
GRID
Version 1.0

GRID Team

avr. 25, 2021

Table des matières

1	Table des matières	3
1.1	Introduction	3
1.2	Infrastructure	4
1.3	Indicateurs et graphiques	6
1.4	Déploiement et installation	6
1.5	Module agri_data	7
1.6	Module app	8
2	Index et recherche	15
	Index des modules Python	17
	Index	19

Bienvenue sur la documentation du GRID !

Ici, vous trouverez :

1. Le fonctionnement général du projet
2. L'infrastructure du projet
3. Comment les indicateurs sont calculés
4. Comment lire le projet
5. La documentation spécifique au module *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 optimisé uniquement pour les écrans d'ordinateurs

CHAPITRE 1

Table des matières

1.1 Introduction

1.1.1 Se connecter

Pour voir le dashboard, 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 paraître long mais nous travaillons pour y remédier.

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

Pour plus de détails 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'entrée :

- Les données externes provenant de Météo France, Copernicus, etc ;
- **Les données liées à l'exploitation :**
 - Données internes rentrées par l'agriculteur dans un questionnaire ;
 - Données financières provenant de l'établissement bancaire.

Données pour le PoC

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

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

Toutes les données sont disponibles [ici](#)

Pour plus de détail sur les données et leur exploitation [Indicateurs et graphiques](#)

1.1.3 Le code

Vous pouvez retrouver notre code dans son intégralité [ici](#).

1.2 Infrastructure

1.2.1 Organisation globale du répertoire

Le répertoire est organisé autour de deux principaux modules :

- *app* : module générant l'application et ses rendus
- *agri_data* : module regroupant et générant les données

Le fichier principal qui lance l'application est situé à la racine, il s'appelle *run.py*

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

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*

La structure du répertoire ,d'un point de vue Python, est la suivante :

```
|  
|-- app/                                     # L'application en elle même  
|   |-- home/                                # Module générant les visuels  
|   |-- content_gen/                          # Données externes pré-traitées  
|   |   |-- data/                             # Génération des graphiques  
|   |   |-- graph_generation.py  
|   |   |-- index_renderer.py  
|   |   |-- map_generation.py  
|   |   |-- questionnaire.py.py  
|  
|-- *****
```

(suite sur la page suivante)

(suite de la page précédente)

```

|           |-- routes.py
|
|           |-- base/
|               |-- forms.py
|               # Script gérant le formulaire de login
|               # et d'inscription
|               |-- 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 nécessaires pour faire fonctionner le code
|-- environment.yml
|   # Environnement anaconda
|-- 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 organisés 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/
|           |   |-- *.html
|           # Ensemble des pages HTML
|
|   |-- base/
|       |-- static/
|           |   |-- <css, JS, images>
|           # Fichiers CSS, Javascripts et images
|           |
|       |-- templates/
|           |   |
|           |   |-- includes/
|           |       |   |-- navigation.html
|           |       # Menu du haut
|           |       |-- sidebar.html
|           |       # Menu latéral
|           |       |-- footer.html
|           |       # Pied de page
|           |       |-- scripts.html
|           |       # Scripts communs aux pages HTML
|           |
|           |   |-- layouts/
|           |       |   |-- base.html
|           |       # Pages masters
|           |       # Layout des pages
|           |
|           |   |

```

(suite sur la page suivante)

(suite de la page précédente)

	-- accounts/	# Pages authentification
	-- login.html	# Page de Login
	-- register.html	# Page d'inscription
	-- ****	

1.3 Indicateurs 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 à barres ils servent à représenter l'évolution temporelle 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 l'onglet Social avec le rayonnement de l'exploitation.
- **Cartes** : ce support est utilisé pour représenter 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 données de l'UE, nous avons pu exporter ces données, les traiter et les nettoyer pour notre usage. Nous avons décidé de choisir les données du modèle du GIEC RCP 4.5 car celui-ci correspond au scenario le plus probable. Ces données ont ensuite été présentées sur une carte disponible dans l'onglet Environnement.

