FeelEvent バックエンド API詳細設計

1PAC. INC.

Version 0.1.0
2016-09-05

目次

1. はじめに

本アプリケーションは `Feelnote` から派生して構築される **イベント情報サイト** のバックエンド詳細設計書になります。

1.1. 技術的備考

- `Feelnote` 側でもイベント情報を利用するため、テーブルはすべて `Feelnote DB` 内に構築します。
- `Feelnote` のサービス画面は `SPA (SinglePageApplication)` として提供しておりますが、 本アプリケーションはイベント情報ページを一般公開するため SEO の観点から従来の `MPA (Multi Page Application)` タイプとして実装します。

1.2. System requirements

- Nginx
- Ruby 2.3.1
- Ruby on Rails 5.0.0.1
- MySQL 5.7系
- Redis

1.3. Coding guideline

- ベーススタイル
  - https://github.com/fortissimo1997/ruby-style-guide/blob/japanese/README.md
  - https://github.com/fortissimo1997/ruby-style-guide/blob/japanese/README.ja.md
- Rubyコーディングスタイル（ベーススタイルへの追加のガイド）
  - https://github.com/styleguide/ruby
- Railsコーディングスタイル
  - https://github.com/bbatsov/rails-style-guide/blob/master/README.md

1.3.1. ワンパクオリジナル

定義

- `class` 定義の下は改行がなしと統一する
- 最終行の `end` の間では空行なしと統一する

後置 if, unless 文の使い分け

条件式が埋もれても見やすくするために、以下の通り後置 if 文を使う最低限のルールを設ける(例外あり)

- `return`, `raise` するときなどそこで処理を中断して終わらせるときの条件を指定する場合に限定する。

```
return unless @user.signed_in?
```

- 処理をする場合や変数への代入してその後の処理が継続して下に続く場合は一行でも通常の if 文にする。

```
if @flow.init?
  @user.assign_attributes @flow.get_data(:form)
end
```

1.4. 補足事項

- API の URL、リクエスト、レスポンスなどの情報は `JSON Schema` 形式で管理しています。
    - 各設計の基本情報内の `JSON Schema` 行で対応する API Document の場所を記載しています。

同 Github リポジトリ内の `doc/api_design/build/ja/` 配下に Markdown ファイルを格納しています。

2. ディレクトリ構成

アプリケーション全体のディレクトリ、ファイル構成の説明です。
※主要なディレクトリ、ファイルを抜粋しています。

```
.
├── Gemfile      # Gem 管理・設定ファイル
├── Gemfile.lock # Gem 管理・設定のロックファイル
├── Rakefile     # ターミナルから実行可能なタスク
├── app/         # アプリケーションを格納するディレクトリ
│   ├── assets/      # スタイルシートや画像などを格納するディレクトリ
│   ├── cells/       # Cells gem 用のロジックを格納するディレクトリ
│   ├── controllers/ # コントローラを格納するディレクトリ
│   ├── helpers/     # ヘルパーを格納するディレクトリ
│   ├── mailers/     # メール用コントローラを格納するディレクトリ
│   ├── models/      # モデルを格納するディレクトリ
│   └── views/       # ビューを格納するディレクトリ
├── config/ # プロジェクトの設定ファイルを格納するディレクトリ
│   ├── application.rb # すべての環境で共通の設定ファイル
│   ├── boot.rb # 起動ファイル
│   ├── database.yml # DB 設定ファイル
│   ├── environment.rb # 環境設定ファイル
│   ├── environments/ # 環境単位の設定ファイルを格納するディレクトリ
│   ├── initializers/ # 初期化ファイルを格納するディレクトリ
│   ├── locales/ # 辞書ファイルを格納するディレクトリ
│   └── routes.rb # ルーティング設定ファイル
├── config.ru # Rack の設定ファイル
├── db/        # DB の設定ファイルを格納するディレクトリ
├── doc/       # ドキュメントを格納するディレクトリ
├── lib/       # ライブラリを格納するディレクトリ
├── log/       # ログを格納するディレクトリ
├── public/    # Web サーバのドキュメントルート
├── spec/      # テスト(仕様)ファイルを格納するディレクトリ
├── tmp/       # キャッシュなど、一時的なファイルを格納するディレクトリ
└── vendor/    # 外部ライブラリを格納するディレクトリ
```

3. Tips

前章の Cells の他に利用している `Gem` の概要を説明します。

3.1. Decorator (Draper)

Model と View の間に Presenter(Draper) という中間層を設け、 Model や View に余計なロジックが増えることをさけるための Gem です。

