GRID

Version 0.1

GRID Team

avr. 25, 2021

Table des matières

Bienvenue sur la documentation du GRID !

Ici, vous trouverez :

1. Le fonctionement général du projet
2. L'insfrastructure du projet
3. Comment les indicateurs sont calculés
4. Comment lire le projet
5. La documentation spécifique au moudule *data_agri* qui génère les données
6. La documentation spécifique au module *app* qui génère l'app

Documentation PDF

Attention : Le dashboard est pour le moment optimiser pour les écrans d'ordinateurs

CHAPITRE 1

Table des matières

1.1 Introduction

1.1.1 Se connecter

Pour voir le dasboard, rendez-vous sur app.grid-tech.fr .

Pour les besoins de la démonstration dans le cadre du concours FIFG, un compte test a été créé :

- **Nom d'utilisateur** : test
- **Mot de passe** : test

Astuce : Avec notre infrastructure actuelle, le chargement des pages peut praître long mais nous y travaillons.

Attention : Le dashboard est pour le moment optimiser pour les écrans d'ordinateurs

Pour plus de détail sur le déploiement *Déploiement et installation*

1.1.2 Les données

Les données d'entrée

Le GRID fonctionne à partir de 2 types de données d'entrées :

- Les données externes provenant de Météo France, Copernicus, etc
- **Les données liées à l'exploitation :**
  - Données internes rentrées par l'agriculteur par un questionaire
  - Données financières rentrées par le banquier

Données pour le PoC

Pour le PoC, afin de démontrer la capacité dynamique du dashboard, à chaque login, une partue des des données sont tirées au hasard, en particulier celles :

— Les scores RSE présentés sur la première page
— Les données financières (cf page indicateur et le module *agri_data* pour plus de détail)

Toutes les données sont ici

Pour plus de détail sur les données et leur exploitation *Indictateurs et graphiques*

1.2 Infrastructure

1.2.1 Organisation global du répertoire

Le répertoire est organisé autour de deux principaux modules :

— `app` : module générant l'application et ses rendus
— `agri_data` : module regroupant générant les données

Le fichier principal lançant l'application est situé à la racine, il s'appelle `run.py`

```
|
|-- app/                                    # L'application en elle même
|    |-- home/                              # Génération des contenus spécifiques␣
↪pas page HTML spécifiques
|    |-- base/                              # Blueprint, contient la structure de l
↪'application
|
|-- agri_data/                              # Générations des données
|
|-- ************************************************************************
```

1.2.2 Point de vue Flask App / Python

Le module `app` est organisé en 2 sous-modules :

— `home` qui sert à générer les visuels, en particulier le sous-module `content_gen`
— `base` qui sert à gérer l'authentification

Le code est ensuite commenté et précisé dans les modules *Module agri_data* et *Module app*

Le répértoire d'un point de vue Python à la structure suivante :

```
|
|-- app/                                    # L'application en elle même
|    |-- home/
|         |-- content_gen/                  # Module générant les visuels
|         |    |-- data/                    # Données externes pré-traités
|         |    |-- graph_generation.py      # Génération des graphiques
|         |    |-- index_renderer.py        # Générationd de l'index
|         |    |-- map_generation.py        # Génération des cartes
|         |    |-- questionaire.py.py       # Génération du questionaire agri
|         |-- routes.py
|
|    |-- base/
|         |-- forms.py                      # Script gérant le formulaire de login␣
↪et d'inscription
```

(suite sur la page suivante)

(suite de la page précédente)

```
|          |-- models.py                          # Script gérant la lecture de la base␣
↪de données des logins
|          |-- routes.py                          # Script gérant les actions
|          |-- util.py                            # Script gérant le hachage du mot de␣
↪passe
|
|-- agri_data/
|     |-- data_draw.py                           # Tirage aléatoire des données
|     |-- data_import.py                         # Import des données de GitHub
|     |-- *.json
|
|-- requirements.txt                             # Librairies nécessaire pour faire␣
↪fonctioner le code
|-- environment.yml                              # Environement anconda
|-- requirements-mysql.txt                       # Module nécessaire pour Mysql DMBS
|-- requirements-pqsql.txt                       # Module nécessaire pour PostgreSql DMBS
|
|-- .env                                         # Variable environnement
|-- config.py                                    # Configuration de l'application
|-- run.py                                       # Lancement de l'application
|
|-- ************************************************************************
```

1.2.3 Point de vue fronte-end / HTML

Les fichiers HTML sont organisées autour de 2 dossiers :

