camptocamp
INNOVATIVE SOLUTIONS
BY OPEN SOURCE EXPERTS

Documentation technique de GetItFixed

Installation, configuration orientée déploiement et référence technique

Documentation générée · 2026-05-07

1 Vue d’ensemble

GetItFixed est une application web basée sur Pyramid pour signaler, modérer et résoudre des problèmes géolocalisés tels que des nids-de-poule, des lampadaires défectueux, des dépôts sauvages ou d’autres incidents de service public.

Les citoyens créent des signalements sur une carte, joignent éventuellement des photos et reçoivent par e-mail un lien privé de suivi. Les administrateurs examinent les signalements entrants, les rendent publics ou privés, mettent à jour leur statut, les commentent et les clôturent une fois résolus.

Cette documentation a été rédigée à partir du dépôt `camptocamp/getitfixed`, commit `a07f690` daté du 2026-04-30.

1.1 Fonctionnalités principales

- Carte publique affichant les signalements validés et non privés.
- Formulaire public de soumission de signalement avec catégorie, type, géométrie, description, texte de localisation, photos et coordonnées du déclarant.
- Vue privée pour le déclarant accessible via un hash de signalement de type UUID.
- Interface d'administration pour la modération et le suivi des signalements.
- Chronologie des événements attachée à chaque signalement.
- Notifications par e-mail pour les nouveaux signalements et les mises à jour.
- Routage des e-mails administratifs basé sur les catégories.
- Icônes personnalisées par catégorie et prise en charge d'une icône par défaut.
- Paramètres de carte, projections et couches de fond configurables.
- Stockage PostgreSQL/PostGIS.
- Sessions adossées à Redis.
- Pile de développement Docker Compose.
- Prise en charge de la localisation en anglais, français et allemand.

1.2 Interfaces et routes

GetItFixed expose trois interfaces principales :

Interface	Route par défaut	Objectif
Public	`/getitfixed/issues`	Parcourir les signalements publics et soumettre un nouveau signalement.
Déclarant privé	`/getitfixed_private/issues/<hash>`	Page de suivi du déclarant accessible depuis les liens e-mail.
Admin	`/getitfixed_admin/issues`	Modérer, mettre à jour et résoudre les signalements.

Il existe également une route historique de redirection privée :

```
/getitfixed/private/issues/<hash>
```

Elle redirige vers l'URL actuelle de suivi privé du déclarant.

1.3 Pile technologique

- Python 3.12+
- Pyramid
- SQLAlchemy
- GeoAlchemy2
- Alembic
- PostgreSQL avec PostGIS
- Sessions Redis via `beaker_redis`
- c2cgeoform pour les formulaires CRUD, les cartes et les points d'accès GeoJSON
- Colander / Deform pour les formulaires pilotés par schéma
- Templates Jinja2
- Bootstrap 3, bootstrap-table, jQuery
- OpenLayers via c2cgeoform
- Catalogues de traduction Lingua/Babel
- Waitress en production
- Docker Compose pour le développement local et les services de démonstration

1.4 Architecture de l'application

Le paquet Python principal est `getitfixed`.

Modules importants :

Chemin	Rôle
`getitfixed/__init__.py`	Fabrique d'application Pyramid, chargement de configuration et inclusion de paquets.
`getitfixed/routes.py`	Enregistrement des routes publiques, privées et

	admin.
`getitfixed/models/getitfixed.py`	Modèles SQLAlchemy et métadonnées de formulaires.
`getitfixed/views/public/`	Vues de carte publique et de soumission de signalement.
`getitfixed/views/private/`	Vues de suivi du déclarant et commentaires du déclarant.
`getitfixed/views/admin/`	Vues admin de liste, d'édition, d'événement et de photo.
`getitfixed/emails/email_service.py`	Envoi d'e-mails SMTP.
`getitfixed/static/`	CSS, JavaScript, icônes et ressources navigateur.
`getitfixed/templates/`	Templates Jinja2.
`getitfixed/alembic/`	Migrations de base de données.
`vars.yaml`	Source principale de personnalisation du projet.
`development.ini` / `production.ini`	Paramètres d'application Pyramid/PasteDeploy.

1.5 Modèle de données

GetItFixed stocke les données dans le schéma PostgreSQL `getitfixed` par défaut.

