FeelEvent バックエンド 管理画面詳細設計

1PAC. INC.

Version 0.2.0
2016-09-25

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. 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。

3.1. 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

3.2. 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

3.3. 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。

4. Tips

前章の Cells の他に利用している `Gem` の概要を説明します。

4.1. 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。

4.1.1. 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

4.2. 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。

4.2.1. 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

5. 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。

5.1. 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`

5.1.1. 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

5.2. 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。

5.2.1. 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

6. A-1 管理アカウント:新規登録

6.1. 基本情報

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

6.2. 特記事項

- このページはスーパーユーザ(*event_admin_users.user_type* が `super_user`)のアカウントのみが利用できる

6.3. リクエスト

- 下記フォーム定義を参照

6.4. 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

6.4.1. Devise 設定情報

- 対象のテーブルは `event_admin_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)`

6.5. 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

- A-1-a へ POST 送信する

6.6. 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	姓 *user[last_name]*	text	○	
02	名 *user[first_name]*	text	○	
03	メールアドレス *user[email]*	email	○	
04	パスワード *user[password]*	password	○	
04	パスワード(確認用) *user[password_confirmation]*	password	○	

7. A-1-a 管理アカウント:新規登録 (処理)

7.1. 基本情報

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

7.2. 特記事項

- このページはスーパーユーザ(*event_admin_users.user_type* が `super_user`)のアカウントのみが利用できる

7.3. リクエスト

- `A-1` フォーム定義を参照

7.4. 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
    - `A-2` へリダイレクトし、フラッシュメッセージを表示する
- 失敗時(バリデートに引っかかった時)
  - `A-1` の入力画面でエラーを表示する

8. A-1-1 管理アカウント:新規登録 (本登録)

8.1. 基本情報

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

8.2. 特記事項

- こちらのページは未ログイン状態でアクセスするため、スーパーユーザの権限チェック処理は行わない

8.3. リクエスト

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

8.4. 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

8.4.1. Devise 設定情報

- 対象のテーブルは `event_admin_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)`

8.5. 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
    - ユーザタイプ(*event_admin_users.user_type*)は `normal_user` として作成する
  - 登録完了後に `B-1` に遷移する
- 失敗時(バリデートに引っかかった時)
  - `admin/users/confirmations/new.html.erb` の入力画面でエラーを表示する

9. A-2 管理アカウント:一覧

9.1. 基本情報

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

9.2. 特記事項

- このページはスーパーユーザ(*event_admin_users.user_type* が `super_user`)のアカウントのみが利用できる

9.3. リクエスト

- なし

9.4. 仕様

1. 一覧
    - ID、指名、メールアドレスを表示する
    - 編集ボタン
        - `A-3` へ遷移する
    - 削除ボタン
        - JavaScript でConfirmを表示し、OKの場合は `A-4` へ DELETE 送信する

10. A-3 管理アカウント:編集

10.1. 基本情報

メソッド,URL	GET /admin/users/:event_admin_user_id/edit
コントローラ,アクション	admin/users#edit
ルート名	edit_admin_user
ビュー	admin/users/edit.html.erb

10.2. 特記事項

- このページはスーパーユーザ(*event_admin_users.user_type* が `super_user`)のアカウントのみが利用できる

10.3. リクエスト

- 下記フォーム定義を参照

10.4. 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
    - `A-2` へ POST 送信する

10.5. 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	姓 *user[last_name]*	text	○	
02	名 *user[first_name]*	text	○	
03	メールアドレス	email	○	

ID	項目名/フィールド名	フォーム種別	必須	備考
	user[email]			
04	パスワード *user[password]*	password		
04	パスワード(確認用) *user[password_confirmation]*	password		

11. A-3-a 管理アカウント:編集 (処理)

11.1. 特記事項

- このページはスーパーユーザ(*event_admin_users.user_type* が `super_user`)のアカウントのみが利用できる

11.2. 基本情報

メソッド,URL	POST /admin/users
コントローラ,アクション	admin/users#update
ルート名	なし
ビュー	なし

11.3. リクエスト

- `A-3` フォーム定義を参照

11.4. 仕様

- 成功時(バリデートが通った時)
  - 対象レコード(*event_admin_users*)を更新する
  - `A-2` へリダイレクトし、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

