

Djangoを最速でマスターする part1

Python Django

この記事は最終更新日から1年以上が経過しています。

初めに

最近、「機械学習に強い」、「簡単」などの理由からPythonを選ぶ人が多いと思います。

そんな人たちがWebを書こうと思った時にぶつかるのがDjangoの壁ですよ。

（あれ、そんなことない？ いやいや難しいですよ！）

僕は最初、Django Tutorialだけ日本語でやって、それ以降は全部英語の文献を読んだり、会社のコードをみて勉強をしたりしましたが、結構大変でした。

webを書きたいけど、PythonよりRubyの方が書きやすいからRubyの方がいい...となってしまうないように、これまで僕が勉強したDjangoの基本や応用を詰め込んで、Djangoを最速でマスターできるような記事を書いていきます！（3, 4記事の長さになる予定です）

* 最速でDjangoをマスターする part2

準備

Djangoが入っていない場合は、

```
$ pip install django
```

でインストールできるらしいです。

僕はAnacondaというディストリビューションでPythonをダウンロードしたので、最初から入っていました。

パソコンの容量を多少食うぐらい平気！という人はぜひ

Anacondaからダウンロードするのをおすすめします。Django以外にも色々最初から入っていて便利ですよ！

* Anacondaのインストールはこちらから

* Anacondaインストールについてわかりやすい記事

いよいよスタート

Webの勉強をする時、すでにWebが書ける人に、

「どうやって勉強すればいいですか？」

と聞くと、必ずと言っていいほど、

「あ、自分で何か一つアプリを作ればいいんじゃない??」

って言われますよね！！！！

確かに自分で全部作るのはかなりいい勉強なのですが、何も
ない状態から全部作るのは大変（というかほぼ無理？）だ
と思います。

なので、この記事では、「**順をおって一つアプリを作りながらDjangoを学ぶ**」という感じで進めて行こうと思います。

GitHubにもレポジトリを作っておきますので、コードもそれを参考にしてください。

（この記事だけでなく、今後の記事も通して「参考になったよ！」っていう方がいらっしゃいましたら、ぜひstarをつけてください！）

* [GitHubレポジトリはこちら](#)

何を作るの？

色々な機能を実装したいので、アプリの内容は面白さよりも実用性を重視しました。

内容としては、「管理ツール」を作っていきたいと思います。

マネジャーとワーカーの二種類のユーザーを想定しています。

マネジャーはワーカーに対して担当してもらう仕事セットを作成したり、ワーカーの稼働状況を確認したり、ワーカーの仕事が完了すればそれをチェックしたりします。

一方ワーカーは、仕事についてのヘルプページを読んだり、自分の労働状況を確認したりします。

これら以外に面白そうな機能が思いついたら実装していきますが、とりあえずこの辺について実装していきたいと思います。

Django基礎

まずはプロジェクトの作成をします。

Djangoでは、一つのアプリケーションをプロジェクトと呼び、一つのプロジェクトの配下にいくつかのアプリが動いている、という構造をしています。

自分がアプリケーションを作りたいディレクトリに移動し、以下のコマンドを実行してください。

（manager_project というのは僕がこのアプリケーションに勝手につけた名前なので、実際はなんでも大丈夫です）

```
$ django-admin startproject manager_project
```

プロジェクトが作成されましたね。

```
-- manager_project/  
    -- manager_project/  
        -- __init__.py  
        -- setting.py  
        -- urls.py  
        -- wsgi.py  
    -- manage.py
```

というファイル構成になっていると思います。

この中で主に使っていくのは setting.py と urls.py , manage.py です。

（ですが、これはワンポイントに使うだけで、主な開発はこれから作る models.py や views.py になります）

ではその次に、アプリを作ります。

```
$ cd path/to/manager_project
```

(先ほどのコマンドから継続して打つなら、\$ cd manager_project だけでオ

```
# 確認のため
```

```
$ ls
```

```
manage.py manager_project
```

```
# となっていますか？(一つ上のmanager_projectの中に入ってください)
```

```
$ python manage.py startapp manager
```



これで manager というアプリが作成されたと思います。

managerの中のファイル構成は以下です。

```
-- manager
  -- migrations
  -- __init__.py
  -- admin.py
  -- apps.py
  -- models.py
  -- tests.py
  -- views.py
```