1.5.1 Category

Une catégorie est une famille de signalements de haut niveau.

Champs :

- `id`
- `label_fr`
- `label_en`
- `email`
- `icon`

L'e-mail de catégorie est important : il est utilisé comme destination des notifications administratives lorsqu'un nouveau signalement est soumis dans cette catégorie, et lorsqu'un déclarant ajoute un commentaire de suivi.

L'icône de catégorie est utilisée dans la légende de la carte et sur les marqueurs de signalement. Si elle est vide, l'icône par défaut configurée est utilisée.

1.5.2 Type

Un type est une sous-catégorie. Chaque signalement est lié à un type.

Champs :

- `id`
- `label_fr`
- `label_en`
- `category_id`
- `wms_layer`

`wms_layer` est facultatif. Lorsqu'il est présent, le front-end peut ajouter une couche WMS associée au type de signalement sélectionné.

1.5.3 Issue

Un signalement est l'objet métier central.

Champs :

- `id`
- `hash`
- `request_date`
- `type_id`
- `status`
- `description`
- `localisation`
- `geometry`, stockée comme un `POINT` en EPSG:4326

- `firstname`
- `lastname`
- `phone`
- `email`
- `private`
- `photos` liées
- `events` liés

L'interface publique identifie les signalements publics existants par `id` numérique. Les liens de suivi privés et admin utilisent `hash`, généré avec des valeurs de type UUID et plus sûr à envoyer par e-mail.

1.5.4 Photo

Les photos sont stockées dans la base de données.

Champs :

- `id`
- `filename`
- `data`
- `hash`
- `issue_id`

Les vues admin et privées de téléchargement de photos utilisent le hash de la photo comme identifiant.

1.5.5 Event

Les événements forment la chronologie du signalement.

Champs :

- `id`
- `issue_id`
- `status`
- `date`
- `comment`
- `private`
- `author`, soit `customer`, soit `admin`

Les événements sont utilisés pour mettre à jour le statut du signalement et pour échanger des commentaires entre le déclarant et l'administrateur.

1.6 Statuts

Le workflow de statut est :

```
new -> validated -> in_progress -> waiting_for_reporter -> resolved
```

Valeurs internes disponibles :

Statut	Signification
`new`	Le signalement a été soumis mais n'est pas encore public.
`validated`	L'administrateur a accepté le signalement.
`in_progress`	Le signalement est en cours de traitement.
`waiting_for_reporter`	Des informations supplémentaires sont requises de la part du déclarant.
`resolved`	Le signalement est clôturé.

La carte publique masque les signalements dont le statut est `new` et les signalements marqués `private`.

2 Installation

2.1 Prérequis

Pour une installation locale basée sur Docker :

- Git
- Docker

- Docker Compose
- GNU Make

Pour le développement direct hors Docker, le paquet cible Python 3.12 ou plus récent. Docker reste la voie de développement principale dans le dépôt.

2.2 Cloner le dépôt

```
git clone https://github.com/camptocamp/getitfixed.git
cd getitfixed
```

2.3 Démarrer avec les données de démonstration

Le chemin le plus rapide est :

```
make meacoffee
```

Cette cible construit les images Docker, démarre les services, exécute les migrations de base de données, charge les données de démonstration et suit les journaux de l'application.

Ouvrir :

- Interface publique : http://localhost:8080/getitfixed/issues
- Interface admin : http://localhost:8080/getitfixed_admin/issues
- Webmail de démonstration : http://localhost:8082/webmail/?_task=mail&_mbox=INBOX

2.4 Démarrer avec une base de données vide

La documentation amont mentionne `make meadeca` pour une base de données vide. Dans la révision du dépôt examinée, la cible Makefile visible est `meacoffee` ; si vous avez besoin d'une base vide, lancez la pile sans charger les données de test, ou adaptez la séquence Makefile/init en conséquence.

