camptocamp
INNOVATIVE SOLUTIONS
BY OPEN SOURCE EXPERTS

GetItFixed Technical Documentation

Installation, deployment-oriented configuration and technical reference

Generated documentation · 2026-05-07

1 Overview

GetItFixed is a Pyramid-based web application for reporting, moderating and resolving geolocated issues such as potholes, broken streetlights, illegal dumping or other public-service incidents.

Citizens create issues on a map, optionally attach photos, and receive a private follow-up link by email. Administrators review incoming reports, make them public or private, update their status, comment on them, and close them when resolved.

This documentation was written from the `camptocamp/getitfixed` repository, commit `a07f690` dated 2026-04-30.

1.1 Main capabilities

- Public map showing validated, non-private issues.
- Public issue submission form with category, type, geometry, description, location text, photos and reporter contact fields.
- Private reporter view accessible through a UUID-like issue hash.
- Administration interface for issue moderation and follow-up.
- Event timeline attached to each issue.
- Email notifications for new issues and updates.
- Category-based routing of administrative emails.
- Custom category icons and default icon support.
- Configurable map settings, projections and base layers.
- PostgreSQL/PostGIS storage.
- Redis-backed sessions.
- Docker Compose development stack.
- English, French and German localisation support.

1.2 Interfaces and routes

GetItFixed exposes three main interfaces:

Interface	Default route	Purpose
Public	`/getitfixed/issues`	Browse public issues and submit a new issue.
Private reporter	`/getitfixed_private/issues/<hash>`	Reporter follow-up page reached from email links.
Admin	`/getitfixed_admin/issues`	Moderate, update and resolve issues.

There is also a legacy private redirect route:

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

It redirects to the current private reporter URL.

1.3 Technology stack

- Python 3.12+
- Pyramid
- SQLAlchemy
- GeoAlchemy2
- Alembic
- PostgreSQL with PostGIS
- Redis sessions through `beaker_redis`
- c2cgeoform for CRUD forms, maps and GeoJSON endpoints
- Colander / Deform for schema-driven forms
- Jinja2 templates
- Bootstrap 3, bootstrap-table, jQuery
- OpenLayers through c2cgeoform
- Lingua/Babel translation catalogues
- Waitress in production
- Docker Compose for local development and demo services

1.4 Application architecture

The core Python package is `getitfixed`.

Important modules:

Path	Role
`getitfixed/__init__.py`	Pyramid application factory, configuration loading and package inclusion.
`getitfixed/routes.py`	Public, private and admin route registration.
`getitfixed/models/getitfixed.py`	SQLAlchemy models and form metadata.
`getitfixed/views/public/`	Public map and issue submission views.
`getitfixed/views/private/`	Reporter follow-up views and reporter comments.

`getitfixed/views/admin/`	Admin list, edit, event and photo views.
`getitfixed/emails/email_service.py`	SMTP email sending.
`getitfixed/static/`	CSS, JavaScript, icons and browser assets.
`getitfixed/templates/`	Jinja2 templates.
`getitfixed/alembic/`	Database migrations.
`vars.yaml`	Main project customisation source.
`development.ini` / `production.ini`	Pyramid/PasteDeploy application settings.

1.5 Data model

GetItFixed stores data in the `getitfixed` PostgreSQL schema by default.

1.5.1 Category

A category is a high-level issue family.

Fields:

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

The category email is important: it is used as the destination for administrative notifications when a new issue is submitted in that category, and when a reporter adds a follow-up comment.

The category icon is used on the map legend and issue markers. If empty, the configured default icon is used.

1.5.2 Type

A type is a sub-category. Each issue is linked to one type.

Fields:

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

`wms_layer` is optional. When present, the front-end can add a WMS overlay associated with the selected issue type.

1.5.3 Issue

An issue is the central business object.

Fields:

- `id`
- `hash`
- `request_date`
- `type_id`
- `status`
- `description`
- `localisation`
- `geometry`, stored as a `POINT` in EPSG:4326
- `firstname`
- `lastname`
- `phone`
- `email`
- `private`
- related `photos`
- related `events`

The public interface identifies existing public issues by numeric `id`. Private and admin follow-up links use `hash`, which is generated with UUID-like values and is safer to send by email.

1.5.4 Photo

Photos are stored in the database.

