稿されたコメントの一覧を閲覧することができます
    a) 一覧は投稿日時（≠ 編集日時）によって新しい順にソートされています
3) ユーザーは自分の投稿したコメント（のみ）を削除することができます
    a) 物理削除か論理削除かは各自で決定してください
4) ユーザーは自分の投稿したコメント内容（のみ）を編集することができます
    a) コメント内容とは投稿したテキストのことを指します

今回は、全体の基本課題を通じて上記「コメント」に関する**CRUD**機能（読取・登録・更新・削除）、すなわち、**GET、POST、PUT、DELETEのAPIを実装してもらいます**。

【基本課題】API・DB設計

本来は設計が終了した段階で課題作成者などに確認してもらう流れになるのですが、中々初めから考慮漏れのない設計を作り上げるというのは難しいですし、開発を進める上で必要に応じて要素の追加・削除を自身で行うというのも大いに勉強になると思いますので、ここの時点（設計段階）での成果物の提出は求めません。もちろん質問や確認等は歓迎です。

（1）API設計①

APIの設計書を作成してみましょう。
以下にテンプレートのリンクを添付していますので、こちらを使用しても、自身で1から作成しても構いません。実装の過程で色々修正しなければいけない箇所が出てくると思いますが、最新の設計を反映するのは（2）の課題で作成するSwagger UIのみで問題ないです。こちらは研修終了時に自分の当初の想定と完成物にどれだけの乖離が生じてしまったかを確認するために初期バージョンのままにしておくのも勉強になるかもしれません。

> API設計書テンプレート

これをレビューの対象にするつもりはないですが、参考にはなるので、”doc”ディレクトリ配下にその作成物を格納してください。

（2）API設計②

（1）で作成した設計書をもとに、OpenAPIが提供するSwagger UIを用いてAPI設計書を作成します。(1)で作成したものはいわゆるドキュメントで誰もが直感的に分かるような形式、ここで作成するのがエンジニアが実務上で使用する設計書というイメージになります。
Swagger UIは http://localhost:8002/ で確認できるようになっています。
デフォルトでは”./docs/api”ディレクトリ配下にある openapi-sample.yml の内容を参照しているため、同階層に自身のyml（yaml）ファイルを作成し、そこに設計書の内容を記載してください。
また、自身の作成したファイルを参照させるために.envファイルの`OPENAPI_FILE_NAME`を作成したファイルの名称に書き換えてください。
どの程度作りこむかはお任せしますが、最低限API設計書として成立するための情報量は記載するようにし、最新の実装部分の内容を反映させるようにしてください。
（後述しますが、Swagger UIはAPI設計書としてだけでなく、Postmanのようにそれ自体からAPIリクエストを飛ばすことで動作確認にも使用できるので、最新の情報を記載しないとSwagger UI上での動作確認ができなくなることもあります）

（3）**DB設計**
DB設計書の形態は色々考えられます。前述の通り、実装なしで初めから考慮漏れのない設計書を作成することは難易度が高いということも踏まえ、成果物の提出は求めません。ただ、最初想定していた設計と最終的に必要なモノの乖離を分析することは大いに勉強になると思いますので、何らかの形で設計書を作成することを強くおすすめします。
個人的にはER図を作成しながら設計の練習をすると、データベースのリレーションなどの勉強にもなると思います。
もしDB設計書を作成した場合は、形式は問わないので、API設計書同様に”doc”ディレクトリ配下にその作成物を格納してください。

【応用課題】Redocの導入
Swagger UIはAPI設計書として広く用いられていますが、いわゆる「設計書」としてはUIの問題もあり少し見づらさがあります（個人の感想）。
Redoc はOpenAPI / Swagger仕様で記述されたyml（yaml）ファイルからhtmlファイルを生成し、静的なドキュメントを作成してくれるツールで、これによってかなり見栄えが良い設計書を生成することができます。
Swagger UIはセットアップ時に導入しているためデフォルトで使用できる状態ですが、Redocも同様の設定を行い、本プロジェクトへ導入してください。
Portは8003を割り当て、Swagger UI（8002）と連番になるようにし、http://localhost:8003/ でアクセスできるようにしてください。

Hint💡）Swagger UIはDockerHub上に公開されているDockerイメージを引っ張ってきて、Port・ボリュームに対応するディレクトリを割り当てることでコンテナを定義しています。
これらは全てdocker-compose.yml上で定義しているので、同様の方法でRedocも導入することができます。

