FeelEvent バックエンド Site詳細設計

1PAC. INC.

Version 0.2.3
2016-09-19

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
```

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
│   ├── serializers/ # ActiveModelSerializers gem 用のディレクトリ
│   └── 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. レイアウト、パーシャル

3.1. レイアウト

`Rails` が提供するレイアウトテンプレートを利用します。
本アプリケーションはシンプルな構成となるためデフォルトである `application` レイアウトのみを利用します。

- レイアウト格納ディレクトリ
  - app/views/layouts/
- レイアウトテンプレート
  - application.html.erb

3.2. パーシャル

ヘッダー、フッターなどの共通で利用する主なパーシャルファイルを説明します。

- 共通利用のパーシャルファイル格納ディレクトリ
  - app/views/shared/

3.2.1. 共通利用する主なパーシャルファイル

```
app/views/shared/
  ├── _body_bottom.html.erb     # </body>直前用パーシャル
  ├── _body_top.html.erb        # <body>直後用パーシャル
  ├── _global_header.html.erb   # グローバルヘッダー
  └── _global_footer.html.erb   # グローバルフッター
```

3.2.2. 例

```
<%= render 'shared/global_header' %>
# app/views/shared/_global_header.html.erb が呼び出される

<%= render 'shared/global_footer', bar: 'baz' %>
# パーシャルに変数を渡す場合(locals: は略すことができる)
```

4. キャッシュ (Cells)

DB 操作などの処理を軽減するため、`Cells` gem のファイルキャッシュを利用します。

4.1. Cells とは

ロジック(コントローラ)とビュー(パーシャル)をセットで処理できる Gem で、
例として、サイドバーのランキングなどで利用するイメージです。

Cells については下記 Github を参考
https://github.com/apotonick/cells

4.2. 使い方

- ロジック(コントローラ)ファイルの格納ディレクトリ
  - app/cells/
- ファイル名
  - *_cell.rb と後ろに `_cell` を含める

4.3. ビューファイルについて

公式では `app/cells/` の中にビューファイルを置く形で説明されていますが、
ビューファイルを集約したいため `app/views/` 配下に格納します。

5. Tips

前章の Cells の他に利用している `Gem` の概要を説明します。

5.1. 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。

5.1.1. 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

5.2. ActiveModelSerializers

ActiveModel を Serialize し、API の JSON 出力を整形するためのライブラリです。

5.2.1. Example

app/models/book.rb

```
class Book < ActiveRecord::Base
  belongs_to :publisher
  has_and_belongs_to_many :authors
end
```

app/serializers/book_serializer.rb

出力したい項目を、attributes に指定することで出力するデータを制御する

```
class BookSerializer < ActiveModel::Serializer
  attributes :id, :title, :price, :published_at, :publisher_id
end
```

app/controllers/books_controller.rb

