第52回 GTUG Girls れいるずハンズオン 資料

対象者

- Ruby, Rails を使ったことはないけれど、他の言語でのプログラミングの知識があること
- HTMLのタグを知っていること、CSSのクラス、IDを知っていること
- Terminalで入力する簡単なコマンド(cd, mkdir, ls 等)について知識があること
- 簡単なgitコマンドが使えること(add, commit, push 等)

今回できること/やらないこと

- できること
    - Ruby on Rails でブログ作る
    - Saas(Heroku)にアプリを公開する
- やらないこと(できないこと)
    - 凝ったデザイン
    - 公開したアプリでの永続的な画像保存

前提条件(環境)

Rails Girls Guide インストール(https://railsgirls.jp/install) を参考に、Ruby, Rails のインストールを行ってください。

Ruby, Rails

- Ruby 2.7以上がインストールされている環境であること
- 確認方法

```
$ ruby -v
ruby 2.7.2p137 (2020-10-01 revision 5445e04352) [x86_64-linux]
```

- Rails 6.0.0以上がインストールされている環境であること
- 確認方法

```
$ rails -v
(Ignoring ~ というメッセージが複数行出力された後にバージョンが出力される)
Rails 6.0.3.4
```

- rails new (アプリ名) を実行して正常にbundle installまで終わること

テキストエディタ

- vim, emacs, vscode 等テキストファイルを編集するエディタが使える環境であること

ブラウザ

- Chrome, Firefox, Safari が使える環境であること

Terminal

- コマンドプロンプト(Windows), Terminal(Mac, Linux) の操作が行えること

Railsでブログアプリを作る

Railsアプリを新規作成する

Terminalに以下のコマンドを入力して、Railsの新しいプロジェクトを作成します。

```
$ rails new gtug_blog
```

すると、`gtug_blog` というディレクトリを作成し、その中に必要なgem(Rubyのライブラリ)等が自動的にインストールされます。(以下の出力で終了となります)

```
(省略)
├─ ws@6.2.1
├─ y18n@4.0.0
├─ yargs-parser@13.1.2
└─ yargs@13.3.2
Done in 12.79s.
Webpacker successfully installed 🎉 🍰
```