3.1.1. Example

app/decorators/user_decorator.rb

```
class UserDecorator < Draper::Decorator
  delegate_all

  def full_name
    "#{first_name} #{last_name}"
  end
end
```

app/views/me/sample.html.erb

```
<h1><%= @user.full_name %></h1>
```

Draper については下記 Github を参考
https://github.com/drapergem/draper

4. 開発するにあたって

開発するにあたって大きく下記2点を意識して開発を行います。

- `RSpec` を用いたテスト駆動開発を心がける。
- `Pull-Request` ベースで極力レビューを行う。

4.1. RSpec を用いたテスト駆動開発

本アプリケーションでは `Rails` デフォルトの `Minitest` ではなく、 `RSpec` を用いてテストを行います。

- FactoryGirl
  - テストデータの機構として Rails デフォルトの `Fixture` がありますが、本アプリケーションでは `FactoryGirl` を利用
- Faker
  - ダミーデータを生成するための `Gem`

4.1.1. テストの種類

- 単体テスト
  - 主に Model や Decorator、Helper などのメソッドのテストを行う
- コントローラテスト
  - `render_views` でビューの生成まで行うことでビューで存在しないメソッドを実行するなどの不具合を潰せるようにする

参考本

> Everyday Rails - RSpecによるRailsテスト入門
> https://leanpub.com/everydayrailsrspec-jp/read

4.2. Pull-Request ベースの開発とレビュー

適切な粒度で作業ブランチを切り、適切な粒度でコミットをつくり、 手元の開発環境でしっかりテストを行った上でレビューを依頼しましょう。
基本はレビューが完了してからマージを行いましょう。

4.2.1. レビュー状態に持っていく前に確認すること

下記のような点を確認し、レビュワーにレビューの依頼を投げましょう。

- ブランチのテーマとは関係のないコードが含まれていないか
- コミットメッセージは簡潔かつ分かりやすいか
- 責務に応じてリファクタリングされているか
- メソッドを不必要にpublicにしていないか
- コードは読みやすいか/分かりやすいか
- クラス/メソッド/変数の名前は適切か
- 既存コードとの重複はないか
- 既存コードに影響を及ぼさないか
- コメントが必要な箇所はないか
- デバッグ用のコードは残っていないか
- もっとよいコードにできないか
- ファイル名は適切か
- 不要なファイルをコミットしていないか
- typoはないか
- コーディング規約に則っているか
- バリデーションは適切か
- セキュリティ上のリスクはないか
- フレームワークのレールから外れていないか
- ライブラリで代替できる処理はないか
- 将来的な負債は予期されないか
- トランザクションが必要な箇所はないか
- キャッシュが必要な箇所はないか
- 不必要なSQLを発行していないか
- テストケースは十分か
- 境界条件で正常に動作するか
- エラーハンドリングは必要な箇所で行なわれているか

5. A0101 イベントお気に入り登録

5.1. 基本情報

JSON Schema	api_design/build/unit/ja/A01.md
コントローラ,アクション	api/favorites#create

5.2. 仕様

1. 事前処理
    - ログイン認証
        - CookieSession を元にログイン認証を行い、未ログイン状態の場合は `403 レスポンス` を返す
2. 正常処理
    - お気に入り情報(*event_user_items*)を作成する
3. 例外処理
    - `event_id` から対象のイベント情報(*event_items*)が存在しなければ `404レスポンス` を返す
    - イベント公開フラグ(*event_items.published*)が **1(公開)** では無い場合は `404レスポンス` を返す
    - 現在日時がイベント公開開始日時(*event_items.published_at*)からイベント公開終了日時(*event_items.privated_at*)の範囲外の場合は `404レスポンス` を返す
    - すでにお気に入り情報(*event_user_items*)に登録済みの場合は `409 レスポンス` を返す

6. A0102 イベントお気に入り削除

6.1. 基本情報

JSON Schema	api_design/build/unit/ja/A01.md
コントローラ,アクション	api/favorites#destroy

6.2. 仕様

1. 事前処理
    - ログイン認証
        - CookieSession を元にログイン認証を行い、未ログイン状態の場合は `403 レスポンス` を返す
2. 正常処理
    - 対象のお気に入り情報(*event_user_items*)を削除する
3. 例外処理
    - `event_id` から対象のイベント情報(*event_items*)が存在しなければ `404レスポンス` を返す
    - イベント公開フラグ(*event_items.published*)が **1(公開)** では無い場合は `404レスポンス` を返す
    - 現在日時がイベント公開開始日時(*event_items.published_at*)からイベント公開終了日時(*event_items.privated_at*)の範囲外の場合は `404レスポンス` を返す
    - 対象のお気に入り情報(*event_user_items*)が無い場合は `404 レスポンス` を返す