12. A-4 管理アカウント:削除

12.1. 基本情報

メソッド,URL	DELETE /admin/users/:event_admin_user_id
コントローラ,アクション	admin/users#destroy
ルート名	destroy_admin_user
ビュー	なし

12.2. 特記事項

- このページはスーパーユーザ(*event_admin_users.user_type* が `super_user`)のアカウントのみが利用できる

12.3. リクエスト

- なし

12.4. 仕様

- 成功時(バリデートが通った時)
  - 対象レコード(*event_admin_users*)を論理削除する
  - `A-2` へリダイレクトし、フラッシュメッセージを表示する
- 失敗時(バリデートに引っかかった時)
  - `A-2` へリダイレクトし、フラッシュメッセージを表示する

13. B-1 ログイン

13.1. 基本情報

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

13.2. リクエスト

- 下記フォーム定義を参照

13.3. 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

13.3.1. Devise 設定情報

- 対象のテーブルは `event_admin_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)`

13.4. 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

13.5. 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		

14. B-2 パスワード変更 (入力)

14.1. 基本情報

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

14.2. リクエスト

- 下記フォーム定義を参照

14.3. 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

14.3.1. Devise 設定情報

- 対象のテーブルは `event_admin_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)`

14.4. 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
    - `B-2-a` へ POST 送信する

14.5. 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	○	

15. B-2-a パスワード変更 (処理)

15.1. 基本情報

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

15.2. リクエスト

- B-2 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_admin_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

- 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)
  - `B-2` の入力画面でエラーを表示する

16. B-3 パスワード再発行 (入力)

16.1. 基本情報

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

16.2. リクエスト

- 下記フォーム定義書を参照

16.3. 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

16.3.1. Devise 設定情報

- 対象のテーブルは `event_admin_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)`

16.4. 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

16.5. 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	○	

17. B-3-a パスワード再発行 (処理)

17.1. 基本情報

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

17.2. リクエスト

- `B-3` フォーム定義を参照

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_admin_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)
  - 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)
  - `B-3` の入力画面でエラーを表示する

18. B-4 ログアウト

18.1. 基本情報

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

18.2. リクエスト

- なし

18.3. 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

18.3.1. Devise 設定情報

- 対象のテーブルは `event_admin_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)`

18.4. 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

19. C-1 記事一覧

19.1. 基本情報

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

19.2. リクエスト

パラメータ名	説明	必須	備考
state	公開状態	○	published(公開), unpublished(非公開)

19.3. 仕様

1. 公開の条件
    - 公開フラグ(*event_items.published*)が **true** で、現在日時が公開開始日時(*event_items.publish_startd_at*)から公開終了日時(*event_items.publish_ended_at*)の範囲内の記事が対象
2. 非公開の条件
    - 上記公開条件以外の記事が対象
3. 最終更新日時
    - 最終更新日時(*event_items.updated_at*)を表示
4. 最終更新担当者
    - 最終更新者ID(*event_items.evnt_admin_user_id*)から担当者名を表示
5. ID
    - イベント記事ID(*event_items.id*)を表示
6. 公開状態
    - 公開フラグ(*event_items.published*)から公開状態を表示

7. タイトル

- イベント記事タイトル(*event_items.title*)を表示

8. 公開開始日時

- 公開開始日時(*event_items.publish_started_at*)を表示

9. 公開終了日時

- 公開終了日時(*event_items.publish_ended_at*)を表示

10公開/非公開切り替えボタン

- `F-3` へ遷移
- `return_to` パラメータで現在の URL(パス) をセットする

11編集ボタン

- `F-2` へ遷移
- `return_to` パラメータで現在の URL(パス) をセットする

12削除ボタン

- JavaScript で Confirm 後に `F-4` へ DELETE 送信
- `return_to` パラメータで現在の URL(パス) をセットする