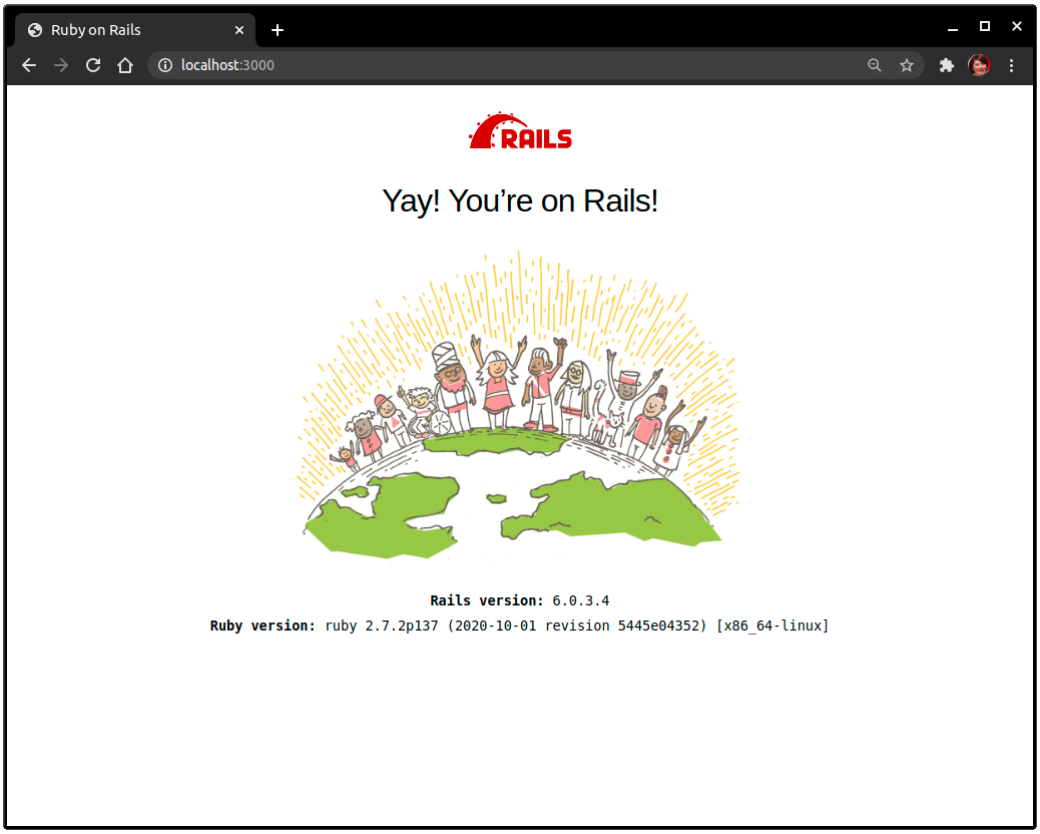
これでRailsアプリケーションが作成されました。

早速ブラウザで確認してみましょう。以下のコマンドをTerminalで入力します。

```
$ cd gtug_blog/ # ディレクトリを移動
$ bin/rails server # Webサーバを起動
```

Terminal に `Use Ctrl-C to stop` と表示されたら、ブラウザで `http://localhost:3000` にアクセスします。

すると、以下のように表示されます。



補足：Railsアプリでは、`rails server` というコマンドで、Webサーバを起動していますが、デフォルトでport 3000で動くようになっています。もし、3000 以外で起動したい場合には、`rails server --port xxxx(ポート番号)` とポート番号を指定してWebサーバを起動し、URLのポート番号を変更してアクセスしてください。

ブログを投稿できるようにする

ブログに必要な項目を洗い出す

論理名	物理名	型
タイトル	title	文字列
内容	description	テキスト
写真	picture	画像
投稿日	published_at	日付

generate scaffoldコマンドを使ってテンプレートを作成する

Terminalに以下のコマンドを入力して、ブログを投稿する画面を作っていきます。

```
$ bin/rails generate scaffold article title:string description:text
picture:string published_at:datetime
(メッセージ省略)
Running via Spring preloader in process 7248
      invoke  active_record
      create    db/migrate/20201106130338_create_articles.rb
      create    app/models/article.rb
```

```
      invoke    test_unit
      create      test/models/article_test.rb
      create      test/fixtures/articles.yml
      invoke  resource_route
       route    resources :articles
      invoke  scaffold_controller
      create    app/controllers/articles_controller.rb
      invoke    erb
      create      app/views/articles
      create      app/views/articles/index.html.erb
      create      app/views/articles/edit.html.erb
      create      app/views/articles/show.html.erb
      create      app/views/articles/new.html.erb
      create      app/views/articles/_form.html.erb
      invoke    test_unit
      create      test/controllers/articles_controller_test.rb
      create      test/system/articles_test.rb
      invoke    helper
      create      app/helpers/articles_helper.rb
      invoke      test_unit
      invoke    jbuilder
      create      app/views/articles/index.json.jbuilder
      create      app/views/articles/show.json.jbuilder
      create      app/views/articles/_article.json.jbuilder
      invoke  assets
      invoke    scss
      create      app/assets/stylesheets/articles.scss
      invoke  scss
      create    app/assets/stylesheets/scaffolds.scss
```

`scaffold` は、建築現場の足場を意味します。

`generate scaffold` コマンドでは、一覧・作成・編集・削除を行う画面に必要なテンプレートを作成します。

- `app` 以下には、`models` 、`controllers`、`views` のMVCが作成されます。
- `db/migrate` には、テーブルの定義のファイルが作成されます。
- `test` には、テスト用のテンプレートが作成されます。