Fields:

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

Admin and private photo download views use the photo hash as identifier.

1.5.5 Event

Events form the issue timeline.

Fields:

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

Events are used to update issue status and to exchange comments between reporter and administrator.

1.6 Statuses

The status workflow is:

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

Available internal values:

Status	Meaning
`new`	The issue was submitted but is not public yet.
`validated`	The administrator accepted the issue.
`in_progress`	The issue is being processed.
`waiting_for_reporter`	More information is required from the reporter.
`resolved`	The issue is closed.

The public map hides issues whose status is `new` and issues marked `private`.

2 Installation

2.1 Requirements

For a local Docker-based installation:

- Git
- Docker
- Docker Compose
- GNU Make

For direct development outside Docker, the package targets Python 3.12 or newer. Docker remains the primary development path in the repository.

2.2 Clone the repository

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

2.3 Start with demo data

The quickest path is:

```
make meacoffee
```

This target builds the Docker images, starts the services, runs database migrations, loads demo data and follows the application logs.

Open:

- Public interface: http://localhost:8080/getitfixed/issues
- Admin interface: http://localhost:8080/getitfixed_admin/issues
- Demo webmail: http://localhost:8082/webmail/?_task=mail&_mbox=INBOX

2.4 Start with an empty database

The upstream documentation mentions `make meadeca` for an empty database. In the reviewed repository revision, the visible Makefile target is `meacoffee`; if you need an empty database, run the stack without loading test data, or adapt the Makefile/init sequence accordingly.

A practical manual sequence is:

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

2.5 Docker Compose services

The development stack defines these services:

Service	Purpose
`db`	PostgreSQL/PostGIS database.
`redis`	Redis session storage.
`getitfixed`	Pyramid web application, served by `pserve`.
`smtp`	SMTP sink used for local/demo email capture.
`courier-imap`	IMAP server for demo mail.
`webmail`	Browser-based access to captured demo email.
`db_tests`	Test database.
`test`	Pytest acceptance-test runner.

2.6 Useful Make targets

Target	Purpose
`make help`	Display available targets.
`make meacoffee`	Build, start, initialise demo data and show logs.
`make up`	Build and start containers.
`make build`	Build runtime files and Docker images.
`make initdb`	Run migrations and load test data.
`make test`	Run acceptance tests.
`make check`	Run formatting and lint checks.
`make black`	Format Python code with Black.
`make docs`	Build the original Sphinx documentation.
`make psql`	Open a PostgreSQL shell in the database container.
`make pshell`	Open a Pyramid shell.
`make clean`	Remove generated translation files.
`make cleanall`	Stop/remove containers, generated `.env`, and Docker images.
`make venv`	Create a local Python virtual environment for IDE use.

2.7 Database migrations

Run migrations inside the application container:

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

Create a new revision:

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

2.8 Test data

Load demo categories, types and issues:

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

The test-data script creates categories, types and 100 sample issues when the corresponding tables are empty.

3 Configuration

Configuration is split between PasteDeploy `.ini` files, environment variables and the generated YAML configuration.

3.1 PasteDeploy settings

Main files:

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

The production app section uses:

```
[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
```

The application factory loads `app.cfg`, then merges the generated YAML configuration into Pyramid settings.

3.2 Environment variables

The Makefile and Docker Compose setup rely on these variables:

Variable	Default	Purpose
`DOCKER_BASE`	`camptocamp/getitfixed`	Docker image prefix.
`DOCKER_TAG`	`latest`	Docker image tag.
`DOCKER_PORT`	`8080`	Host port mapped to container port 8080.
`PGHOST`	`db`	PostgreSQL host.
`PGHOST_SLAVE`	`db`	Secondary/read PostgreSQL host, if used.
`PGPORT`	`5432`	PostgreSQL port.
`PGDATABASE`	`getitfixed`	Database name.
`PGUSER`	`getitfixed`	Database user.
`PGPASSWORD`	`getitfixed`	Database password.
`PROXY_PREFIX`	empty	URL prefix used by PasteDeploy proxy-prefix filter.
`DEVELOPMENT`	`TRUE`	Selects development or production ini path in `.env.mako`.

3.3 `vars.yaml` and `config.yaml`

`vars.yaml` is the human-maintained configuration source. `config.yaml` is generated from it.

