

FeelEvent バックエ ンド サイト詳細設計

1PAC. INC.

Version 0.1.0
2016-09-01

目次

1. はじめに	1
2. ディレクトリ構成	3
3. レイアウト、パーシャル	4
4. キャッシュ (Cells)	5
5. Tips	6
6. 開発するにあたって	7
7. A-1 ログイン	9
8. A-2 ログアウト	11
9. A-3 パスワード変更 (入力)	12
10. A-3-a パスワード変更 (処理)	13
11. A-4 パスワード再発行 (入力)	15
12. A-4-a パスワード再発行 (処理)	17
13. D-1 会員登録	19

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行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
│   └── 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: は略することができる)
```

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>

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、レビュー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
- フレームワークのル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	sessions#signin
ルート名	new_user_session
ビュー	devise/sessions/new.html.erb

7.2. リクエスト

パラメータ名	説明	必須	備考
event_user[email]	メールアドレス	○	
event_user[password]	パスワード	○	
event_user[remember_me]	ログイン情報の保持		

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。



Devise 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

The screenshot shows the login page for 'WorldSchool', powered by Feelnote. It includes links for '会員登録' (Sign Up) and 'ログイン' (Login), along with social media icons for Twitter and Facebook. The main heading is 'ログイン' (Login). Below this, there are four numbered annotations: 1. Points to the 'E-mailアドレス' (Email Address) input field. 2. Points to the 'パスワード' (Password) input field. 3. Points to the checkbox labeled '次回から自動でログインできるようにする' (Remember me for automatic login next time). 4. Points to the 'ログイン' (Login) button.

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

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	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。



Devise 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` ヘリダイレクトする

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	devise/password/new
ルート名	new_user_password
ビュー	devise/password/new.html.erb

9.2. リクエスト

- なし

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。



Devise 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

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	devise/password#create
ルート名	user_password
ビュー	devise/password/new_complete.html.erb
メールビュー	devise/mailer/ reset_password_instructions.html.erb

10.2. リクエスト

パラメータ名	説明	必須	備考
event_user[email]	メールアドレス	○	

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。



Devise 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	devise/password#edit
ルート名	edit_user_password
ビュー	devise/password/edit.html.erb

11.2. リクエスト

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

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。



Devise 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

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	devise/password#update
ルート名	(default)
ビュー	なし

12.2. リクエスト

パラメータ名	説明	必須	備考
event_user[password]	パスワード	○	
event_user[password_confirmation]	パスワード確認用	○	

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。



Devise 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-3` の入力画面でエラーを表示する

13. D-1 会員登録

13.1. 基本情報

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

13.2. リクエスト

- WIP

13.3. 仕様

- WIP