Une séquence manuelle pratique est :

```
make build
docker compose rm --stop --force getitfixed
docker compose up -d
docker compose exec getitfixed alembic -n getitfixed upgrade head
```

2.5 Services Docker Compose

La pile de développement définit ces services :

Service	Objectif
`db`	Base de données PostgreSQL/PostGIS.
`redis`	Stockage des sessions Redis.
`getitfixed`	Application web Pyramid, servie par `pserve`.
`smtp`	Récepteur SMTP utilisé pour la capture locale/de démonstration des e-mails.
`courier-imap`	Serveur IMAP pour les e-mails de démonstration.
`webmail`	Accès navigateur aux e-mails de démonstration capturés.
`db_tests`	Base de données de test.
`test`	Lanceur de tests d'acceptation Pytest.

2.6 Cibles Make utiles

Cible	Objectif
`make help`	Afficher les cibles disponibles.
`make meacoffee`	Construire, démarrer, initialiser les données de démonstration et afficher les journaux.
`make up`	Construire et démarrer les conteneurs.
`make build`	Construire les fichiers d'exécution et les images Docker.

`make initdb`	Exécuter les migrations et charger les données de test.
`make test`	Exécuter les tests d'acceptation.
`make check`	Exécuter les vérifications de formatage et de lint.
`make black`	Formater le code Python avec Black.
`make docs`	Construire la documentation Sphinx originale.
`make psql`	Ouvrir un shell PostgreSQL dans le conteneur de base de données.
`make pshell`	Ouvrir un shell Pyramid.
`make clean`	Supprimer les fichiers de traduction générés.
`make cleanall`	Arrêter/supprimer les conteneurs, le `.env` généré et les images Docker.
`make venv`	Créer un environnement virtuel Python local pour l'utilisation avec un IDE.

2.7 Migrations de base de données

Exécuter les migrations dans le conteneur de l'application :

```
docker compose exec getitfixed alembic -n getitfixed upgrade head
```

Créer une nouvelle révision :

```
docker compose run --rm --user "$(id -u)" getitfixed \
  alembic -c /app/alembic.ini -n getitfixed revision --autogenerate -m "Revision name"
```

2.8 Données de test

Charger les catégories, types et signalements de démonstration :

```
docker compose exec getitfixed getitfixed_setup_test_data getitfixed://development.ini#app
```

Le script de données de test crée des catégories, des types et 100 signalements d'exemple lorsque les tables correspondantes sont vides.

3 Configuration

La configuration est répartie entre les fichiers PasteDeploy `.ini`, les variables d'environnement et la configuration YAML générée.

3.1 Paramètres PasteDeploy

Fichiers principaux :

- `development.ini`
- `production.ini`
- `tests.ini`

La section de l'application de production utilise :

```
[app:app]
use = egg:getitfixed
pyramid.default_locale_name = en
pyramid.available_languages = en fr de
pyramid.includes =
    pyramid_tm

sqlalchemy.url = postgresql://%(PGUSER)s:%(PGPASSWORD)s@%(PGHOST)s:%(PGPORT)s/%(PGDATABASE)s

session.type = ext:redis
session.url = redis://redis:6379/0

app.cfg = config.yaml
```

La fabrique d'application charge `app.cfg`, puis fusionne la configuration YAML générée dans les paramètres Pyramid.

3.2 Variables d'environnement

Le Makefile et la configuration Docker Compose s'appuient sur ces variables :

Variable	Défaut	Objectif
`DOCKER_BASE`	`camptocamp/getitfixed`	Préfixe de l'image Docker.
`DOCKER_TAG`	`latest`	Tag de l'image Docker.
`DOCKER_PORT`	`8080`	Port hôte mappé sur le port conteneur 8080.
`PGHOST`	`db`	Hôte PostgreSQL.
`PGHOST_SLAVE`	`db`	Hôte PostgreSQL secondaire/lecture, si utilisé.
`PGPORT`	`5432`	Port PostgreSQL.
`PGDATABASE`	`getitfixed`	Nom de la base de données.
`PGUSER`	`getitfixed`	Utilisateur de base de données.
`PGPASSWORD`	`getitfixed`	Mot de passe de base de données.
`PROXY_PREFIX`	vide	Préfixe d'URL utilisé par le filtre proxy-prefix de PasteDeploy.
`DEVELOPMENT`	`TRUE`	Sélectionne le chemin ini de développement ou de production dans `.env.mako`.

3.3 `vars.yaml` et `config.yaml`

`vars.yaml` est la source de configuration maintenue par l'humain. `config.yaml` est généré à partir de celui-ci.