`generate scaffold` のパラメータには、 モデルクラス名(単数形、テーブル名はモデルクラス名の複数型になります)、`カラム名:型` をスペース区切りで繋げて指定します。

指定できる型(一部)と指定方法を以下にあげます。

型	指定方法
文字列(255文字まで)	string
文字列	text
整数(4バイト)	integer

型	指定方法
整数(8バイト)	bigint
浮動小数	float
精度の高い小数	decimal
日時	datetime
時間	time
日付	date
真偽値	boolean

補足：Railsでは、`generate scaffold` コマンドで作成されるテーブル定義は、以下のカラムが自動的に付与されます。

- PRIMARY_KEY となる `id` (シーケンス値)
- レコード作成日時となる `created_at`
- レコード更新日時となる `updated_at`

テーブルを作成する

Terminalに以下のコマンドを入力して、scaffold コマンドで作成したDB定義のファイルからテーブルを作成します。

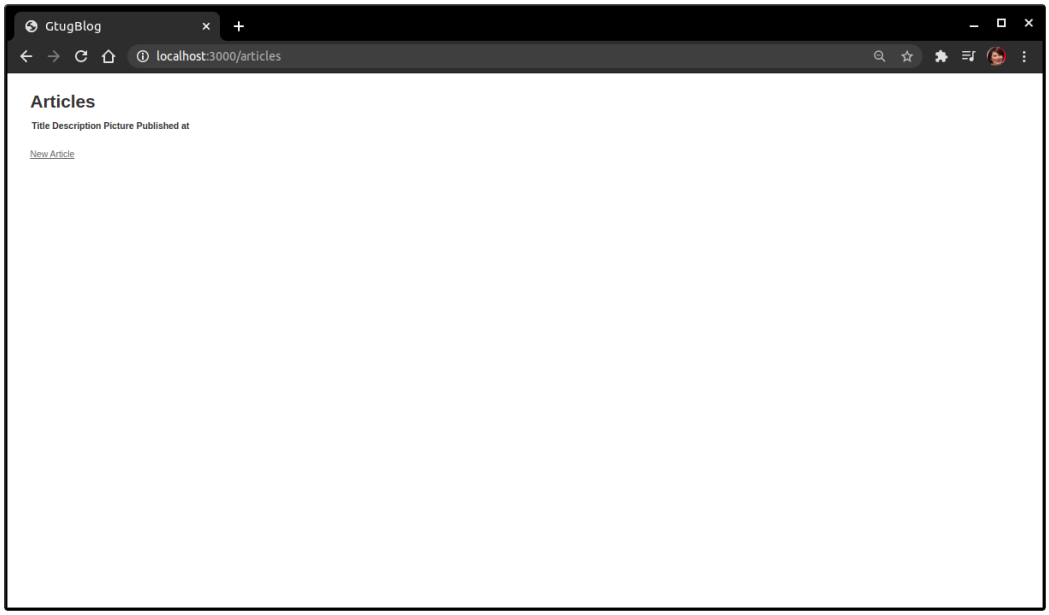
```
$ bin/rails db:migrate
== 20201106133059 CreateArticles: migrating
===================================
-- create_table(:articles)
   -> 0.0017s
== 20201106133059 CreateArticles: migrated (0.0020s)
==========================
```

起動して確認する

Terminalに以下のコマンドを入力して、ブラウザで `http://localhost:3000/articles` にアクセスします。

```
$ bin/rails server
```

すると、以下のような画面が表示され、ブログの一覧・作成・編集・削除が行えるよう画面が表示されます。



デザインを整える

naviバーを追加する

bootstrapを使って、いい感じにしておきます。

`app/views/layouts/application.html.erb` をテキストエディタで開きます。

```
 1 <!DOCTYPE html>
 2 <html>
 3   <head>
 4     <title>GtugBlog</title>
 5     <%= csrf_meta_tags %>
 6     <%= csp_meta_tag %>
 7
 8     <%= stylesheet_link_tag 'application', media: 'all', 'data-
turbolinks-track': 'reload' %>
 9     <%= javascript_pack_tag 'application', 'data-turbolinks-
track': 'reload' %>
10   </head>
11
12   <body>
13     <%= yield %>
14   </body>
15 </html>
```