この中でよく使うのは `models.py` , `views.py` , `tests.py` です。

普段はここばかりいじることになります。

ここで、「アプリを作ったよ!」ということをプロジェクトに報告しなければなりません。

ここでは `setting.py` を以下のように変更します。

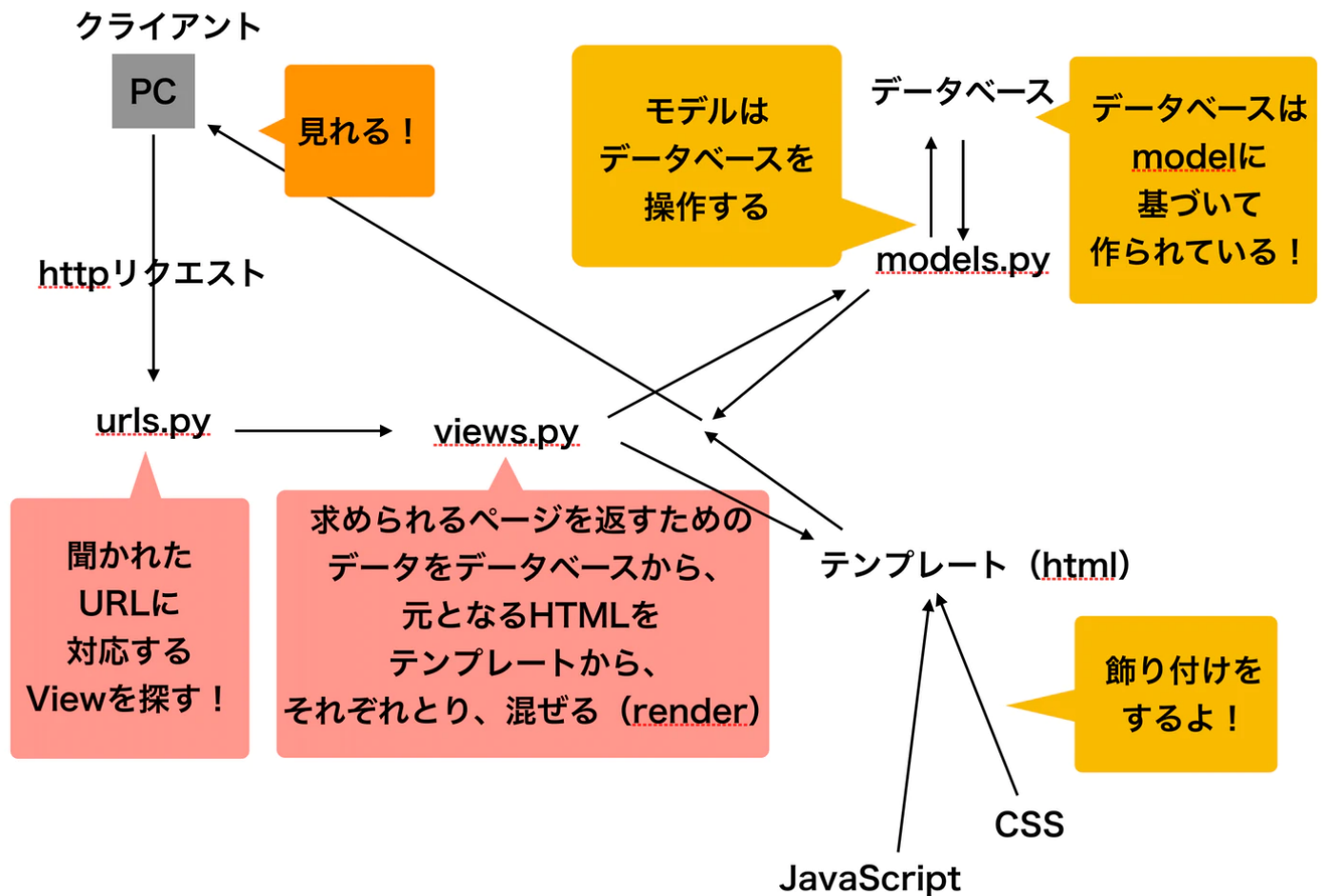
`setting.py`

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'manager', # 追加部分  
]
```

これでアプリが完成しました。

実際コードを書いていく前に、リクエストを受けてから Djangoがどのようにレスポンスを返すか見ていきます。

リクエストを受けてからレスポンスを返すまで



シンプルですね。実はこれだけです！

まずクライアントがhttpリクエストを送信する (<https://google.com> みたいなやつです！)

urls.py にはurlの正規表現とそれに対応するViewがいくつか書かれていて、マッチするurlがあればそのViewを見にいき、なければ「ページが見つかりません」的なのを返す（403とか404とかいうやつですね）。

余談ですが、正規表現を書くときにめっちゃ便利なサイトがあるので紹介しておきます！

[Online Regrex Tester](#)

Viewは `views.py` の中に書かれていて、主な操作をするやつです。データベースの中から必要なデータを、テンプレートの中から元となるhtmlファイルをとってきて両者を混ぜ合わせて、HTTPレスポンスとして返します。

これによりクライアントは欲しいデータが綺麗な形で記載されたページを見ることができるわけです！

Modelを書いてみよう

ということで、実際にコードを書いていきましょう。

ログイン機能やHijack機能など、先に実装したいものもありますが、少し複雑になってしまうので、今回はとりあえずワーカーのリストを表示する画面を実装していきます。

modelを作って、データの作成をしていきます。

（初めなので、コードは全部載せます。長くなるので次回以降は一部のみ載せる予定です。）

models.py

```
from django.db import models
```

```
class Person(models.Model):
```

```
    MAN = 0
```

```
    WOMAN = 1
```

```
    HOKKAIDO = 0
```

```
    TOHOKU = 5
```

```
    TOKYO = 10
```

```
    CHIBA = 11
```

KANAGAWA = 12

SAITAMA = 13

TOCHIGI = 14

IBARAGI = 15

CHUBU = 20

KANSAI = 25

CHUGOKU = 30

SHIKOKU = 35

KYUSHU = 40

OKINAWA = 45

名前

name = models.CharField(max_length=128)

誕生日