La règle Makefile est :

```
config.yaml: vars.yaml
    c2c-template --vars vars.yaml --get-config config.yaml project smtp getitfixed
```

Le générer avec :

```
make config.yaml
```

Les clés de premier niveau importantes sont :

```yaml
vars:
  project: getitfixed

  smtp:
    host: smtp

  getitfixed:
    default_icon: "static://getitfixed:static/icons/cat-default.png"
    map: {}
    admin_new_issue_email: {}
    new_issue_email: {}
    update_issue_email: {}
    resolved_issue_email: {}
```

3.4 Paramètres e-mail

La livraison des e-mails est gérée par `getitfixed/emails/email_service.py`.

Le service lit :

- `smtp`, pour les paramètres de connexion ;
- `getitfixed.<template_name>`, pour l'expéditeur, le sujet et le corps.

Si `smtp` est absent ou vide, l'application journalise un avertissement et n'envoie pas d'e-mail.

3.4.1 Configuration SMTP

Exemple :

```yaml
vars:
  smtp:
```

```
    host: smtp.example.org
    ssl: false
    starttls: true
    user: getitfixed@example.org
    password: change-me
```

Champs :

Champ	Obligatoire	Signification
`host`	oui	Nom d'hôte du serveur SMTP.
`ssl`	non	Utiliser `SMTP_SSL` lorsque vrai.
`starttls`	non	Appeler `starttls()` après la connexion lorsque vrai.
`user`	non	Nom d'utilisateur SMTP. Aucune authentification n'est tentée lorsqu'il est absent ou vide.
`password`	si `user` est défini	Mot de passe SMTP.

L'implémentation actuelle se connecte sans port explicite. Le port SMTP par défaut de Python est donc utilisé sauf si le code est étendu.

3.4.2 Templates d'e-mail

Quatre templates sont configurés sous `vars.getitfixed`.

3.4.2.1 `admin_new_issue_email`

Envoyé à l'administrateur de catégorie lorsqu'un déclarant soumet un nouveau signalement.

Destination :

```
issue.category.email
```

Variables typiques disponibles dans le corps :

- `{username}`
- `{issue}`
- `{issue-link}`

Exemple :

```
admin_new_issue_email:
  email_from: info@example.org
  email_subject: A new issue has been created
  email_body: |
    A new issue has been submitted.
    You can review it here: {issue-link}
```

3.4.2.2 `new_issue_email`

Envoyé au déclarant après la soumission réussie du signalement.

Destination :

```
issue.email
```

Exemple :

```
new_issue_email:
  email_from: info@example.org
  email_subject: Issue declaration confirmation
  email_body: |
    Hello {username},

    We confirm that we have received your issue at {issue.localisation}.

    You can follow it here: {issue-link}
```

3.4.2.3 `update_issue_email`

Envoyé lorsqu'un commentaire de suivi public ou une mise à jour de statut doit notifier l'autre partie.

Destinations typiques :

- e-mail du déclarant, lorsqu'un administrateur publie un événement non privé ;
- e-mail de catégorie, lorsqu'un déclarant publie un commentaire depuis l'interface privée.

Exemple :

```
update_issue_email:
  email_from: info@example.org
  email_subject: Status update for an issue
  email_body: |
    Hello {username},

    The issue {issue.hash} created on {issue.request_date} is now {issue.status_en}.

    {event.comment}

    Follow it here: {issue-link}
```

3.4.2.4 `resolved_issue_email`

Configuré pour les notifications de résolution au déclarant.

Exemple :

```
resolved_issue_email:
  email_from: info@example.org
  email_subject: Issue resolved
  email_body: |
    Hello {username},

    The issue {issue.hash} created on {issue.request_date} has now been resolved.
```

Note d'implémentation : dans la révision examinée, la vue admin des événements affecte `issue.status = event_status` avant de vérifier si le statut a changé vers `resolved`. Cela rend la branche spécifique d'e-mail de résolution inaccessible telle qu'écrite. Le chemin de notification générique de mise à jour peut toujours s'appliquer pour les événements admin non privés. Si les e-mails de résolution sont critiques pour le métier, vérifiez et ajustez `getitfixed/views/admin/events.py`.

