

FeelEvent バックエンド Site詳細設計

1PAC. INC.

Version 0.3.0

2016-09-25

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	8
7. A-1 ログイン	10
8. A-2 ログアウト	13
9. A-3 パスワード変更 (入力)	14
10. A-3-a パスワード変更 (処理)	16
11. A-4 パスワード再発行 (入力)	18
12. A-4-a パスワード再発行 (処理)	20
13. B-1 トップページ	21
14. B-1-a トップページ:プロモーション(Cells)	23
15. C-1 マイページ	24
16. C-2 設定変更	26
17. C-2-a 設定変更 (処理)	28
18. C-3 お気に入り一覧	29
19. D-1 会員登録	30
20. D-1-a 会員登録 (処理)	32
21. D-2 会員登録 (本登録)	33

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
│   ├── 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>

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、レビュー者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	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。



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). 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

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 <i>user[email]</i>	email	○	
02	パスワード <i>user[password]</i>	password	○	
03	ログイン情報の保持 <i>user[remember_me]</i>	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。



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	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。



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

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 <i>user[email]</i>	email	<input type="radio"/>	

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。



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	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。



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

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 <i>user[password]</i>	password	<input type="radio"/>	
02	パスワード <i>user[password_confirmation]</i>	password	<input type="radio"/>	

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。



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(バリデーションを通った時)
 - 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(バリデーションに引っかかった時)
 - 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 <i>keyword_ids</i>	checkbox		
02	種別 <i>event_types</i>	checkbox		
03	開催場所 <i>held_area</i>	select(single)		
04	開催開始日 <i>held_started_on</i>	date		• format: YYYY-MM-DD
05	開催終了日 <i>held_ended_on</i>	date		• format: YYYY-MM-DD
06	参加費 <i>entry_fee_type</i>	select(single)		• pattern: (free:無料, pay:有料)
07	対象年齢 <i>qualifying_age_id</i>	select(single)		
08	フリーワード <i>word</i>	text		

14. B-1-a トップページ:プロモーション(Cells)

14.1. Cells基本情報

コントローラ,アクション	events#top_promotion
ビュー	events/top_promotion.html.erb
キャッシュタイム	300秒

14.2. パラメータ

- なし

14.3. 仕様

- 枠説明
 - 左から順番に プロモーション情報の枠番号 (*event_promotions.frame_number*)が 1, 2, 3, 4 の情報がそれぞれ表示される
- 枠内のプロモーション情報
 - 各枠で掲載中で、かつ掲載終了日時(*event_promotions.ended_at*)が近いデータを取得
 - 各枠で掲載データがなければ該当枠を非表示にする
 - 全ての枠で掲載データがなければプロモーションブロック全体を非表示にする
- プロモーションバナー画像
 - トップページ用画像(*event_promotion.top_image*)を表示
- リンク
 - イベント詳細 [B-2](#) へのリンクを設置

15. C-1 マイページ

15.1. 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

15.2. リクエスト

- なし

15.3. ユーザー情報エリア仕様

1. メールアドレス

- メールアドレス(*event_users.email*)を表示する

2. 対象年齢

- ユーザの対象年齢(*event_qualifying_ages_event_users*)から登録情報を表示する

3. 種別

- ユーザの種別(*event_types_event_users*)から登録情報を表示する
- 複数あり

4. キーワード

- ユーザのキーワード(*event_users_keywords*)から登録情報を表示する
- 複数あり

15.4. お気に入りエリア仕様

- ユーザお気に入り情報(*event_favorites*)から **直近3件** のデータを表示する
- 3件以上存在する場合は「もっと見る」ボタンを設置

15.5. 種別イベント情報エリア仕様

- 枠説明
 - 種別情報の(`event_types`)の情報をそれぞれ表示する
 - 種別情報の作成日時(`event_types.created_at`)を昇順でソートする
- 枠内イベント情報
 - 開催中の情報で申込期限(`event_items.entry_ended_at`)に近い **9件** を出力する。
 - 3件表示状態とし、残りの情報は非表示状態で JavaScript で表示制御を行う。
 - 各枠で掲載データがなければ該当枠を非表示にする
 - 全ての枠で掲載データがなければ種別イベント情報エリア全体を非表示にする

16. C-2 設定変更

16.1. 基本情報

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

16.2. リクエスト

- 下記フォーム定義を参照

16.3. 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
 - 存在するデータはプリセットする

2. 送信ボタン

- C-2-a へ POST 送信する

16.4. 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	対象年齢 <i>user[qualifying_age_id]</i>	select(single)	○	
02	メールアドレス <i>user[email]</i>	email	○	
03	パスワード <i>user[password]</i>	password		
04	パスワード(確認用) <i>user[password_confirmation]</i>	password		
05	キーワード <i>user[keyword_ids]</i>	checkbox		
06	種別 <i>user[event_types]</i>	checkbox		
07	ニュースレター受け取りフラグ <i>user[subscribe_newsletter]</i>	checkbox		

17. C-2-a 設定変更 (処理)

17.1. 基本情報

メソッド,URL	PATCH /me/settings
コントローラ,アクション	me/settings#update
ルート名	なし
ビュー	なし

17.2. リクエスト

- C-2 フォーム定義を参照

17.3. 仕様

- 成功時(バリデーションが通った時)
 - ユーザ情報(*event_users*)を更新する
 - 下記ユーザ設定情報を更新する
 - 対象年齢(*event_qualifying_ages_event_users*)
 - キーワード(*event_users_keywords*)
 - 種別(*event_types_event_users*)
 - C-1 ヘリダイレクトしフラッシュメッセージを表示する
- 失敗時(バリデーションに引っかかった時)
 - C-2 の入力画面でエラーを表示する

18. C-3 お気に入り一覧

18.1. 基本情報

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

18.2. リクエスト

- なし

18.3. 仕様

1. 初回ページから API で表示するためバックエンド側の処理は不要

19. D-1 会員登録

19.1. 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

19.2. リクエスト

- 下記フォーム定義を参照

19.3. 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>

19.3.1. 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)`

19.4. 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

19.5. 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 <i>user[qualifying_age_id]</i>	select(single)	○	
02	メールアドレス <i>user[email]</i>	email	○	
03	パスワード <i>user[password]</i>	password	○	
04	パスワード(確認用) <i>user[password_confirmation]</i>	password	○	
05	キーワード <i>user[keyword_ids]</i>	checkbox		
06	種別 <i>user[event_types]</i>	checkbox		
07	ニュースレター受け取りフラグ <i>user[subscribe_newsletter]</i>	checkbox		

20. D-1-a 会員登録 (処理)

20.1. 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

20.2. リクエスト

- D-1 フォーム定義を参照

20.3. 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

21. D-2 会員登録 (本登録)

21.1. 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

21.2. リクエスト

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

21.3. 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>

21.3.1. 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)`

21.4. 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