birthday = models.DateTimeField()

性別

sex = models.IntegerField(editable=False)

出身地

address_from = models.IntegerField()

現住所

current_address = models.IntegerField()

メールアドレス

email = models.EmailField()

class Manager(models.Model):

部署の定数

DEP_ACCOUNTING = 0 # 経理

```
DEP_SALES = 5 # 営業
DEP_PRODUCTION = 10 # 製造
DEP_DEVELOPMENT = 15 # 開発
DEP_HR = 20 # 人事
DEP_FIN = 25 # 財務
DEP_AFFAIRS = 30 # 総務
DEP_PLANNING = 35 # 企画
DEP_BUSINESS = 40 # 業務
DEP_DISTR = 45 # 流通
DEP_IS = 50 # 情報システム
```

```
# 人
```

```
person = models.ForeignKey('Person')
```

```
# 部署
```

```
department = models.IntegerField()
```

```
# 着任時期
```

```
joined_at = models.DateTimeField()
```

```
# やめた時期
```

```
quited_at = models.DateTimeField(null=True, blank=True)
```

```
class Worker(models.Model):
```

```
# 人
```

```
person = models.ForeignKey('Person')
```

```
# 着任時期
```

```
joined_at = models.DateTimeField()
```

```
# やめた時期
```

```
quited_at = models.DateTimeField(null=True, blank=True)
```

担当上司

```
manager = models.ForeignKey('Manager')
```

ここで、コードの説明です。

モデルのクラスは基本的に `models.Model` を継承して作ります。

属性はそのデータ型を指定する必要があって、例えば整数値なら `IntegerField` のようにします。

- `CharField` : 文字列（メモリ管理上、`max_length`を指定しなければいけない）
- `IntegerField` : 整数
- `DateTimeField` : 時刻

など生のデータを入れる属性もあれば、

- `ForeignKey`
- `ManyToMany`

のように、モデル同士の関係を表す属性もあります。

Model field reference (Fieldの一覧)

例えば、ForeignKeyは「1 : 多」の関係を示すフィールドです。

Managerクラスの中に、