- **/home** : ici sont stockés les fichiers HTML des pages du dashboard
- **/base** : ici sont stockés les fichiers HTML servant de modèles pour générer les pages

```
|
|-- app/
|     |-- home/
|          |-- templates/                         # Ensemble des pages HTML
|          |     |     |-- *.html
|
|     |-- base/
|          |-- static/
|          |     |-- <css, JS, images>            # Fichiers CSS, Javascripts et images
|          |
|          |-- templates/                         # Modèles pour le rendu des pages
|               |
|               |-- includes/
|               |     |-- navigation.html        # Menu du haut
|               |     |-- sidebar.html           # Menu latéral
|               |     |-- footer.html            # Pied de page
|               |     |-- scripts.html           # Script communs aux pages HTML
|               |
|               |-- layouts/                     # Pages masters
|               |     |-- base.html              # Layout des pages
|               |
|               |-- accounts/                    # Pages authentification
|                     |-- login.html             # Page de Login
|                     |-- register.html          # Page d'inscription
|
|-- ************************************************************************
```

1.3 Indictateurs et graphiques

1.3.1 Les types de représentations

Afin de rendre compte au mieux des données, nous utilisons trois types de représentations :

— **Compteurs** : ceux-ci codés en JS représentent les 3 scores ESG sur la page d'accueil
— **Graphiques** : que ce soit des graphiques lignes ou bar ils servent à représenter l'évolution temporel d'un indicateur
— **Echelles de couleurs** : lorsque qu'un indicateur est calculé à partir d'un modèle, il est représenté sous la forme d'une échelle de couleurs comme on peut le retrouver dans la page Social avec le rayonement de l'exploitation.
— **Cartes** : ce support est utilisé pour représentées des données spatiales avec une dimension temporelle

1.3.2 Exemple d'indicateurs

Carte des feu de forêts

Sur la base des données du Climate Data Store, base de l'UE, nous avons pu exporter ces données, les traiter et les netoyer pour notre usage. Nous avons décidé de choisir les données du modèle du GIEC RCP 4.5 car représente le scenario le plus probable. Ces données ont ensuite été présentées sur une carte disponible dans Environnement.

Graph des canicules

Toujours sur la base des données du Climate Data Store, nous avons selectioner ces données représentant le nombre de jour de canicule. Il nous a semblé plus judicieux de ne représenter les jours de canciules que à l'emplacement du viticultuteur.

1.4 Déploiement et installation

1.4.1 Déploiement en ligne

Le code est stockée sur GitHub puis déployé sur Heroku pour qu'il soit accessible en ligne. Ce choix a été fait pour simplifier la création et la visualtions du PoC dans un premier temps. Cependant, à terme, l'application sera hébergée sur Google Cloud.

La conséquence principale de cela est que l'application met du temps à charger.

1.4.2 Installation en local

Si vous les souhaitez, il est possible de faire tourner l'application en local, cependant cela nécessite Python 3.x et un manager de module type pip ou anaconda. Dans la suite, nous suposerons que ces pré-requis sont remplis.

Pour utiliser l'application en local :

1. Colonez la branche principale du répértoire GitHub
2. **Créez un environnement virtuel soit avec :**
    1. pip : `python3 -m pip install -r requirements.txt`
    2. anaconda `conda env create -f environment.yml`

1.5 Module agri_data

1.5.1 agri_data package

Submodules

agri_data.data_draw module

© GRID Team, 2021

class agri_data.data_draw.**RandomDraw**

Bases : object

Cette classe télécharge les données de GitHub et les stocke en locale. Pour certains jeux de données, ils sont modifiés par un tri alétoire à chaque login

data_agri()

Télécharge et enrehistre les données liées à l'emplacement de l'agriculteur

financial_data()

Télécharge et enrehistre les données liées aux données financières Elles sont randomisées avant l'enregistrement

gauges_val()

Télécharge et enregistre les données pour générer échelles de couleurs

graph_val()

Télécharge et enregistre les données pour générer les graphs

indic_critique()

Télécharge et enregistre les données donnant les indices critiques

main()

scoring_data()

Télécharge et enregistre les données de scoring ESG. Elles sont randomisées avant l'enregistrement

stat_data()

Télécharge et enregistre les données donnant les statiques liés à la région

agri_data.data_import module

© GRID Team, 2021

class agri_data.data_import.**ReadData**(*name*)

Bases : object

Cette classe lit les données json disponible en local et retourne une dataframe

read_json()