Graph des canicules

Toujours sur la base des données du [Climate Data Store](#), nous avons selectionner ces données représentant le nombre de jours de canicule. Il nous est paru plus pertinent de représenter les jours de canicule uniquement à l'emplacement du viticulteur.

1.4 Déploiement et installation

1.4.1 Déploiement en ligne

Le code est stocké 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 principale conséquence de ce choix est le temps que met l'application à charger.

1.4.2 Installation en local

Si vous le 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. Pour la suite, nous supposerons que ces pré-requis sont remplis.

Pour utiliser l’application en local :

1. Clonez la branche principale du [répertoire 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 local. Pour certains jeux de données, ils sont modifiés par un tri aléatoire à chaque login

data_agri()

Télécharge et enregistre les données liées à l’emplacement de l’agriculteur.

financial_data()

Télécharge et enregistre les données liées aux données financières. Elles sont randomisées avant l’enregistrement.

gauges_val()

Télécharge et enregistre les données pour générer les 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 RSE. 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 disponibles en locals et retourne une dataframe
read_json()
```

Module contents

1.6 Module app

1.6.1 Subpackages

app.base package

Submodules

app.base.forms module

Modified for GRID, 2021

Copyright (c) 2019 - present AppSeed.us

Génère les formulaires d'inscription et de 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', 'validator...

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

Modified for GRID, 2021

Copyright (c) 2019 - present AppSeed.us

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

Modified for GRID, 2021

Copyright (c) 2019 - present AppSeed.us

Gère les routines des connexions 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

Modified for GRID, 2021

Copyright (c) 2019 - present AppSeed.us

```
app.base.util.hash_pass(password)
    Hash mot de passe SHA-256
app.base.util.verify_pass(provided_password, stored_password)
    Verification du mot de passe par Hash
```

Module contents

Modified for GRID, 2021

Copyright (c) 2019 - present AppSeed.us

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 donné

Paramètres

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

plot()

Les données sont importées 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 la page Environnement. 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'objet json dans self.graphjson

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

Bases : object

Cette classe génère les diagrammes 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 importées depuis l'`__init__`. Génère un graphique barre

Renvoie list d'objet json

Type renvoyé list[json]

plot_mltp1_line()

Les données sont importées depuis l'`__init__`. Génère un graphique ligne avec 2 axes y

Renvoie list d'objet json

Type renvoyé list[json]

plot_sgl_line()

Les données sont importées depuis l'`__init__`. Génère un graphique ligne

Renvoie list d'objet json

Type renvoyé list[json]

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

Bases : object

Cette classe génère les diagrammes 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 importées depuis l'`__init__`

Renvoie objet json contenant le plot

Type renvoyé json

app.home.content_gen.index_renderer module

© GRID Team, 2021

class app.home.content_gen.index_renderer.CriticalAlert

Bases : object

Cette classe donne la liste des indicateurs considérés comme critique.

main()

Renvoie liste de listes (une 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 rendu des gauges indiquant les scores ESG

bin()

Génère les intervalles autour de la valeur moyenne

main()

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

Type renvoyé list

app.home.content_gen.map_generation module

© GRID Team, 2021

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-traitées et stockées dans le même répertoire.

main()

Fonction lançant le tout

Renvoie objet json

Type renvoyé json

plot_at_date()

Crée un carte pour un date données

Renvoie objet json

Type renvoyé json

plot_cursor()

Crée un carte pour différentes dates avec un slider temporel (dates définies 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-traitées et stockées dans le même répertoire.
    color_scale(zmax)
        Cette fonction accomplit 2 choses en parallèle : création d'une échelle de couleurs pour correspondre au
        Fire Index européen et trouve les valeurs centrales de chacun des intervalles utilisés pour afficher l'échelle
        de couleur annotée
        Renvoie liste de l'échelle de couleurs normée (i.e. valeurs entre 0 et 1) et liste du centre des
        intervalles
        Type renvoyé list

main()
    Fonction lançant le tout
    Renvoie objet json
    Type renvoyé json

plot_at_date()
    Crée une carte pour une date donnée
    Renvoie objet json
    Type renvoyé json

plot_cursor()
    Crée une carte pour différentes dates avec un slider temporel (dates définies 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.questionnaire module

© GRID Team, 2021