```
person = models.ForeignKey('Person')
```

とありますが、これはpersonとmanagerが「1 : 多」の関係を示していることを表します。

(同じ人でも何回か違う部署のマネジャーになることがありますよね！)

(※注意 :

Django2.0からは、ForeignKeyにはon_delete引数が必須になりました。

参考 : <https://qiita.com/lemmy/items/5fa7f6e5ca29d2446174>
)

また、

```
# 部署の定数

DEP_ACCOUNTING = 0 # 経理
DEP_SALES = 5 # 営業
DEP_PRODUCTION = 10 # 製造
DEP_DEVELOPMENT = 15 # 開発
DEP_HR = 20 # 人事
DEP_FIN = 25 # 財務
DEP_AFFAIRS = 30 # 総務
DEP_PLANNING = 35 # 企画
DEP_BUSINESS = 40 # 業務
DEP_DISTR = 45 # 流通
DEP_IS = 50 # 情報システム
```

のように、定数を設定するやり方はよく使われます。

数値を飛び飛びにしているのは、途中で間に何か入れたい時に対応できるようにするためです。

モデル（クラス）を新しく作ったり、モデルの中の属性を変更したりした時は、データベースのテーブルに変更を反映する必要があります。

それでは**master directory**から以下のコマンドを実行してください。

```
$ python manage.py makemigrations  
$ python manage.py migrate
```

これで変更はデータベースに反映されます。

DjangoではデフォルトでSQLiteが採用されていますが、僕は普段MySQLを使っています。

この辺りのデータベースの話は長くなるし、そんなに気にしなくても開発は結構できるので、この記事では無視します。

それではこのモデルを元にデータを作ります。

データベースをいじるときなどは、DjangoのShellに移動します。

master directoryから以下のコマンドを実行して見てください。

```
$ python manage.py shell
```

```
[~/Library/Mobile Documents/com-apple-CloudDocs/dev/django/manager_project]
python manage.py shell
Python 3.6.0 |Anaconda 4.3.0 (x86_64)| (default, Dec 23 2016, 13:19:00)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?      -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help    -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

In [1]:
```

こんな感じの画面になると思います！

（にっこりマークがあったり、右に現在のGitHubのブランチ名が表示されたりしているのは、ターミナルをbashからzshに変えているからです。コマンドの切れ目がわかったり、今いるブランチ名がわかったりとすごく便利なので、[この記事](#)を参考にしてぜひ変更して見てください！）

これでDjangoのShellに入ることができました！ここでデータを作っていきます。

出身地とか性別はかなり適当に作りました。

```
import datetime
import pytz
from manager.models import Person, Manager, Worker

for i in range(200):
```

```
birthday = datetime.datetime(year=1980 + i % 20, month=1
Person.objects.create(name="person{}".format(i), birthda
```

まずPersonに基づくデータを作ります。

datetimeやpytzについては省略します。

オブジェクトの作り方としては2通りあります。

1) MODEL_NAME.objects.create(kwargs)

2) obj = MODEL_NAME(kwargs)
obj.save()

MODEL_NAMEはモデルクラスの名前
kwargsはキーワード引数

です。

create関数はデータベースに保存するところまでやってくれます。

ちなみに、作ったモデルインスタンス（個々のデータのこと）を削除するには以下のようにします。

```
obj.delete() # objはインスタンス
```

こんな感じでデータを自由に作って見てください！
（僕は出身地とか性別を変えて1200人作りました）

rangeの中身については

```
range(200)  
range(200, 300)  
range(300, 400)  
...
```

のように変えていくと、名前とかメールアドレスとかが被らなくていいですよ。

Personだけでなく、PersonにひもづくManagerも作っていきます。

```
dep_list = [Manager.DEP_ACCOUNTING, Manager.DEP_SALES, Manager.DEP_OTHER]

for i in range(1, 201):
    p = Person.objects.get(id=i)
    joined_date = datetime.datetime(year=2005 + i % 10, month=1, day=1)
    Manager.objects.create(person=p, department=dep_list[i % 3], joined_at=joined_date)
```

Personのオブジェクトをデータベースから取得する方法はいくつかあります（後述）が、ここではその一つのgetを使っています。

idが1~200までのPersonインスタンスそれぞれにひもづくManagerを作っています。

Workerの方も残りの1000人で同じように作りました。