```
def index
  @books = Book.all
  respond_to do |format|
    format.html
    format.json { render json: @books }
  end
end
```

上記サンプルの他に、条件に応じて出力内容を変えたり、Association の解決などさまざまな機能がありますので 適宜公式サイト等で使い方を確認してください。

ActiveModelSerializers については下記 Github を参考
https://github.com/rails-api/active_model_serializers

6. 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。

6.1. 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`

6.1.1. 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

6.2. 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。

6.2.1. 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

7. A-1 ログイン

7.1. 基本情報

メソッド,URL	GET|POST /signin
コントローラ,アクション	users/sessions#signin
ルート名	new_user_session
ビュー	users/sessions/new.html.erb

7.2. リクエスト

- 下記フォーム定義を参照

7.3. 特記事項 (Devise)

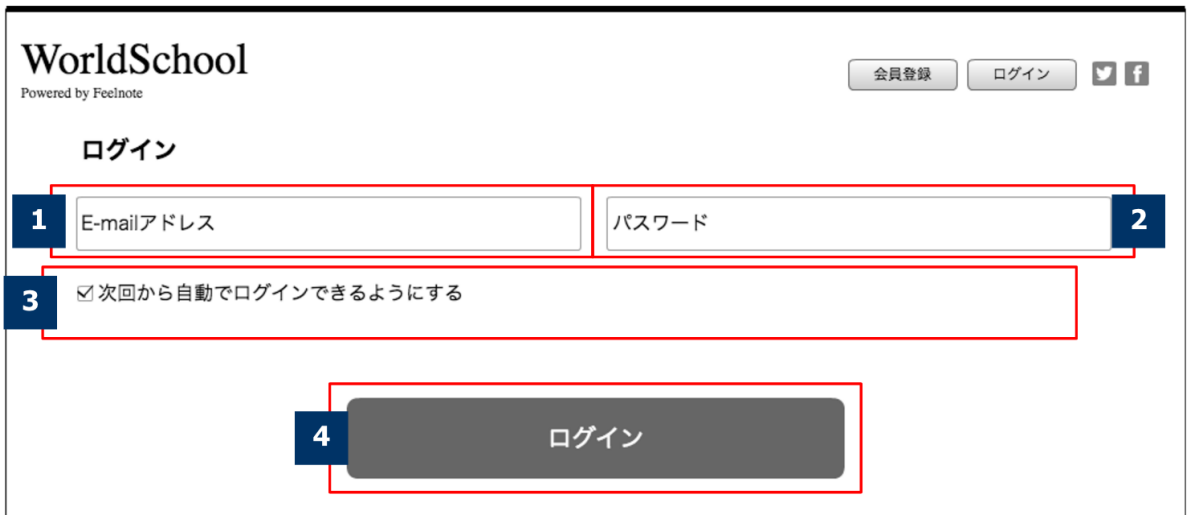
本ページはユーザ管理・認証を行う `Devise` gem を利用して実現します。

> Device については下記 Github を参考
> https://github.com/plataformatec/devise

7.3.1. Devise 設定情報

- 対象のテーブルは `event_users`
- 対象のログインキーのカラムは `email`
- 対象のパスワードのカラムは `password (encrypted_password)`

7.4. 仕様



1. メールアドレス
    - `email` フィールドとして input タグを出力
2. パスワード
    - `password` フィールドとして input タグを出力
    - エラーで同画面に戻ってきた際は値を保持しない
3. 自動ログイン
    - `checkbox` として input タグを出力
    - デフォルトは未選択状態
4. ログインボタン
    - ログイン成功時
        - セッション発行などの処理は `Devise` の処理に任せる
        - `B-1` へリダイレクトする
    - ログイン失敗時
        - 同画面を表示し、適宜エラーを表示する

7.5. フォーム定義

ID	項目名/フィールド名	フォーム種別	必須	備考
01	メールアドレス *user[email]*	email	○	
02	パスワード *user[password]*	password	○	
03	ログイン情報の保持 *user[remember_me]*	checkbox		

8. A-2 ログアウト

8.1. 基本情報

メソッド,URL	DELETE /signout
コントローラ,アクション	users/sessions#destroy
ルート名	destroy_user_session
ビュー	なし

8.2. リクエスト

- なし

8.3. 特記事項 (Devise)

本ページはユーザ管理・認証を行う `Devise` gem を利用して実現します。

Device については下記 Github を参考
https://github.com/plataformatec/devise

8.3.1. Devise 設定情報

- 対象のテーブルは `event_users`
- 対象のログインキーのカラムは `email`
- 対象のパスワードのカラムは `password (encrypted_password)`

8.4. 仕様

1. 全体仕様
    - セッション終了などの処理は `Devise` の処理に任せる
    - `B-1` へリダイレクトする

9. A-3 パスワード変更 (入力)

9.1. 基本情報

メソッド,URL	GET /password/new
コントローラ,アクション	users/passwords#new
ルート名	new_user_password
ビュー	users/passwords/new.html.erb

9.2. リクエスト

- 下記フォーム定義を参照

9.3. 特記事項 (Devise)

本ページはユーザ管理・認証を行う `Devise` gem を利用して実現します。

Device については下記 Github を参考
https://github.com/plataformatec/devise

9.3.1. Devise 設定情報

- 対象のテーブルは `event_users`
- 対象のログインキーのカラムは `email`
- 対象のパスワードのカラムは `password (encrypted_password)`

9.4. 仕様

1. メールアドレス
    - `email` フィールドとして input タグを出力