```
class app.home.content_gen.questionnaire.QuestionnairesAgri(*args, **kwargs)
    Bases : flask_wtf.form.FlaskForm
    Cette classe génère le questionnaire Flask nécessaire au rendu HTML
    address = <UnboundField(TextField, ('Address',), {})>
    age = <UnboundField(TextField, ('Age',), {})>
    autract = <UnboundField(TextField, ('autre activité',), {})>
    autrcult = <UnboundField(SelectField, ('autre cultures',), {'choices': [('init', 'sélectionnez la culture'), ('autre', 'autre')]})>
    autrecertif = <UnboundField(TextField, ('autre certification',), {})>
    autrequal = <UnboundField(TextField, ('autre qualité',), {})>
    cepage = <UnboundField(SelectMultipleField, (), {'choices': [('init', 'sélectionnez la culture'), ('autre', 'autre')]})>
    certif = <UnboundField(SelectField, ('certification',), {'choices': [('bio', 'label BIO'), ('non bio', 'non BIO')]})>
    etp = <UnboundField(TextField, ('etp',), {})>
    haie = <UnboundField(SelectField, ('Présence haies',), {'choices': [('init', 'sélectionnez la réponse'), ('oui', 'Oui'), ('non', 'Non')]})>
    ift = <UnboundField(TextField, ('ift',), {})>
    intrant = <UnboundField(TextField, ('intrant',), {})>
    irrig = <UnboundField(RadioField, (), {'choices': [('init', 'sélectionnez la proposition'), ('oui', 'Oui'), ('non', 'Non')]})>
```

```

mutu = <UnboundField(SelectMultipleField, (), {'choices': [('init', 'sélectionnez la p
name_exploit = <UnboundField(TextField, ('Nom exploitation',), {})>
qual = <UnboundField(SelectField, ('certification qualite',), {'choices': [('init', 's
sau = <UnboundField(TextField, ('sau',), {})>
submit = <UnboundField(SubmitField, ('Enregistrer',), {})>
typecult = <UnboundField(TextField, ('type culture',), {})>
typefonc = <UnboundField(SelectField, ('type de foncier',), {'choices': [('init', 'sélé
app.home.content_gen.questionnaire.save_data(data)
Cette fonction enregistre les données du questionnaire
    return dernières données rentrées pour l'affichage
Type renvoyé pandas df

```

Module contents

Submodules

app.home.routes module

Modified for GRID, 2021

Copyright (c) 2019 - present AppSeed.us

```

app.home.routes.get_segment(request)
app.home.routes.index()
app.home.routes.route_template(template)

```

Module contents

Modified for GRID, 2021

Copyright (c) 2019 - present AppSeed.us

1.6.2 Module contents

Modfied for GRID, 2021

Copyright (c) 2019 - present AppSeed.us

```

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

agri_data, 8
agri_data.data_draw, 7
agri_data.data_import, 8
app, 13
app.base, 9
app.base.forms, 8
app.base.models, 8
app.base.routes, 9
app.base.util, 9
app.home, 13
app.home.content_gen, 13
app.home.content_gen.graph_generation,
 10
app.home.content_gen.index_renderer, 11
app.home.content_gen.map_generation, 11
app.home.content_gen.questionnaire, 12
app.home.routes, 13