```
for i in range(201, 1201):
    p = Person.objects.get(id=i)
    m = Manager.objects.get(id=1 + i % 200)
    joined_date = datetime.datetime(year=2005 + i % 10, month=1, day=1)
    Worker.objects.create(person=p, manager=m, joined_at=joined_date)
```

get関数は弱点があります。

それは、条件にあうインスタンス（オブジェクト）が見つからなかったとき、HTTPエラーではないエラー (ObjectDoesNotExist)を返してしまう点です。

つまり、変なURLを打ち込まれたとき、エラーを表示してしまい、何かと不都合です（セキュリティ的な問題も若干あります）。

ここではidがこの範囲なら必ずオブジェクトが存在するという前提の元、get関数を使いましたが、普通views.pyとかの中でオブジェクトを取得する時は、 `get_object_or_404` 関数を使います。

views.py

```
from django.shortcuts import get_object_or_404
```

```
get_object_or_404(Person, id=20)
```

のように使います。

これなら、もしオブジェクトが存在しない場合でも、HTTPの404エラーを返してくれるので、Page Not Found みたいに表示できます。（独自のpage not found画面を作って、毎回404が出るたびにそれを表示することもできます）

これなら見た目的にも、セキュリティ的にも安心です。

したがって、「必ずオブジェクトが存在する」という前提の元なら楽にかけるget関数を使ってもいいですが、それ以外の場面で一つのオブジェクトを取得する場合はget_object_or_404関数を使いましょう。

次にビューを書いて、ページを表示させていきます！

Viewを書く

viewを書く前に、urls.py を書きます。

urls.py

```
from django.conf.urls import url
from django.contrib import admin

import manager.views as manager_view

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^worker_list/', manager_view.WorkerListView.as_view(
]
```



こいつがURLにマッチするViewを見つけるんでしたね。

まずは、ワーカーのリストを表示するページを作るので、
base_url/worker_list/ というURLと WorkerListView というViewを組み合わせています！

次に、Viewを書きます。

views.py

```
from django.shortcuts import render, redirect, get_object_or_404
from django.views.generic import TemplateView

from manager.models import *
```



```
class WorkerListView(TemplateView):
    template_name = "worker_list.html"

    def get(self, request, *args, **kwargs):
        context = super(WorkerListView, self).get_context_data
        return render(self.request, self.template_name, context)
```

Viewはデータとテンプレートを混ぜて表示するのです。

ここではまずデータを取り出さず、テンプレートだけを表示させて見ます。

```
manager
  - templates
    - worker_list.html
```

という階層になるように、 templates というフォルダーと、 worker_list.html というファイルを作ってください。

こうすることで、 Djangoは worker_list.html がテンプレートであることを認識します。

get関数の中身を簡単に説明すると、 context というデータを入れる入れ物を作り、render関数で、テンプレートとcontextを混ぜる（requestも）といった感じです。

htmlファイルは以下のように簡単にしておきます。

```
worker_list.html
```

```
It Works!!
```

ページを表示してみる

まずはDjangoのサーバーを立ち上げます。

master directory から、以下のコマンドを実行してください。

```
$ python manage.py runserver 8000
```

末尾の 8000 はつけなくても大丈夫です。

（末尾でポート番号の指定ができるということを言いたかつ

ただけです！)

先ほど作ったurlでリクエストを投げて見ましょう。



これで表示できました！

これからなにをする？

ここでSkipしたのは以下の2点です。

- Viewで、データをとってくること
- HTMLを豪華にすること

ではこの2点をやっていきます。

Viewでデータベースからデータを取得する

views.py

```
def get(self, request, *args, **kwargs):
    context = super(WorkerListView, self).get_context_data

    workers = Worker.objects.all() # データベースからオブジ:
    context['workers'] = workers # 入れ物に入れる

    return render(self.request, self.template_name, contex
```

データベースを取得する時は、このように書きます。

```
workers = Worker.objects.all()
```

意味としては、「Workerモデルにひもづくオブジェクトをすべて取る」ということです。

全て、じゃなくて一部を取ることもできます。