20. E-1 記事検索

20.1. 基本情報

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

20.2. リクエスト

- 下記フォーム定義書を参照

20.3. 仕様

1. 検索条件
    - すべての検索条件は検索項目単位で `AND` でつなげる
    - 公開フラグ
        - 公開フラグ(*event_items.published*)を元に検索する
    - ID
        - イベント記事ID(*event_items.id*)を元に検索する
    - フリーワード
        - イベント情報(*event_items*)のタイトル(*title*)と概要(*summary*)を対象にLIKE で部分一致検索する
    - 開催開始日
        - 開催開始日(*event_items.held_started_at*)が指定された日付以降の情報を対象とする
    - 開催終了日
        - 開催終了日(*event_items.held_ended_at*)が指定された日付以前の情報を対象とする
    - 最終更新者

◦ 最終更新者(*event_items.event_admin_user_id*)を元に検索する

2. 例外処理

- 該当の記事が0件の場合は検索結果、ページャを非表示にする

20.4. 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	公開フラグ *q[published]*	checkbox		
02	ID *q[id]*	text		
03	フリーワード *q[word]*	text		
04	開催開始日時 *q[held_started_at]*	date		
05	開催終了日時 *q[held_ended_at]*	date		
09	最終更新者 *q[event_admin_user_id]*	select(single)		

21. F-1 イベント新規作成 (入力)

21.1. 基本情報

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

21.2. リクエスト

- 下記フォーム定義書を参照

21.3. 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
2. 送信ボタン
   - `F-1-a` へ POST 送信する

21.4. 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	公開開始日時 *event_item[publish_started_at]*	datetime	○	
02	公開終了日時 *event_item[publish_ended_at]*	datetime	○	
03	タイトル *event_item[title]*	text	○	
04	主催者(複数可) *event_organizers[name]*	text	○	
05	キーワード(複数可) *event_keywords[id]*	checkbox	○	

ID	項目名/フィールド名	フォーム種別	必須	備考
06	種別 *event_type[id]*	select(single)	○	
07	開催開始日時 *event_item[held_started_at]*	date	○	
08	開催終了日時 *event_item[held_ended_at]*	date	○	
09	申込開始日時 *event_item[entry_started_at]*	date	○	
10	申込終了日時 *event_item[entry_ended_at]*	date	○	
11	参加費用 *event_item[entry_fee]*	text	○	
12	通過 *event_item[currency]*	select(single)	○	
13	対象者(複数可) *event_qualifying_ages[id]*	checkbox	○	
14	賞 *event_item[prize]*	text		
15	一覧用画像 *event_item[index_image]*	file	○	
16	詳細用カバー画像 *event_item[detail_image]*	file	○	
17	概要文 *event_item[summary]*	textarea	○	

21.5. 開催エリア情報のフォーム定義

※JavaScript側で下記セットを複数出力

ID	項目名/フィールド名	フォーム種別	必須	備考
01	エリア *event_places[area]*	select(single)	○	
02	GoogleMap表示フラグ	checkbox		

ID	項目名/フィールド名	フォーム種別	必須	備考
	event_places[map_enabled]			
03	タイトル *event_places[title]*	text		
04	郵便番号 *event_places[zip_code]*	text		
05	国(海外の時) *event_places[country]*	text		
06	州(海外の時) *event_places[state]*	text		
07	都道府県 *event_places[prefecture]*	select(single)		
08	市区町村 *event_places[city]*	text		
09	住所1 *event_places[address1]*	text		
10	住所2 *event_places[address2]*	text		

21.6. 概要文モジュールのフォーム定義

※JavaScript側で下記セットを複数出力

ID	項目名/フィールド名	フォーム種別	必須	備考
01	小見出し *event_summary_contents[headline]*	text		
02	本文 *event_summary_contents[content]*	textarea		
03	画像 *event_summary_contents[image]*	file		
04	レイアウト *event_summary_contents[layout]*	select(single)		
05	おすすめフラグ *event_summary_contents[recommended]*	checkbox		

21.7. 口コミモジュールのフォーム定義

※JavaScript側で下記セットを複数出力

ID	項目名/フィールド名	フォーム種別	必須	備考
01	小見出し *event_review_contents[headline]*	text		
02	本文 *event_review_contents[content]*	textarea		
03	画像 *event_review_contents[image]*	file		
04	レイアウト *event_review_contents[layout]*	select(single)		
05	プロフィール氏名 *event_review_contents[profile_name]*	text		
06	プロフィール画像 *event_review_contents[profile_image]*	file		