`8行目` の `stylesheet_link_tag` の上に、以下のbootstrapのstylesheetのリンクタグを入れてください。

```
<link rel="stylesheet"
href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.c
ss">
<link rel="stylesheet"
href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-
theme.min.css">
```

`12行目`の`<body>`タグの下に、以下のnaviバーを追加します。

```
<nav class="navbar navbar-default navbar-fixed-top"
role="navigation">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="/">〇〇のブログ</a>
    </div>
    <div class="collapse navbar-collapse">
      <ul class="nav navbar-nav">
        <li class="active"><a href="/articles">投稿一覧</a></li>
      </ul>
    </div>
  </div>
</nav>
```

「○○のブログ」、「投稿一覧」は自由に変更してもらってかまいません。

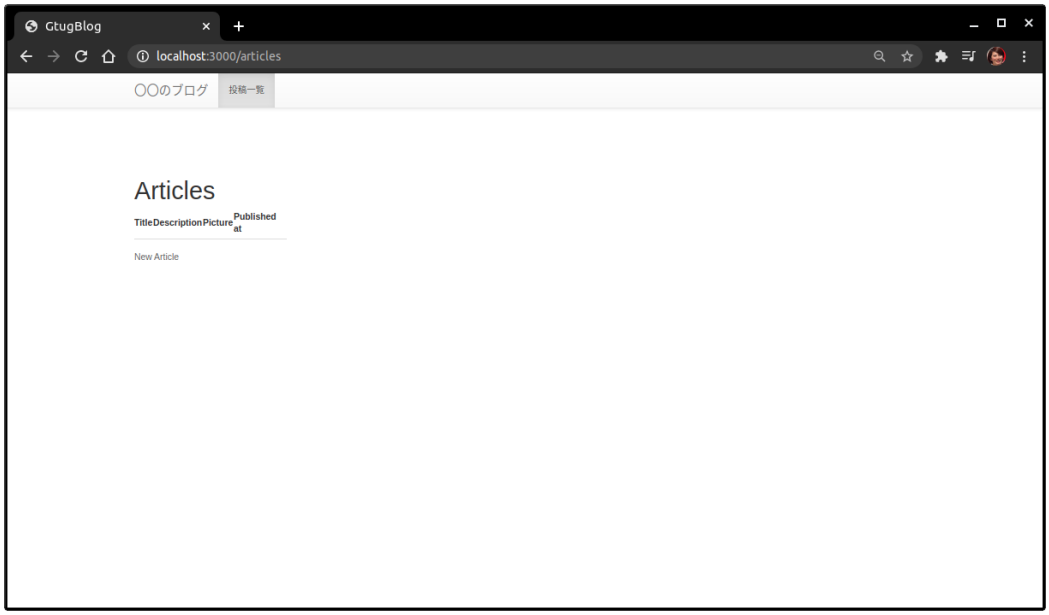
`13行目` の `<%= yield %>` を以下のように変更します。

```
    <div class="container">
      <%= yield %>
    </div>
```

`app/assets/stylesheets/application.css` をテキストエディタで開きます。

ファイルの一番下に、以下の定義を追加します。

```
body { padding-top: 100px; }
table, td, th { vertical-align: middle; border: none; }
th { border-bottom: 1px solid #DDD; }
```



補足：`app/views/layouts/application.html.erb` は、アプリケーションの共通のレイアウトになります。`yield` と書いてある部分に、各画面のテンプレート部分(今回作ったところでは、`app/views/articles` 以下にあるファイル)が埋め込まれた形で画面に表示されます。