```
workers = Worker.objects.filter(person__sex=Person.MAN)
```

とすると、性別が男性のワーカーだけを取得します。

`person__sex` というのは、`worker`の`person`属性に入っている`Person`オブジェクトの`sex`属性という意味です（言葉にするとややこしい？笑）。

`Person.MAN` のように、別のクラスの定数を参照する時はクラス名をつける必要がありますね（Pythonの基礎です！）。

また、

```
context['workers'] = workers
```

としたことによって、htmlファイルの中で workers という変数が有効になります！

（['']の中身がhtmlファイルで使えるようになる変数名です。）

Djangoテンプレートを書いてみる

ワーカーたちを表形式で表示して見ましょう。

worker_list.html

```
<!DOCTYPE html>
<html lang='ja'>
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="description" content="">
  <meta name="author" content="">

  <title>Worker List</title>
</head>
<body>
  <table>
    <thead>
```

```
<tr>
    <th>ID</th>
    <th>名前</th>
    <th>性別</th>
    <th>誕生日</th>
</tr>
</thead>
<tbody>
    {% for worker in workers %}
    <tr>
        <td>{{worker.id}}</td>
        <td>{{worker.person.name}}</td>
        <td>{{worker.person.sex}}</td>
        <td>{{worker.person.birthday}}</td>
    </tr>
    {% endfor %}
</tbody>
</table>
</body>
</html>
```

Djangoでは、HTMLファイルの中で（若干）Pythonが使えるように、テンプレート言語を用意しています。

Build in tamplate tags and filters

ここでは、