2. 送信ボタン
    - `A-3-a` へ POST 送信する

9.5. フォーム定義

ID	項目名/フィールド名	フォーム種別	必須	備考
01	メールアドレス *user[email]*	email	○	

10. A-3-a パスワード変更 (処理)

10.1. 基本情報

メソッド,URL	POST /password
コントローラ,アクション	users/passwords#create
ルート名	user_password
ビュー	users/passwords/new_complete.html.erb
メールビュー	users/mailer/ reset_password_instructions.html.erb

10.2. リクエスト

- A-3 フォーム定義を参照

10.3. 特記事項 (Devise)

本ページはユーザ管理・認証を行う Devise gem を利用して実現します。

Device については下記 Github を参考
https://github.com/plataformatec/devise

10.3.1. Devise 設定情報

- 対象のテーブルは event_users
- 対象のログインキーのカラムは email
- 対象のパスワードのカラムは password (encrypted_password)

10.4. 仕様

- 成功時(バリデートが通った時)
  - パスワード再設定準備などの処理は `Devise` の処理に任せる
    - メール送信処理も `Devise` に含まれる
    - テンプレートは適宜文言の差し替え処理を行う
- 失敗時(バリデートに引っかかった時)
  - `A-3` の入力画面でエラーを表示する

11. A-4 パスワード再発行 (入力)

11.1. 基本情報

メソッド,URL	GET /password/edit
コントローラ,アクション	users/passwords#edit
ルート名	edit_user_password
ビュー	users/passwords/edit.html.erb

11.2. リクエスト

- 下記フォーム定義書を参照

11.3. 特記事項 (Devise)

本ページはユーザ管理・認証を行う `Devise` gem を利用して実現します。

Device については下記 Github を参考
https://github.com/plataformatec/devise

11.3.1. Devise 設定情報

- 対象のテーブルは `event_users`
- 対象のログインキーのカラムは `email`
- 対象のパスワードのカラムは `password (encrypted_password)`

11.4. 仕様

1. 全体仕様
    - トークンチェックなどの処理は `Devise` の処理に任せる
2. パスワード
    - `password` フィールドとして input タグを出力
    - エラーで同画面に戻ってきた際は値を保持しない
3. パスワード(確認)

- `password` フィールドとして input タグを出力
- エラーで同画面に戻ってきた際は値を保持しない

11.5. フォーム定義

ID	項目名/フィールド名	フォーム種別	必須	備考
01	パスワード *user[password]*	password	○	
02	パスワード *user[password_confirmation]*	password	○	

12. A-4-a パスワード再発行 (処理)

12.1. 基本情報

メソッド,URL	PATCH /password
コントローラ,アクション	users/passwords#update
ルート名	(default)
ビュー	なし

12.2. リクエスト

- A-4 フォーム定義を参照

12.3. 特記事項 (Devise)

本ページはユーザ管理・認証を行う Devise gem を利用して実現します。

Device については下記 Github を参考
https://github.com/plataformatec/devise

12.3.1. Devise 設定情報

- 対象のテーブルは event_users
- 対象のログインキーのカラムは email
- 対象のパスワードのカラムは password (encrypted_password)

12.4. 仕様

- 成功時(バリデートを通った時)
  - パスワード再設定準備などの処理は Devise の処理に任せる
    - メール送信処理も Devise に含まれる
    - テンプレートは適宜文言の差し替え処理を行う
- 失敗時(バリデートに引っかかった時)
  - A-4 の入力画面でエラーを表示する

13. B-1 トップページ

13.1. 基本情報

メソッド,URL	GET /
コントローラ,アクション	events#index
ルート名	root
ビュー	events/index.html.erb

13.2. リクエスト

- 下記フォーム定義を参照

13.3. 仕様

1. 各フォーム要素
    - フォーム定義書の各フィールドを出力する
2. もっと見るボタン
    - `data-url` 属性にもっと見るの URL をセットする
    - 次のページが存在しない場合はボタンを表示しない
3. 検索結果
    - 初期表示は Rails 側で表示する
    - パラメータが付いていない場合は条件をクリアした条件とした検索結果を表示する

13.4. フォーム定義

ID	項目名/フィールド名	フォーム種別	必須	備考
01	キーワード *keyword_ids*	checkbox		