おまけ：`rails server` コマンドは、app以下のファイルの修正をしている場合には再起動の必要はありません。ブラウザでアクセスした場合に、変更された内容を自動的に再読込して表示されます。(css, javascriptを修正した場合には、読み込みに少し時間がかかります。) テーブルの変更などをした場合には、`rails server` コマンドを一度止めてから、再起動してください。(Webサーバの再起動が必要です。)

使いやすくする

トップページを記事一覧にする

「○○のブログ」というトップページ(`http://localhost:3000` で表示される画面)は、最初の `Yay! You're on Rails!' の画面のままなので、記事一覧に変えます。

`config/routes.rb` をテキストエディタで開き、以下を追加します。

```
 Rails.application.routes.draw do
+  root to: redirect('/articles')
   # For details on the DSL available within this file, see
https://guides.rubyonrails.org/routing.html
   resources :articles
 end
```

`bin/rails server` を再起動して、`http://localhost:3000` にアクセスすると、記事一覧が表示されます。

補足：`config/routes.rb` では、WebサーバにアクセスされたURLとRailsアプリケーション側のマッピング(紐づけ)を行います。このファイルに定義されていないURLでアクセスがあると、Railsフレームワーク内でエラーになるようになっています。
`http://localhost:3000/rails/info/routes` を表示すると、Railsアプリケーションでアクセスできるurlと、アクセスされた場合にどのControllerのactionが呼ばれるかの確認が行えます。

時間をJSTにする

記事の新規投稿画面(http://localhost:3000/articles/new) を表示した時に、投稿日時(published_at) に設定されているは、現在時刻より-9時間になっています。

これは、Railsアプリケーションの時刻設定が、`UTC` になっているためなので、`JST` にします。

`config/application.rb` をテキストエディタで開き、`18〜19行目` を追加します。

```
  9 module GtugBlog
 10   class Application < Rails::Application
 11     # Initialize configuration defaults for originally
generated Rails version.
 12     config.load_defaults 6.0
 13
 14     # Settings in config/environments/* take precedence over
those specified here.
 15     # Application configuration can go into files in
config/initializers
 16     # -- all .rb files in that directory are automatically
loaded after loading
 17     # the framework and any gems in your application.
 18     config.time_zone = 'Tokyo'
 19     config.active_record.default_timezone = :local
 20   end
 21 end
```

`bin/rails server` を再起動して、 `http://localhost:3000/articles/new` にアクセスすると、`JST`の時刻で表示されるようになります。

画像もアップロードできるようにする

`carrierwave` というgem(Rubyのライブラリ)を使って、画像を添付できるようにしていきます。

Carrierwaveをインストールする

`Gemfile` をテキストエディタで開き、`10行目` の `gem 'carrierwave'` を追加します。

```
 8 # Use sqlite3 as the database for Active Record
 9 gem 'sqlite3', '~> 1.4'
10 gem 'carrierwave'
```

Terminalに以下のコマンドを入力して、インストールします。

```
$ bundle i # bundle install でもOK
```

`Gemfile` は、このアプリケーションで使用しているライブラリ(gem)を管理しているファイルで、使いたい gem があった場合には、このファイルに定義を追加し、その後`bundle install` コマンドでインストールします。

画像をアップロードして、表示できるようにする

Terminalに以下のコマンドを入力して、画像をアップロードするクラスを作ります。

```
$ bin/rails generate uploader Picture
Running via Spring preloader in process 60998
      create  app/uploaders/picture_uploader.rb
```

今作ったアップロード用のクラスを使って、テーブルに登録できるように設定をしていきます。

`app/models/article.rb` をテキストエディタで開き、2行目を追加します。

```
1 class Article < ApplicationRecord
2   mount_uploader :picture, PictureUploader
3 end
```

次に、投稿する時に画像をアップロードできるように、Viewを修正します。

`app/views/articles/_form.html.erb` をテキストエディタで開き、`text_field` を`file_field` に変更します。

```
   <div class="field">
     <%= form.label :picture %>
-    <%= form.text_field :picture %>
+    <%= form.file_field :picture %>
   </div>
```