```
{% for worker in workers %}  
{% endfor %}
```

がそれに当たります！

ここで workers が使えるのは、Viewで context の中に入れたからです！

他にも、

- {% block body %}{% endblock %} のように {% block 変数 %}{% endblock %}
- {% load 変数 %}
- {% if 条件 %}{% else %}{% endif %}

などをよく使います！（以下で実際使ってみましょう！）

Bootstrapを入れる


CSSやJavaScriptはフルで書くと相当めんどくさいので、Bootstrapを入れておきます。

* Bootstrapのページ

CSSやJavaScriptをもともと書いてくれているので、楽です。
ここでは SB admin2 を使います。（汎用性かなり高いです！）

SB Admin 2

A Bootstrap admin theme, dashboard, or web app UI featuring powerful jQuery plugins for extended functionality.



Toptal matches you with top developers who are guaranteed to succeed.
ads via Carbon

Get Started >

Start Bootstrap / SB Admin 2

SB Admin v2.0

Dashboard

26

New Comments

12

New Tables

124

New Orders

13

Support Tickets

Area Chart Example

Bar Chart Example

Notifications Panel

Donut Chart Example

LIVE PREVIEW

DOWNLOAD

VIEW SOURCE

Fork

Star

4,707

Follow @SBBootstrap

Tweet

Need Help?

Feeling stuck? Need something more? You can **hire a designer** to help you with your project or to create a custom build of any theme!

Description:

SB Admin 2 Bootstrap admin theme, dashboard template, or webapp UI starter. The theme features a variety of custom jQuery plugins to add extended functionality past the built in Bootstrap UI features.

Features:

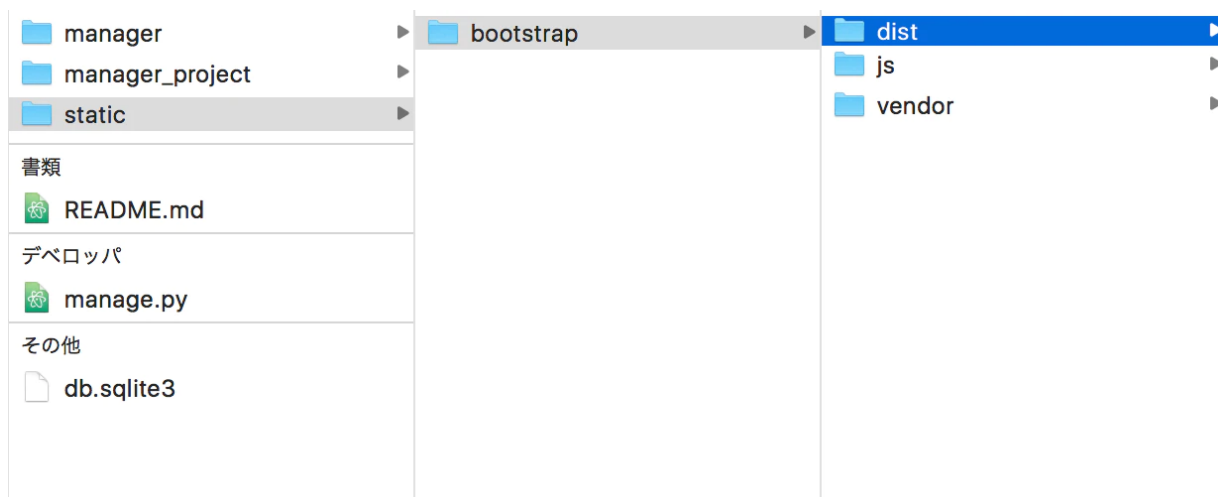
- Responsive sidebar menu with multi-level

User Contributed Versions

赤く丸したところからダウンロードします。

Zipファイルを解凍して、中のファイルをそのまま、 /static/bootstrap/ 配下にコピーしてください。

そのあと、 dist , js , vendor という名前のフォルダ以外を全部消去してください。



こんな感じになります。

これらのファイルをHTMLから読み込みます。

ですが、毎回毎回 head 要素の中にlinkとかscriptとか書いたりコピーしたりするのってめんどいですよね。

なので、baseとなるhtmlファイルを作り、それを拡張するようにします（文字にするとわかりにくい??笑）。

/manager/templates/ の配下に base.html を作ります。

base.html

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    {% load staticfiles %}
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial
    <meta name="description" content="">
    <meta name="author" content="">

    <title>{% block title %}{% endblock %}</title>

    <link href="{% static 'bootstrap/vendor/bootstrap/css/boot
    <link href="{% static 'bootstrap/vendor/metisMenu/metisMen
    <link href="{% static 'bootstrap/dist/css/sb-admin-2.css' %}
    <link href="{% static 'bootstrap/vendor/font-awesome/css/fa

    <link href="{% static 'manager/css/structure.css' %}" rel=

    <script type="text/javascript" src="{% static 'bootstrap/v
    <script src="https://ajax.googleapis.com/ajax/libs/jqueryu
```


```

<script src="{% static 'bootstrap/vendor/bootstrap/js/boot
<script src="{% static 'bootstrap/vendor/metisMenu/metisMe
<script src="{% static 'bootstrap/vendor/datatables/js/jqu
<script src="{% static 'bootstrap/vendor/datatables-plugin
<script src="{% static 'bootstrap/vendor/datatables-respon
<script src="{% static 'bootstrap/dist/js/sb-admin-2.js' %

<!-- HTML5 Shim and Respond.js IE8 support of HTML5 elemen
<!-- WARNING: Respond.js doesn't work if you view the page
<!--[if lt IE 9]>
    <script src="https://oss.maxcdn.com/libs/html5shiv/3.7
    <script src="https://oss.maxcdn.com/libs/respond.js/1.
<![endif]-->
</head>
<body>
    <div id="wrapper">
        <nav class="navbar navbar-default navbar-static-top mana
            <div class="navbar-header">
                <a class="navbar-brand">gragragrao Company</a>
            </div>
            <div class="navbar-default sidebar" role="navigation">
                <div class="sidebar-nav navbar-collapse">
                    <ul class="nav" id="side-menu">
                        <li><a href="/worker_list/"><i class="fa fa-bar-
                    </ul>
                </div>
            </div>
        </nav>

```

```
        {% block body %}
        {% endblock %}
    </div>
</body>
</html>
```



{% block title %}{% endblock %} や {% block body %}{% endblock %} は、拡張先のHTMLファイルでの同じ blockの中に入っているスクリプトをそのままそこに代入するということです。