Module contents

1.6 Module app

1.6.1 Subpackages

app.base package

Submodules

app.base.forms module

Modfied for GRID, 2021

Génère les formulaires d'inscription et connexion

class app.base.forms.**CreateAccountForm**(**args*, ***kwargs*)
Bases : flask_wtf.form.FlaskForm
email = <UnboundField(TextField, ('Email',), {'id': 'email_create', 'validators': [<wt
password = <UnboundField(PasswordField, ('Password',), {'id': 'pwd_create', 'validator
username = <UnboundField(TextField, ('Username',), {'id': 'username_create', 'validato

class app.base.forms.**LoginForm**(**args*, ***kwargs*)
Bases : flask_wtf.form.FlaskForm
password = <UnboundField(PasswordField, ('Password',), {'id': 'pwd_login', 'validators
username = <UnboundField(TextField, ('Username',), {'id': 'username_login', 'validator

app.base.models module

Modfied for GRID, 2021

Sert à lire et écrire dans la db des logins

class app.base.models.**User**(***kwargs*)
Bases : sqlalchemy.ext.declarative.api.Model, flask_login.mixins.UserMixin
email
id
password
username

app.base.models.**request_loader**(*request*)

app.base.models.**user_loader**(*id*)

app.base.routes module

Modfied for GRID, 2021

Gère les routines des connnexions et inscription

app.base.routes.**access_forbidden**(*error*)

app.base.routes.**internal_error**(*error*)

app.base.routes.**login**()

app.base.routes.**logout**()

app.base.routes.**not_found_error**(*error*)

app.base.routes.**register**()

app.base.routes.**route_default**()

app.base.routes.**shutdown**()

app.base.routes.**unauthorized_handler**()

app.base.util module

Copyright (c) 2019 - present AppSeed.us

app.base.util.**hash_pass**(*password*)
Hash a password for storing.

app.base.util.**verify_pass**(*provided_password*, *stored_password*)
Verify a stored password against one provided by user

Module contents

Copyright (c) 2019 - present AppSeed.us Modfied the GRID, 2021

app.home package

Subpackages

app.home.content_gen package

Submodules

app.home.content_gen.graph_generation module

© GRID Team, 2021

class app.home.content_gen.graph_generation.**BulletChart**(*indic*, *indic_name*)
Bases : object
Cette classe génère une échelle à 3 couleurs pour un indicateur donnée

Paramètres
— **indic** (*str*) – le code indicateur au format Ex, Sx ou Gx (où x est un int)
— **indic_name** – le nom de l'indicateur utiliser pour le titre

plot()
Les données sont importés depuis l'init

Renvoie objet json contenant le plot

Type renvoyé json

class app.home.content_gen.graph_generation.**CaniculePlot**
Bases : object
Cette classe génère le graphique des canicules dans environement. Les données sont importées directement

find_closest()
Sur la base de la localisation de la PME, recherche le point de donnée le plus proche. Ces données deviennent les variables self.lat et self.lon

main()
Fonction principale de la classe

Renvoie objet json

Type renvoyé json

plot()

Plot un graphique ligne et stocke l'object json dans self.graphjson

class app.home.content_gen.graph_generation.**FinancialChart**(**args*)

Bases : object

Cette classe génère les diagrames pour la partie finance

Paramètres ****args** – le code indicateur au format Ex, Sx ou Gx (où x est un int)

plot_bar()

Les données sont importés depuis l'init. Génère un graphique barre

Renvoie list d'object json

Type renvoyé list[json]

plot_mltpl_line()

Les données sont importés depuis l'init. Génère un graphique ligne avec 2 axes y

Renvoie list d'object json

Type renvoyé list[json]

plot_sgl_line()

Les données sont importés depuis l'init. Génère un graphique ligne

Renvoie list d'object json

Type renvoyé list[json]

class app.home.content_gen.graph_generation.**PieChart**(*indic*, *indic_name*)

Bases : object

Cette classe génère les diagrames camembert

Paramètres

— **indic** (*str*) – le code indicateur au format Ex, Sx ou Gx (où x est un int)

— **indic_name** – le nom de l'indicateur utiliser pour le titre

plot()

Les données sont importés depuis l'init

Renvoie objet json contenant le plot

Type renvoyé json

app.home.content_gen.index_renderer module

class app.home.content_gen.index_renderer.**CriticalAlert**

Bases : object

Cette classe donne les indicateurs considérés comme critique

main()

Renvoie liste de liste (1 par indicateur) contenant pour chaque la liste des indicateurs critiques

Type renvoyé list

class app.home.content_gen.index_renderer.**Scoring**

Bases : object

Cette classe donne les données nécessaires au rendus des gauges indiquant les scores ESG

bin()

Génère les intervalles autour de la valeur moyenne

main()

Renvoie liste de liste (1 par indicateur) contenant pour chaque : sa valeur, la valeur max de l'echelle, une liste avec les intervalles de couleurs