Index

A

access_forbidden () (dans le module `app.base.routes`), 9
address (attribut `app.home.content_gen.questionnaire.QuestionnairesAgri`), 12
age (attribut `app.home.content_gen.questionnaire.QuestionnairesAgri`), `app.home.content_gen.questionnaire.QuestionnairesAgri`), 12
agri_data
 module, 8
agri_data.data_draw
 module, 7
agri_data.data_import
 module, 8
app
 module, 13
app.base
 module, 9
app.base.forms
 module, 8
app.base.models
 module, 8
app.base.routes
 module, 9
app.base.util
 module, 9
app.home
 module, 13
app.home.content_gen
 module, 13
app.home.content_gen.graph_generation
 module, 10
app.home.content_gen.index_renderer
 module, 11
app.home.content_gen.map_generation
 module, 11
app.home.content_gen.questionnaire
 module, 12
app.home.routes
 module, 13

autract (attribut `app.home.content_gen.questionnaire.QuestionnairesAgri`), 12
autrcult (attribut `app.home.content_gen.questionnaire.QuestionnairesAgri`),
 autrecertif (attribut
 autrequal (attribut `app.home.content_gen.questionnaire.QuestionnairesAgri`), 12
 B

bin () (méthode `app.home.content_gen.index_renderer.Scoring`), 11
BulletChart (classe dans `app.home.content_gen.graph_generation`), 10

C

CaniculePlot (classe dans `app.home.content_gen.graph_generation`), 10
CaniculePlot (classe dans `app.home.content_gen.map_generation`), 11
cepage (attribut `app.home.content_gen.questionnaire.QuestionnairesAgri`), 12
certif (attribut `app.home.content_gen.questionnaire.QuestionnairesAgri`), 12
color_scale () (méthode `app.home.content_gen.map_generation.FirePlot`), 12
configure_database () (dans le module `app`), 13
create_app () (dans le module `app`), 13
CreateAccountForm (classe dans `app.base.forms`), 8
CriticalAlert (classe dans `app.home.content_gen.index_renderer`), 11

D

data_agri () (méthode `agri_data.data_draw.RandomDraw`), 7

E

email (attribut app.base.forms.CreateAccountForm), 8
 email (attribut app.base.models.User), 8
 etp (attribut app.home.content_gen.questionnaire.QuestionnairesAgri),
 12

F

financial_data() (méthode
 agri_data.data_draw.RandomDraw), 7
 FinancialChart (classe dans
 app.home.content_gen.graph_generation),
 10
 find_closest() (méthode
 app.home.content_gen.graph_generation.CaniculePlot),
 10
 FirePlot (classe dans
 app.home.content_gen.map_generation),
 12

G

gauges_val() (méthode
 agri_data.data_draw.RandomDraw), 7
 get_segment () (dans le module app.home.routes), 13
 graph_val() (méthode
 agri_data.data_draw.RandomDraw), 7

H

haie (attribut app.home.content_gen.questionnaire.QuestionnairesAgri),
 12
 hash_pass () (dans le module app.base.util), 9

I

id (attribut app.base.models.User), 8
 ift (attribut app.home.content_gen.questionnaire.QuestionnairesAgri),
 12
 index () (dans le module app.home.routes), 13
 indic_critique() (méthode
 agri_data.data_draw.RandomDraw), 7
 internal_error() (dans le module
 app.base.routes), 9
 intrant (attribut app.home.content_gen.questionnaire.QuestionnairesAgri),
 12
 irrig (attribut app.home.content_gen.questionnaire.QuestionnairesAgri),
 12

L

login () (dans le module app.base.routes), 9
 LoginForm (classe dans app.base.forms), 8
 logout () (dans le module app.base.routes), 9