補足：画像をアップロードする画面は、作成画面(new)と編集画面(edit)がありますが、

アップロードした画像を表示できるように、Viewを修正します。

`app/views/articles/show.html.erb` をテキストエディタで開き、変更します。

```
 <p>
   <strong>Picture:</strong>
-  <%= @article.picture %>
+  <%= image_tag(@article.picture_url, width: 600) if
@article.picture.present? %>
 </p>
```

サムネイルを作って、一覧にも表示できるようにする

サムネイルは、`mini_magick` というgemを使います。

`Gemfile` をテキストエディタで開き、`mini_magick` を追加したら、Terminal 上で `bundle i` します。

```
 gem 'carrierwave'
+gem 'mini_magick'
 # Use Puma as the app server
```

`app/uploaders/picture_uploader.rb` をテキストエディタで開き、以下の変更を行います。

```
 class PictureUploader < CarrierWave::Uploader::Base
   # Include RMagick or MiniMagick support:
   # include CarrierWave::RMagick
-  # include CarrierWave::MiniMagick
+  include CarrierWave::MiniMagick
+
+  version :thumb do
+    process :resize_to_fill => [50, 50]
+  end
```

一覧画面で、サムネイルを表示するように修正します。

`app/views/articles/index.html.erb` をテキストエディタで開き、変更します。

```
       <tr>
         <td><%= article.title %></td>
         <td><%= article.description %></td>
-        <td><%= article.picture %></td>
+        <td><%= image_tag article.picture_url(:thumb) if
article.picture.present? %></td>
         <td><%= article.published_at %></td>
```

Herokuでアプリを公開する

Heroku アカウントを作成し、Herokuコマンド(CLI)をインストールする

Rails Girls のガイドのHeroku ページを参考に、アカウントの作成とCLIのインストールをしてください。

https://railsgirls.jp/heroku

Heroku用の設定を行う

HerokuのGitリポジトリにpushすることで、アプリケーションが公開されるようになっています。

そのため、画像がアップロードされるディレクトリ(`public/uploads`)を、`.gitignore`に設定し、commit します。

```
$ echo public/uploads >> .gitignore
$ git add .
$ git commit -m "initial commit"
```

次に、ローカルの環境では `sqlite3` を使っていましたが、Heroku のデータベースは、`PostgreSQL` になるため、gem の設定を変更します。

`Gemfile` をテキストエディタで開き、以下の変更を行います。

```
 # Use sqlite3 as the database for Active Record
-gem 'sqlite3', '~> 1.4'
+group :development do
+  gem 'sqlite3', '~> 1.4'
+end
+group :production do
+  gem 'pg'
+end
 gem 'carrierwave'
```

`Gemfile` の変更を行ったので、`bundle i` を行いますが、今回は `--without` オプションを付けて実行してください。

```
$ bundle i --without production
```

補足：`group :development`, `group: production` は、それぞれの環境で使用するgem を分けて定義する方法です。

変更した内容を commitします。

```
$ git add .
$ git commit -m 'add pg.gem and update Gemfile.lock'
```

アプリを公開する

公開するアプリ名(my-first-rails-app) を決め、Terminalに以下のコマンドを入力します。

```
$ heroku create my-first-rails-app
```

既に名前が使われている場合は、以下のように表示されます。

```
$ heroku create my-first-rails-app
Creating ⬢ my-first-rails-app... !
 ▸    Name my-first-rails-app is already taken
```

正常にHerokuの環境ができた場合には、以下のようなメッセージが表示されます。

```
$ heroku create gtug-emorima-app
Creating ⬢ gtug-emorima-app... done
https://gtug-emorima-app.herokuapp.com/ |
https://git.heroku.com/gtug-emorima-app.git
```