21.8. セミナー情報モジュールのフォーム定義

※JavaScript側で下記セットを複数出力

ID	項目名/フィールド名	フォーム種別	必須	備考
01	項目名 *event_seminar_contents[title]*	text		
02	内容 *event_seminar_contents[content]*	textarea		

22. F-1-a イベント新規作成 (確認)

22.1. 基本情報

メソッド,URL	POST /admin/events/new
コントローラ,アクション	admin/events#new_confirm
ルート名	confirm_new_admin_event
ビュー	admin/events/new_confirm.html.erb

22.2. リクエスト

- `F-1` フォーム定義を参照

22.3. 仕様

- 成功時(バリデートを通った時)
  - 定義しているビューの画面を表示する
- 失敗時(バリデートに引っかかった時)
  - `F-1` の入力画面でエラーを表示する

23. F-1-b イベント新規作成 (処理)

23.1. 基本情報

メソッド,URL	POST /admin/events
コントローラ,アクション	admin/events#create
ルート名	create_admin_event
ビュー	なし

23.2. リクエスト

- `F-1` フォーム定義を参照

23.3. 仕様

- 成功時(バリデートを通った時)
  - 該当のテーブルにそれぞれレコードを作成
  - `C-1` へリダイレクトする
- 失敗時(バリデートに引っかかった時)
  - `F-1` の入力画面でエラーを表示する

24. F-2 イベント変更 (入力)

24.1. 基本情報

メソッド,URL	GET /admin/events/:event_item_id/edit
コントローラ,アクション	admin/events#edit
ルート名	edit_admin_event
ビュー	admin/events/edit.html.erb

24.2. リクエスト

- 下記フォーム定義書を参照

24.3. 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
2. 送信ボタン
   - `F-2-a` へ POST 送信する

24.4. 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	公開開始日時 *event_item[published_started_at]*	datetime	○	
02	公開終了日時 *event_item[published_ended_at]*	datetime	○	
03	タイトル *event_item[title]*	text	○	
04	主催者(複数可) *event_organizers[name]*	text	○	
05	キーワード(複数可) *event_keywords[id]*	checkbox	○	

ID	項目名/フィールド名	フォーム種別	必須	備考
06	種別 *event_type[id]*	select(single)	○	
07	開催開始日時 *event_item[held_started_at]*	date	○	
08	開催終了日時 *event_item[held_ended_at]*	date	○	
09	申込開始日時 *event_item[entry_started_at]*	date	○	
10	申込終了日時 *event_item[entry_ended_at]*	date	○	
11	参加費用 *event_item[entry_fee]*	text	○	
12	通過 *event_item[currency]*	select(single)	○	
13	対象者(複数可) *event_qualifying_ages[id]*	checkbox	○	
14	賞 *event_item[prize]*	text		
15	一覧用画像 *event_item[index_image]*	file	○	
16	詳細用カバー画像 *event_item[detail_image]*	file	○	
17	概要文 *event_item[summary]*	textarea	○	

24.5. 開催エリア情報のフォーム定義

※JavaScript側で下記セットを複数出力

ID	項目名/フィールド名	フォーム種別	必須	備考
01	エリア *event_places[area]*	select(single)	○	
02	GoogleMap表示フラグ	checkbox		

ID	項目名/フィールド名	フォーム種別	必須	備考
	event_places[map_enabled]			
03	タイトル *event_places[title]*	text		
04	郵便番号 *event_places[zip_code]*	text		
05	国(海外の時) *event_places[country]*	text		
06	州(海外の時) *event_places[state]*	text		
07	都道府県 *event_places[prefecture]*	select(single)		
08	市区町村 *event_places[city]*	text		
09	住所1 *event_places[address1]*	text		
10	住所2 *event_places[address2]*	text		