M

main () (méthode agri_data.data_draw.RandomDraw),
 7

main () (méthode app.home.content_gen.graph_generation.CaniculePlot),
 10
 main () (méthode app.home.content_gen.index_renderer.CriticalAlert),
 11
 main () (méthode app.home.content_gen.index_renderer.Scoring),
 11
 main () (méthode app.home.content_gen.map_generation.CaniculePlot),
 11
 main () (méthode app.home.content_gen.map_generation.FirePlot),
 12
 module
 agri_data, 8
 agri_data.data_draw, 7
 agri_data.data_import, 8
 app, 13
 app.base, 9
 app.base.forms, 8
 app.base.models, 8
 app.base.routes, 9
 app.base.util, 9
 app.home, 13
 app.home.content_gen, 13
 app.home.content_gen.graph_generation,
 10
 app.home.content_gen.index_renderer,
 11
 app.home.content_gen.map_generation,
 11
 app.home.content_gen.questionnaire,
 12
 app.home.routes, 13
 mutu (attribut app.home.content_gen.questionnaire.QuestionnairesAgri),
 12

N

name_exploit (attribut
 app.home.content_gen.questionnaire.QuestionnairesAgri),
 13
 not_found_error () (dans le module
 app.base.routes), 9

P

password (attribut app.base.forms.CreateAccountForm),
 8
 password (attribut app.base.forms.LoginForm), 8
 password (attribut app.base.models.User), 8
 PieChart (classe dans
 app.home.content_gen.graph_generation),
 10
 plot () (méthode app.home.content_gen.graph_generation.BulletChart),
 10
 plot () (méthode app.home.content_gen.graph_generation.CaniculePlot),
 10

```

plot() (méthode app.home.content_gen.graph_generation.PieChart), a () (dans le module
11 app.home.content_gen.questionnaire), 13
plot_at_date() (méthode Scoring (classe dans
app.home.content_gen.map_generation.CaniculePlot), app.home.content_gen.index_renderer), 11
11 scoring_data() (méthode
plot_at_date() (méthode agri_data.data_draw.RandomDraw), 7
app.home.content_gen.map_generation.FirePlot), shutdown () (dans le module app.base.routes), 9
12 stat_data() (méthode
plot_bar() (méthode agri_data.data_draw.RandomDraw), 7
app.home.content_gen.graph_generation.FinancialChart), (attribut app.home.content_gen.questionnaire.QuestionnairesAgri),
10 13
plot_cursor() (méthode
app.home.content_gen.map_generation.CaniculePlot),
11 typecult (attribut app.home.content_gen.questionnaire.QuestionnairesAgr
plot_cursor() (méthode 13
app.home.content_gen.map_generation.FirePlot), typefond (attribut app.home.content_gen.questionnaire.QuestionnairesAgr
12 13
plot_mltpl_line() (méthode
app.home.content_gen.graph_generation.FinancialChart),
10 unauthorized_handler () (dans le module
plot_sgl_line() (méthode app.base.routes), 9
app.home.content_gen.graph_generation.FinancialChart (classe dans app.base.models), 8
10 user_loader () (dans le module app.base.models), 9
username (attribut app.base.forms.CreateAccountForm),
8
Q
qual (attribut app.home.content_gen.questionnaire.QuestionnairesAgri) (attribut app.base.forms.LoginForm), 8
13 username (attribut app.base.models.User), 8
QuestionnairesAgri (classe dans
app.home.content_gen.questionnaire), 12
V
R
RandomDraw (classe dans agri_data.data_draw), 7
read_json() (méthode
agri_data.data_import.ReadData), 8
read_json() (méthode
app.home.content_gen.map_generation.CaniculePlot),
11
read_json() (méthode
app.home.content_gen.map_generation.FirePlot),
12
ReadData (classe dans agri_data.data_import), 8
register() (dans le module app.base.routes), 9
register_blueprints() (dans le module app), 13
register_extensions() (dans le module app), 13
request_loader() (dans le module
app.base.models), 8
route_default() (dans le module app.base.routes),
9
route_template() (dans le module
app.home.routes), 13
S
sau (attribut app.home.content_gen.questionnaire.QuestionnairesAgri),
13

```