3.4.3 Formatage des templates

Les corps d'e-mail sont des chaînes de format Python. Les valeurs sont interpolées avec :

```
template["email_body"].format(template, *template_args, **template_kwargs)
```

Utilisez des accolades simples pour les variables, par exemple `{issue-link}` ou `{issue.status_en}`.

Comme certaines clés contiennent des traits d'union, comme `issue-link`, l'implémentation les transmet via `template_kwargs` ; conservez le modèle existant lors de l'ajout de nouveaux templates.

3.4.4 Extraction des traductions

Les sujets et corps d'e-mail sont extraits pour traduction par `getitfixed/lingua_extractor.py`.

Les chemins extraits sont :

- `getitfixed.admin_new_issue_email.email_subject`
- `getitfixed.admin_new_issue_email.email_body`
- `getitfixed.new_issue_email.email_subject`
- `getitfixed.new_issue_email.email_body`
- `getitfixed.update_issue_email.email_subject`
- `getitfixed.update_issue_email.email_body`
- `getitfixed.resolved_issue_email.email_subject`
- `getitfixed.resolved_issue_email.email_body`

Le `vars.yaml` par défaut marque les corps comme `no_interpreted`, ce qui préserve le formatage littéral lors de l'étape de templating.

3.5 Configuration de la carte

La configuration de la carte se trouve sous `vars.getitfixed.map`.

Le code du modèle par défaut fusionne cette configuration avec les valeurs par défaut de c2cgeoform et force la prise en charge mobile :

```
_map_config = {
    **default_map_settings,
    **{"mobile": True},
```

```
    **_getitfixed_config.get("map", {}),
}
```

Clés courantes :

Clé	Objectif
`srid`	SRID utilisé par la sérialisation/désérialisation du widget de carte.
`projections`	Définitions de projections enregistrées dans le navigateur.
`baseLayers`	Définitions de couches OpenLayers/c2cgeoform.
`view`	Paramètres de projection, d'emprise, de centre ou de zoom.
`fitMaxZoom`	Zoom maximal lors de l'ajustement sur une géométrie de signalement existante.

Exemple pour WMTS EPSG:2056 :

```yaml
getitfixed:
  map:
    srid: 2056
    projections:
      - code: "EPSG:2056"
        definition: "+proj=somerc +lat_0=46.95240555555556 +lon_0=7.439583333333333 +k_0=1 +x_0=2600000
         +y_0=1200000 +ellps=bessel +units=m +no_defs"
    baseLayers:
      - type_: "WMTS"
        url:
         "https://example.org/tiles/1.0.0/{{Layer}}/default/{{TileMatrixSet}}/{{TileMatrix}}/{{TileRow}}/{{TileCol}}.png"

        requestEncoding: "REST"
        layer: "map"
        matrixSet: "epsg2056_005"
        style: "default"
        projection: "EPSG:2056"
    view:
      projection: "EPSG:2056"
      initialExtent: [2488941, 1077631, 2829623, 1295254]
    fitMaxZoom: 10
```

Lors de l'écriture d'URL dans `vars.yaml`, échappez les accolades comme requis par le traitement `c2c-template` utilisé dans ce projet. Les exemples du dépôt utilisent des accolades doublées ou quadruplées lorsque nécessaire.

4 Personnalisation

4.1 Personnalisation de la mise en page

La mise en page par défaut est :

```
getitfixed:templates/layout.jinja2
```

La surcharger sous `vars.getitfixed.layout` :

```yaml
vars:
  getitfixed:
    layout: mypackage:templates/getitfixed/layout.jinja2
```

Une mise en page personnalisée peut étendre celle par défaut et surcharger des blocs Jinja :

```
{% extends "getitfixed:templates/layout.jinja2" %}

{% block title %}
<title>{{ _('My City / GetItFixed') }}</title>
<link rel="shortcut icon" href="{{ request.static_url('mypackage:static/favicon.ico') }}">
{% endblock title %}

{% block style %}
<link href="{{ request.static_url('mypackage:static/getitfixed.css') }}" rel="stylesheet">
{% endblock style %}

{% block header %}
<header class="container">
  <h3 class="title">{{ _('My City / GetItFixed') }}</h3>
</header>
```

```
{% endblock header %}

{% block footer %}
<footer class="footer text-muted">
  <div class="container">
    <p>Contact: <a href="https://example.org/contact">Contact form</a></p>
  </div>
</footer>
{% endblock footer %}
```