実装

【基本課題】APIの実装

設計書に従って（適宜修正しながら）GET、POST、PUT、DELETEのAPIを実装してみてください。同時にDB設計書に従って必要なDB構成を実装しましょう。

指示が曖昧ですが、ここから先に「正解」はないので、色々調べたり質問しながら作業を進めてみてください。冒頭に連ねたキーワード群は主にここのパートの助けになるものがほとんどなので、手が止まってしまった際は立ち戻って、それらを活用してみてください。

注1）Bladeテンプレートを用いて画面を作成する必要はなく、APIの作成のみが今回の課題です。従って、エンドポイントは"./routes"配下の"web.php"ではなく"api.php"に定義するようにしてください。

【基本課題】Seederの実装

APIの動作確認を行おうにも、データがないことには何も始まりません。サンプルデータ・マスタデータを一括で定義・登録する方法としてSeederが用意されているので、こちらを用いて十分量のデータを用意してください。

注2）マニュアルで定義するのもよいですが、Factoryのfakerヘルパを使用すると任意の文字列、人名などをランダムに生成できるので簡単に大量のデータを作成できます。ただ、入出力データの形式のテスト等のために、ある程度は自分で作成したほうが良いかもしれません。

【応用課題】認証機能の実装

認証（会員情報登録・ログイン）機能を作成し、コメントの投稿・編集・削除はログインしないとできない状態にしてください。
逆に、未ログインユーザーや会員でないユーザー（ゲスト）はコメントの閲覧のみが可能であり、他機能をリクエストした場合はログインや会員登録を要求する旨のエラーメッセージをレスポンスするようにしてください。

Laravelでは認証機能のためのBreezeをはじめとする様々なパッケージが存在し、本機能はそれらを用いることで簡単に実装ができます。実務面でもそれらを利用することがほとんどなのですが、今回は勉強のためにそれらは使用せずに、なるべく多くの部分を自分の手で実装してみてください。
（SanctumはSailでのLaravel導入時に自動でインストールされています。）

上記以外の詳細な仕様は以下の通りです。

1) ログインに必要なのはメールアドレスとパスワードの2つ
2) 会員登録は二段階認証を行う
    a) まずメールアドレスのみを登録し、そのアドレスにアクセストークンを送信する
    b) アクセストークンが認証できた場合のみ会員情報登録を行う
    c) パスワードはハッシュ化してデータベースに登録するようにする

これらが完了したうえでまだ余裕がある場合は、パスワードリセット・会員情報更新機能の実装にもチャレンジしてみてください。

注3）アクセストークンはURLでも文字列でも形態は問いません。パッケージを用いない実装が目的なので、必要以上に利便性・堅牢性にこだわる必要はありません。

注4）メールサーバーとしてMailpit（ http://localhost:8025/ ）をデフォルトで導入していますので、.envで設定を行い、使用するようにするとスムーズかと思います。

提出方法とその後

提出の際はREADMEに

- どの課題に取り組んだか
    - 各実装（基本・応用）をそれぞれ実施したかどうか
    - 途中であればその進捗
- 取り組んだ課題ごとの工夫点・アピールポイント（あれば）
    - 多少はあることが望ましいです
- 取り組んだ課題ごとの質問・疑問（あれば）
- 課題以外の点での工夫点・アピールポイント（あれば）
- 課題以外の点での質問・疑問（あれば）
- 注意して見てほしい点（あれば）
- 得た学び（感想でも）

を記載してください。上に記載したのはテンプレートのようなものなので、自分なりに様々な事柄を記載してみてほしいです。
（調べたり質問することで解決した問題についても、概要だけでも記載してもらえると非常に助かります。）

READMEの記載時には基本的にmarkdownを使用することになると思いますが、markdownはバックエンドに限らずエンジニアとしてやっていく際の必須技能ですので、レイアウト等にも配慮しながら色々試してみてください。思っている以上に色々なことが手軽に実現できるはずです。

~~また、こちら側で動作確認等を行う予定はないので、~~作成したSwagger UIを用いて全APIが想定通りの動作をするかは各自で十分に確認をお願いします。

以下の点のチェックもしっかりと実施してください。

- 指定されている仕様に則っているか
- 「人に理解してもらう」・「実際にサービスを利用してもらう」という点を踏まえたものになっているか

Issueの形式で担当者がレビューを返す形になると思いますので、それ以降はその指示内容に従ってください。