これを拡張して `worker_list.html` を作ります！

`worker_list.html`

```
{% extends "base.html" %}
{% block title %}Worker List{% endblock %}
{% load staticfiles %}

    {% block body %}
    <div id="wrapper">
        <div id="page-wrapper">
            <div class="row">
```

```

<div class="col-lg-6 full-width margin-top-20percent">
    <div class="panel panel-default full-width">

        <div class="panel-heading">
            Edit Help
        </div>

        <div class="panel-body full-width full-height">
            <table id="worker-list-table" class="table table-striped">
                <thead>
                    <tr>
                        <th>ID</th>
                        <th>名前</th>
                        <th>性別</th>
                        <th>誕生日</th>
                    </tr>
                </thead>
                <tbody>
                    {% for worker in workers %}
                        <tr>
                            <td>{{worker.id}}</td>
                            <td>{{worker.person.name}}</td>
                            <td>{{worker.person.sex}}</td>
                            <td>{{worker.person.birthday}}</td>
                        </tr>
                    {% endfor %}
                </tbody>
            </table>
        </div>
    </div>

```

```
        </div>
    </div>
</div>
</div>
</div>
<script>
    $(document).ready(function() {
        $('#worker-list-table').DataTable({
            responsive: true,
            // sort機能の無効化
            ordering: false,
            // ページの表示件数を変更する
            displayLength: 20,
        });
    });
</script>
{% endblock %}
```

- {% extends ファイル名 %} で元となるファイルを拡張します

- {% load staticfiles %} は /static/配下に置いた静的ファイルを読み込みます。（これが無いと読み込めません！）

ただ、このままでは staticfiles ってなんぞや？って思われてしまうので、 setting.py でDjangoに正体を教えてあげましょう。

以下をsetting.pyファイルのどこかに追加してください。

setting.py


```
# Static file settings
STATIC_ROOT = os.path.join(BASE_DIR, 'assets')

STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "static"),
)
```

あとは少しCSSを整えます。

structure.css


```
/* -----  
/* general */  
/* -----  
  
.full-height {  
    height: 100%;  
}  
  
.full-width {  
    width: 100%;  
}  
  
.margin-top-20percent {  
    margin-top: 20px;  
}  
  
.no-margin {  
    margin: 0 !important;  
}
```



base.html で、このファイルを以下のように呼んでいます。

```
<link href="{% static 'manager/css/structure.css' %}" rel="sty
```



static配下にあるファイルはこんな感じで呼び出すことができます！

なので、 `structures.css` は `/static/manager/css/` 配下に置きましょう！

これでひとまず完成です！！

もし、うまくいかないとかあればコメントなどで報告していただけるとありがたいです。

ターミナルからDjangoのサーバーを立ち上げて、どんなページができているか見てみましょう！

master directory から、以下を実行してください！

```
$ python manage.py runserver
```

これで下に出てくる

```
Starting development server at http://127.0.0.1:8000/  
Quit the server with CONTROL-C.
```

の、 `http://127.0.0.1:8000/worker_list/` にブラウザから接続してみましょう！

gragragao Company

Worker一覧

Edit Help

Show 1,000 entries Search:

ID	名前	性別	誕生日
1	person0	0	Dec. 31, 1979, 3 p.m.
2	person1	0	Feb. 1, 1981, 3 p.m.
3	person2	0	March 2, 1982, 3 p.m.
4	person3	0	April 3, 1983, 3 p.m.
5	person4	0	May 4, 1984, 3 p.m.
6	person5	0	June 5, 1985, 3 p.m.
7	person6	0	July 6, 1986, 3 p.m.
8	person7	0	Aug. 7, 1987, 3 p.m.
9	person8	0	Sept. 8, 1988, 3 p.m.
10	person9	0	Oct. 9, 1989, 3 p.m.
11	person10	0	Nov. 10, 1990, 3 p.m.
12	person11	0	Dec. 11, 1991, 3 p.m.
13	person12	0	Jan. 12, 1992, 3 p.m.
14	person13	0	Feb. 13, 1993, 3 p.m.
15	person14	0	March 14, 1994, 3 p.m.
16	person15	0	April 15, 1995, 3 p.m.
17	person16	0	May 16, 1996, 3 p.m.
18	person17	0	June 17, 1997, 3 p.m.
19	person18	0	July 18, 1998, 3 p.m.
20	person19	0	Aug. 19, 1999, 3 p.m.

Showing 1 to 20 of 1,000 entries

Previous 1 2 3 4 5 ... 50 Next

こんな感じになりました！