Les blocs par défaut disponibles incluent :

- `title`
- `style`
- `header`
- `content`
- `footer`
- `scripts`

4.2 Icônes

Les icônes sont configurées à deux niveaux :

1. L'icône globale par défaut.
2. L'icône attachée à chaque catégorie.

4.2.1 Icône par défaut

Définir l'icône par défaut dans `vars.yaml` :

```yaml
vars:
  getitfixed:
    default_icon: "static://getitfixed:static/icons/cat-default.png"
```

Lorsqu'une catégorie n'a pas d'icône, cette valeur par défaut est utilisée.

4.2.2 Icônes de catégories

Le champ de base de données `category.icon` stocke la définition de l'icône.

L'application prend en charge deux formes d'URL :

Forme	Exemple	Comportement
URL statique Pyramid	`static://getitfixed:static/icons/gif-green.png`	Convertie avec `request.static_url`.
URL absolue	`https://example.org/icons/tree.png`	Utilisée telle quelle.

La forme statique est analysée par `getitfixed/url.py` :

```
static://<package>:<static-root>/<path>
```

Exemples :

```sql
INSERT INTO getitfixed.category (label_fr, label_en, email, icon)
VALUES (
  'Voirie',
  'Roads',
  'roads@example.org',
  'static://getitfixed:static/icons/gif-red.png'
);
```

```sql
INSERT INTO getitfixed.category (label_fr, label_en, email, icon)
VALUES (
  'Parcs',
  'Parks',
  'parks@example.org',
  'https://static.example.org/getitfixed/icons/parks.png'
);
```

4.2.3 Affichage des icônes

Les icônes sont utilisées par :

- les marqueurs de signalement via `Issue.icon_url()` ;

- la sélection de type/catégorie via les données de catégorie ;
- la légende de la carte publique générée par `getitfixed/static/scripts/legend.js`.

Propriétés recommandées pour les icônes :

- format PNG ;
- petites dimensions, par exemple 24×24 ou 32×32 pixels ;
- arrière-plan transparent lorsque possible ;
- visuellement distinctes par catégorie.

4.3 Catégories et types

Les catégories et les types peuvent être créés directement dans PostgreSQL ou via tout outillage administratif personnalisé que vous ajoutez autour des modèles.

Exemple de catégorie :

```
INSERT INTO getitfixed.category (label_fr, label_en, email, icon)
VALUES (
  'Déchets',
  'Waste',
  'waste@example.org',
  'static://getitfixed:static/icons/gif-green.png'
);
```

Exemple de type :

```
INSERT INTO getitfixed.type (label_fr, label_en, category_id, wms_layer)
VALUES (
  'Dépôt sauvage',
  'Illegal dumping',
  1,
  NULL
);
```

Si un type possède un `wms_layer`, le JavaScript du formulaire public peut ajouter cette couche WMS à la carte lorsque l'utilisateur sélectionne le type.

4.4 Textes et traductions

L'application enregistre les traductions depuis :

```
getitfixed:locale
```

Les langues disponibles par défaut sont configurées dans le fichier `.ini` :

```
pyramid.default_locale_name = en
pyramid.available_languages = en fr de
```

Le négociateur de locale utilise d'abord la négociation de locale par défaut de Pyramid, puis se rabat sur la correspondance `Accept-Language` avec les langues configurées.

Cibles Make utiles :

```
make update-catalog
make compile-catalog
```

`update-catalog` extrait et fusionne les chaînes de traduction. `compile-catalog` compile les fichiers `.po` en fichiers `.mo`.

4.5 Page 404

L'application enregistre une vue 404 par défaut rendue avec :

```
getitfixed:templates/404.jinja2
```

Pour la personnaliser, fournissez une mise en page personnalisée ou enregistrez votre propre vue Pyramid not-found dans une application d'intégration.