Herokuにコードをpush すると、必要なライブラリのインストールなどが開始されます。

```
$ git push heroku master
Enumerating objects: 200, done.
Counting objects: 100% (200/200), done.
Delta compression using up to 4 threads
Compressing objects: 100% (174/174), done.
Writing objects: 100% (200/200), 436.36 KiB | 13.64 MiB/s, done.
Total 200 (delta 43), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
(省略)
To https://git.heroku.com/gtug-emorima-app.git
 * [new branch]      master -> master
```

コードをpushしただけでは、テーブルが作成されていないので、Heroku上でテーブルの作成を行います。

```
$ heroku run rails db:migrate
```

migrateが終わったら、Terminal上で、以下のコマンドを入力すると、ブラウザが起動して作成したアプリが表示されます。

```
$ heroku open
```

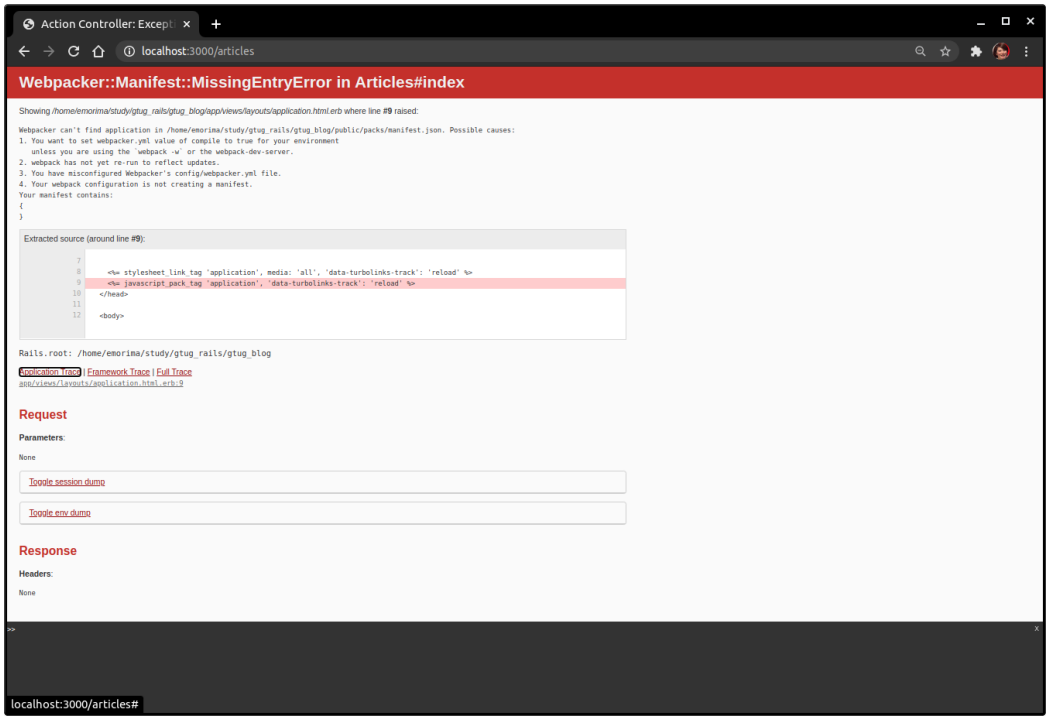
注意点

- 一定時間立つと、アップロードした画像は消えてしまいます。画像を残しておきたい場合には、保存先を外部ストレージに設定する必要があります。

トラブルシューティング

Webpacker::Manifest::MissingEntryError in Articles#index

ブラウザで `http://localhost:3000/articles` にアクセスした場合に表示されるかもしれません。



原因と対応方法

Rails が使用しているWebpacker内で、node V14+ である必要があるため、node のバージョンが古い場合に発生します。

node.js のバージョンを確認し、v14以上でない場合には、アップデートを行ってください。

```
$ node -v
v14.15.0
```

node.js のアップデートを行った後に、以下を行ってください。

```
$ rm yarn.lock
$ rm -rf node_modules/
$ bin/rails webpacker:install
```