02	種別 *event_types*	checkbox		
03	開催場所 *held_area*	select(single)		
04	開催開始日 *held_started_on*	date		• **format:** YYYY-MM-DD
05	開催終了日 *held_ended_on*	date		• **format:** YYYY-MM-DD
06	参加費 *entry_fee_type*	select(single)		• **pattern:** (free:無料, pay:有料)
07	対象年齢 *qualifying_age_id*	select(single)		
08	フリーワード *word*	text		

14. C-1 マイページ (WIP)

14.1. 基本情報

メソッド,URL	GET /me
コントローラ,アクション	me#show
ルート名	me
ビュー	me/show.html.erb

14.2. リクエスト

- なし

14.3. 仕様

1. TODO

15. D-1 会員登録

15.1. 基本情報

メソッド,URL	GET /users/signup
コントローラ,アクション	users/registrations#new
ルート名	new_user_registration
ビュー	users/registrations/new.html.erb

15.2. リクエスト

- 下記フォーム定義を参照

15.3. 特記事項 (Devise)

本ページはユーザ管理・認証を行う `Devise` gem を利用して実現します。

Device については下記 Github を参考
https://github.com/plataformatec/devise

15.3.1. Devise 設定情報

- 対象のテーブルは `event_users`
- 対象のログインキーのカラムは `email`
- 対象のパスワードのカラムは `password (encrypted_password)`

15.4. 仕様

1. フォーム
    - フォーム定義書の各フィールドを出力する
2. 送信ボタン
    - `D-1-a` へ POST 送信する

15.5. フォーム定義

ID	項目名/フィールド名	フォーム種別	必須	備考
01	対象年齢 *user[qualifying_age_id]*	select(single)	○	
02	メールアドレス *user[email]*	email	○	
03	パスワード *user[password]*	password	○	
04	パスワード(確認用) *user[password_confirmation]*	password	○	
05	キーワード *user[keyword_ids]*	checkbox		
06	種別 *user[event_types]*	checkbox		
07	ニュースレター受け取りフラグ *user[subscribe_newsletter]*	checkbox		

16. D-1-a 会員登録 (処理)

16.1. 基本情報

メソッド,URL	POST /users
コントローラ,アクション	users/registrations#create
ルート名	user_registration
ビュー	users/registrations/new_complete.html.erb
メールビュー	users/mailer/confirmation_instructions.html.erb

16.2. リクエスト

- `D-1` フォーム定義を参照

16.3. 仕様

- 成功時(バリデートが通った時)
  - 仮登録処理は `Devise` の処理に任せる
    - メール送信処理も `Devise` に含まれる
    - 下記ユーザ設定情報を作成する
      - 対象年齢(*event_qualifying_ages_event_users*)
      - キーワード(*event_users_keywords*)
      - 種別(*event_types_event_users*)
    - 仮登録完了画面を表示する
- 失敗時(バリデートに引っかかった時)
  - `D-1` の入力画面でエラーを表示する

17. D-2 会員登録 (本登録)

17.1. 基本情報

メソッド,URL	GET /users/confirmation
コントローラ,アクション	users/confirmations#show
ルート名	new_user_confirmation
ビュー	なし

17.2. リクエスト

パラメータ名	説明	必須	備考
confirmation_token	トークン	○	

17.3. 特記事項 (Devise)

本ページはユーザ管理・認証を行う `Devise` gem を利用して実現します。

Device については下記 Github を参考
https://github.com/plataformatec/devise

17.3.1. Devise 設定情報

- 対象のテーブルは `event_users`
- 対象のログインキーのカラムは `email`
- 対象のパスワードのカラムは `password (encrypted_password)`

17.4. 仕様

- 成功時(バリデートを通った時)
  - 本登録などの処理は `Devise` の処理に任せる
  - 登録完了後に `A-1` に遷移する
- 失敗時(バリデートに引っかかった時)
  - `users/confirmations/new.html.erb` の入力画面でエラーを表示する