5 Flux de processus

5.1 Flux de haut niveau

```
Reporter opens public map
        |
        v
Reporter creates a new issue
        |
        v
Issue is stored with status = new
        |
        +--> Email confirmation to reporter
        |
        +--> Email notification to category administrator
        |
        v
Administrator reviews issue
        |
        +--> Keeps private or makes public
        |
        +--> Changes status and/or comments
        |
        v
Reporter follows private link and may comment
        |
        v
Administrator continues processing
        |
        v
Issue is resolved and disappears from public open lists/map
```

5.2 Soumission publique d’un signalement

1. Le déclarant ouvre `/getitfixed/issues`.
2. La vue carte affiche les signalements existants qui sont à la fois :
    - non `new` ;
    - non `private`.
3. Le déclarant crée un nouveau signalement.
4. Le formulaire capture :
    - catégorie ;
    - type ;
    - position sur la carte ;
    - description ;
    - texte de localisation ;
    - photos ;
    - prénom ;
    - nom ;
    - téléphone ;
    - e-mail.
5. Le signalement est enregistré avec le statut par défaut `new`.
6. Deux e-mails sont envoyés :
    - `new_issue_email` au déclarant ;
    - `admin_new_issue_email` à l’e-mail de catégorie.
7. Le déclarant est redirigé vers la page privée du signalement avec un message de succès.

5.3 Affichage de la carte publique

La requête publique exclut :

```
Issue.status.notin_([STATUS_NEW])
Issue.private.is_(False)
```

Par conséquent :

- les nouveaux signalements sont masqués jusqu’à leur modération ;
- les signalements privés n’apparaissent jamais sur la carte publique ;
- les signalements résolus ne sont pas explicitement exclus par la requête publique dans la révision examinée, mais le filtrage de liste ouverte admin traite `resolved` comme clôturé.

5.4 Flux de sélection catégorie/type

Sur le formulaire public de signalement :

1. Le navigateur récupère les données catégorie/type depuis la route `categories.json`.
2. Les catégories sont chargées dans le champ de sélection de catégorie.
3. La sélection d’une catégorie filtre les types disponibles.
4. La sélection d’un type peut ajouter une couche WMS configurée à la carte.
5. Sur mobile, le choix de la carte/catégorie/type est présenté comme une première étape focalisée avant

de compléter le reste du formulaire.

La route JSON renvoie des objets de la forme :

```
[
  {
    "id": 1,
    "label": "Roads",
    "icon": "...",
    "types": [
      {
        "id": 10,
        "label": "Pothole",
        "wms_layer": null
      }
    ]
  }
]
```

5.5 Flux administrateur

1. L'administrateur ouvre `/getitfixed_admin/issues`.
2. La liste admin affiche par défaut les signalements ouverts, c'est-à-dire ceux dont le statut n'est pas `resolved`.
3. La liste peut être filtrée par statut et par catégorie.
4. L'administrateur ouvre un signalement.
5. Il peut :
   - inspecter les détails du signalement ;
   - inspecter les photos ;
   - ajouter un événement ;
   - mettre à jour le statut ;
   - ajouter des commentaires publics ou privés ;
   - marquer le signalement comme privé ou public.
6. Les événements admin publics peuvent notifier le déclarant via `update_issue_email`.
7. Les commentaires admin privés restent visibles uniquement par les administrateurs.

5.6 Flux privé de suivi du déclarant

1. Le déclarant reçoit un e-mail contenant un lien privé vers le signalement.
2. Le lien pointe vers `/getitfixed_private/issues/<hash>`.
3. Le formulaire de signalement est rendu en lecture seule.
4. Les événements publics sont affichés.
5. Le déclarant peut ajouter un commentaire.
6. Les commentaires du déclarant créent un événement dont l'auteur est `customer`.
7. Une notification est envoyée à l'e-mail de catégorie avec `update_issue_email`.

5.7 Visibilité des événements

- Interface admin : affiche tous les événements du signalement.
- Interface privée du déclarant : affiche uniquement les événements publics.
- Les commentaires du déclarant ont pour auteur `customer`.
- Les commentaires de l'administrateur ont pour auteur `admin`.
- Les événements admin privés ne devraient pas notifier le déclarant.

5.8 Flux de confidentialité

Les administrateurs peuvent basculer `Issue.private`.

Lorsque `private = true` :

- le signalement est masqué de la carte publique ;
- le signalement est inaccessible via l'URL numérique publique ;
- il reste accessible dans l'interface admin ;
- il reste accessible via l'URL privée avec hash pour le déclarant.