Type renvoyé list

app.home.content_gen.map_generation module

class app.home.content_gen.map_generation.**CaniculePlot**

Bases : object

Cette classe génère une heat map des canicules sur la base des données de Copernicus Les données ont été pré-traités et stockés dans le même répertoire.

main()

Fonction lançant le tout

Renvoie objet json

Type renvoyé json

plot_at_date()

Créé un carte pour un date données

Renvoie objet json

Type renvoyé json

plot_cursor()

Créé un carte pour différentes dates avec un slider temporel (dates définis dans la variable *list_date*)

Renvoie objet json

Type renvoyé json

read_json()

class app.home.content_gen.map_generation.**FirePlot**

Bases : object

Cette classe génère une carte avec un scatter plot des risques incendies sur la base des données de Copernicus Les données ont été pré-traités et stockés dans le même répertoire.

color_scale(*zmax*)

Cette fonction accomplie 2 choses en parallèle : création d'une echelle de couleur pour correpondre au Fire Index européen et trouve les valeurs centrales de chacun des intervales utilisées pour afficher l'echelle de couleur annotée

Renvoie liste de l'echelle de couleurs normé (i.e. valeurs entre 0 et 1) et liste du centre des intervales

Type renvoyé list

main()

Fonction lançant le tout

Renvoie objet json

Type renvoyé json

plot_at_date()

Créé un carte pour un date données

Renvoie objet json

Type renvoyé json

plot_cursor()

Créé un carte pour différentes dates avec un slider temporel (dates définis dans la variable *list_date*)

Renvoie objet json

Type renvoyé json

read_json()

Lecture du fichier .json et tri de l'index

app.home.content_gen.questionaire module

class app.home.content_gen.questionaire.**QuestionairesAgri**(**args*, ***kwargs*)

Bases : flask_wtf.form.FlaskForm

Cette classe génère le questionaire Flask nécessaire au rendu HTML

address = <UnboundField(TextField, ('Address',), {'validators': [<wtforms.validators.Da
age = <UnboundField(TextField, ('Age',), {'validators': [<wtforms.validators.DataRequi
autract = <UnboundField(TextField, ('autre activite',), {'validators': [<wtforms.valida
autrcult = <UnboundField(SelectField, ('autre cultures',), {'choices': [('init', 'séle
autrecertif = <UnboundField(TextField, ('autre certication',), {'validators': [<wtforms
autrequal = <UnboundField(TextField, ('autre qualite',), {'validators': [<wtforms.vali
cepage = <UnboundField(SelectMultipleField, (), {'choices': [('init', 'sélectionnez la
certif = <UnboundField(SelectField, ('certification',), {'choices': [('bio', 'label BI
etp = <UnboundField(TextField, ('etp',), {'validators': [<wtforms.validators.DataRequi
haie = <UnboundField(SelectField, ('Presence haies',), {'choices': [('init', 'sélectio
ift = <UnboundField(TextField, ('ift',), {'validators': [<wtforms.validators.DataRequi
intrant = <UnboundField(TextField, ('intrant',), {'validators': [<wtforms.validators.Da
irrig = <UnboundField(RadioField, (), {'choices': [('init', 'sélectionnez la propositi
mutu = <UnboundField(SelectMultipleField, (), {'choices': [('init', 'sélectionnez la p
name_exploit = <UnboundField(TextField, ('Nom exploitation',), {'validators': [<wtforms
qual = <UnboundField(SelectField, ('certification qualite',), {'choices': [('init', 's
sau = <UnboundField(TextField, ('sau',), {'validators': [<wtforms.validators.DataRequi
submit = <UnboundField(SubmitField, ('Enregistrer',), {})>
typecult = <UnboundField(TextField, ('type culture',), {'validators': [<wtforms.valida
typefonc = <UnboundField(SelectField, ('type de foncier',), {'choices': [('init', 'sél

app.home.content_gen.questionaire.**save_data**(*data*)

Cette fonction enregistre les données du questionaire

return dernières données rentrées pour l'affichage

Type renvoyé pandas df

Module contents

Submodules

app.home.routes module

app.home.routes.**get_segment**(*request*)

app.home.routes.**index**()

app.home.routes.**route_template**(*template*)

Module contents

Copyright (c) 2019 - present AppSeed.us Modfied the GRID, 2021

1.6.2 Module contents

Copyright (c) 2019 - present AppSeed.us Modfied the GRID, 2021

app.**configure_database**(*app*)

app.**create_app**(*config*)

app.**register_blueprints**(*app*)

app.**register_extensions**(*app*)

CHAPITRE 2

Index et recherche

— genindex
— modindex
— search

Index des modules Python

a

Index

E

F

G

H

I

L

M

N

P

Q

R

S

T

U

V