24.6. 概要文モジュールのフォーム定義

※JavaScript側で下記セットを複数出力

ID	項目名/フィールド名	フォーム種別	必須	備考
01	小見出し *event_summary_contents[headline]*	text		
02	本文 *event_summary_contents[content]*	textarea		
03	画像 *event_summary_contents[image]*	file		
04	レイアウト *event_summary_contents[layout]*	select(single)		
05	おすすめフラグ *event_summary_contents[recommended]*	checkbox		

24.7. 口コミモジュールのフォーム定義

※JavaScript側で下記セットを複数出力

ID	項目名/フィールド名	フォーム種別	必須	備考
01	小見出し *event_review_contents[headline]*	text		
02	本文 *event_review_contents[content]*	textarea		
03	画像 *event_review_contents[image]*	file		
04	レイアウト *event_review_contents[layout]*	select(single)		
05	プロフィール氏名 *event_review_contents[profile_name]*	text		
06	プロフィール画像 *event_review_contents[profile_image]*	file		

24.8. セミナー情報モジュールのフォーム定義

※JavaScript側で下記セットを複数出力

ID	項目名/フィールド名	フォーム種別	必須	備考
01	項目名 *event_seminar_contents[title]*	text		
02	内容 *event_seminar_contents[content]*	textarea		

25. F-2-a イベント変更 (確認)

25.1. 基本情報

メソッド,URL	POST /admin/events/:event_id/edit
コントローラ,アクション	admin/events#edit_confirm
ルート名	confirm_edit_admin_event
ビュー	admin/events/edit_confirm.html.erb

25.2. リクエスト

- `F-2` フォーム定義を参照

25.3. 仕様

- 成功時(バリデートを通った時)
  - 定義しているビューの画面を表示する
- 失敗時(バリデートに引っかかった時)
  - `F-2` の入力画面でエラーを表示する

26. F-2-b イベント変更 (処理)

26.1. 基本情報

メソッド,URL	PATCH /admin/events/:event_id
コントローラ,アクション	admin/events#update
ルート名	update_admin_event
ビュー	なし

26.2. リクエスト

- `F-2` フォーム定義を参照

26.3. 仕様

- 成功時(バリデートを通った時)
  - 該当データをそれぞれ変更
  - `C-1` へリダイレクトする
- 失敗時(バリデートに引っかかった時)
  - `F-2` の入力画面でエラーを表示する

27. F-3 イベント公開状態切り替え

27.1. 基本情報

メソッド,URL	POST /admin/events/:event_item_id/:state
コントローラ,アクション	admin/events#update_state
ルート名	update_admin_event_state
ビュー	なし

27.2. リクエスト

パラメータ名	説明	必須	備考
event_item_id	イベント記事ID	○	
state	公開状態	○	published(公開), unpublished(非公開)
return_to	戻り先URL		

27.3. 仕様

1. 処理成功時

- 対象のイベント記事の公開フラグ(*event_items.published*)を **true(stateがpublishedの場合)** , **falsestateが(unpublishedの場合)** に変更する
- 併せて最終更新者ID(*event_items.event_admin_user_id*)も更新する
- `return_to` の戻り先 URL へリダイレクトし、フラッシュメッセージを表示する
  - `return_to` の指定が無い場合は `C-1` へリダイレクトし、フラッシュメッセージを表示する

2. 処理失敗時

- 存在しないイベント記事の場合は `404 レスポンス` を返却する
- 対象のイベント記事が既に指定された公開状態の場合は `404 レスポンス` を返却する

28. F-4 イベント削除

28.1. 基本情報

メソッド,URL	DELETE /admin/events/:event_item_id
コントローラ,アクション	admin/events#destroy
ルート名	destroy_admin_event
ビュー	なし

28.2. リクエスト

パラメータ名	説明	必須	備考
event_item_id	イベント記事ID	○	

28.3. 仕様

1. 処理成功時
    - 対象のイベント記事を削除する **(WIP:TODO:論理削除 or 物理削除を決定してから仕様に落とし込む)**
    - `return_to` の戻り先 URL へリダイレクトし、フラッシュメッセージを表示する
        - `return_to` の指定が無い場合は `C-1` へリダイレクトし、フラッシュメッセージを表示する
2. 処理失敗時
    - 存在しないイベント記事の場合は `404 レスポンス` を返却する