5.9 Matrice de notification par e-mail

Déclencheur	Template d'expéditeur	Destinataire	Cible du lien
Nouveau signalement soumis	`new_issue_email`	E-mail du déclarant	URL privée du signalement pour le déclarant.
Nouveau signalement	`admin_new_issue_email`	E-mail de catégorie	URL admin du

Nouveau signalement soumis	`admin_new_issue_email`	E-mail de catégorie	URL admin du signalement.
Le déclarant ajoute un commentaire	`update_issue_email`	E-mail de catégorie	URL admin du signalement ancrée sur les événements.
L'admin ajoute un événement non privé	`update_issue_email`	E-mail du déclarant	URL privée du signalement pour le déclarant ancrée sur les événements.
Signalement résolu	`resolved_issue_email`	E-mail du déclarant	Prévu pour la notification de résolution ; vérifier la note d'implémentation ci-dessus.

6 Exploitation et maintenance

6.1 Journaux

Pendant le développement, `make meacoffee` se termine en suivant les journaux de l'application :

```
docker compose logs -f getitfixed
```

Pour tous les services :

```
docker compose logs -f
```

6.2 Accès à la base de données

```
make psql
```

Commande équivalente :

```
docker compose exec -u postgres db psql getitfixed
```

6.3 Shell Pyramid

```
make pshell
```

Le shell fournit une session SQLAlchemy liée à la requête sous le nom `s`.

6.4 Exécuter les tests

```
make test
```

Cela démarre `db_tests` et exécute les tests d'acceptation dans le service `test`.

6.5 Vérifications de code

```
make check
```

Cela exécute Black en mode vérification et Flake8.

Formater le code :

```
make black
```

7 Intégration avec GeoMapFish

GetItFixed peut être intégré dans un projet GeoMapFish.

Étapes d'intégration typiques :

1. Ajouter la configuration GetItFixed au `vars.yaml` GeoMapFish.
2. Créer/mettre à jour le schéma de base de données `getitfixed` avec Alembic.

3. Accorder la permission `getitfixed_admin` au rôle GeoMapFish approprié.
4. Exposer les routes publiques et admin.

Exemple de commande de migration dans un conteneur GeoMapFish :

```
docker-compose exec geoportal alembic -n getitfixed upgrade head
```

Exemple d'entrée ACL :

```
class Root:
    __acl__ = [
        (Allow, "role_admin", ALL_PERMISSIONS),
        (Allow, "role_getitfixed", "getitfixed_admin"),
    ]
```

Routes attendues :

- Public : `<your_geomapfish_root_url>/getitfixed`
- Admin : `<your_geomapfish_root_url>/getitfixed_admin`

8 Annexe : exemple rapide de configuration

```
vars:
  project: getitfixed

  smtp:
    host: smtp.example.org
    starttls: true
    user: getitfixed@example.org
    password: change-me

  getitfixed:
    default_icon: "static://getitfixed:static/icons/cat-default.png"

    layout: mypackage:templates/getitfixed/layout.jinja2

    map:
      srid: 3857
      baseLayers:
        - type_: "OSM"
      view:
        projection: "EPSG:3857"
        zoom: 12
      fitMaxZoom: 16

    admin_new_issue_email:
      email_from: noreply@example.org
      email_subject: A new issue has been created
      email_body: |
        A new issue has been submitted.
        Review it here: {issue-link}

    new_issue_email:
      email_from: noreply@example.org
      email_subject: Issue declaration confirmation
      email_body: |
        Hello {username},

        We confirm that your issue has been registered.
        You can follow it here: {issue-link}

    update_issue_email:
      email_from: noreply@example.org
      email_subject: Status update for an issue
      email_body: |
        Hello {username},

        The issue {issue.hash} is now {issue.status_en}.
        {event.comment}

        Follow it here: {issue-link}

    resolved_issue_email:
      email_from: noreply@example.org
      email_subject: Issue resolved
      email_body: |
        Hello {username},

        The issue {issue.hash} has now been resolved.

no_interpreted:
  - getitfixed.admin_new_issue_email.email_body
  - getitfixed.new_issue_email.email_body
```

```
  - getitfixed.update_issue_email.email_body
  - getitfixed.resolved_issue_email.email_body
```