The Makefile rule is:

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

Generate it with:

```
make config.yaml
```

The important top-level keys are:

```
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 Email settings

Email delivery is handled by `getitfixed/emails/email_service.py`.

The service reads:

- `smtp`, for connection settings;
- `getitfixed.<template_name>`, for sender, subject and body.

If `smtp` is missing or empty, the application logs a warning and does not send email.

3.4.1 SMTP configuration

Example:

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

Fields:

Field	Required	Meaning
`host`	yes	SMTP server hostname.
`ssl`	no	Use `SMTP_SSL` when true.
`starttls`	no	Call `starttls()` after connecting when true.
`user`	no	SMTP username. No authentication is attempted when absent or empty.
`password`	if `user` is set	SMTP password.

The current implementation connects without an explicit port. Python's default SMTP port is therefore used unless the code is extended.

3.4.2 Email templates

Four templates are configured under `vars.getitfixed`.

3.4.2.1 `admin_new_issue_email`

Sent to the category administrator when a reporter submits a new issue.

Destination:

```
issue.category.email
```

Typical variables available in the body:

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

Example:

```
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`

Sent to the reporter after successful issue submission.

Destination:

```
issue.email
```

Example:

```
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`

Sent when a public follow-up comment or status update should notify the other party.

Typical destinations:

- reporter email, when an administrator posts a non-private event;
- category email, when a reporter posts a comment from the private interface.

Example:

```
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`

Configured for resolved notifications to the reporter.

Example:

```
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.
```

Implementation note: in the reviewed revision, the admin event view assigns `issue.status = event_status`

before checking whether the status changed to `resolved`. That makes the specific resolved-email branch unreachable as written. The generic update notification path may still apply for non-private admin events. If resolved emails are business-critical, verify and adjust `getitfixed/views/admin/events.py`.

3.4.3 Template formatting

Email bodies are Python format strings. Values are interpolated with:

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

Use single braces for variables, for example `{issue-link}` or `{issue.status_en}`.

Because some keys contain hyphens, such as `issue-link`, the implementation passes them through `template_kwargs`; keep the existing pattern when adding new templates.

3.4.4 Translation extraction

Email subjects and bodies are extracted for translation by `getitfixed/lingua_extractor.py`.

The extracted paths are:

- `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`

The default `vars.yaml` marks bodies as `no_interpreted`, preserving literal formatting in the templating step.

3.5 Map configuration

Map configuration lives under `vars.getitfixed.map`.

The default model code merges this configuration with c2cgeoform defaults and forces mobile support:

```
_map_config = {
    **default_map_settings,
    **{"mobile": True},
    **_getitfixed_config.get("map", {}),
}
```

Common keys:

Key	Purpose
`srid`	SRID used by the map widget serialization/deserialization.
`projections`	Projection definitions registered in the browser.
`baseLayers`	OpenLayers/c2cgeoform layer definitions.
`view`	Projection, extent, center or zoom parameters.
`fitMaxZoom`	Maximum zoom when fitting on an existing issue geometry.

Example for EPSG:2056 WMTS:

```
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
```

When writing URLs inside `vars.yaml`, escape braces as required by the `c2c-template` processing used in this project. The repository examples use doubled or quadrupled braces where necessary.

4 Customization

4.1 Layout customisation

The default layout is:

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

Override it under `vars.getitfixed.layout`:

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

A custom layout can extend the default one and override Jinja blocks:

```
{% 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 %}
```

Available default blocks include:

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

4.2 Icons

Icons are configured at two levels:

1. The global default icon.
2. The icon attached to each category.

4.2.1 Default icon

Set the default icon in `vars.yaml`:

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

When a category has no icon, this default is used.

4.2.2 Category icons

The `category.icon` database field stores the icon definition.

The application supports two URL forms:

Form	Example	Behaviour
Pyramid static URL	`static://getitfixed:static/icons/gif-green.png`	Converted with `request.static_url`.
Absolute URL	`https://example.org/icons/tree.png`	Used as-is.

The static form is parsed by `getitfixed/url.py`:

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

Examples:

```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 Icon display

Icons are used by:

- issue markers through `Issue.icon_url()`;
- type/category selection through category data;
- the public map legend generated by `getitfixed/static/scripts/legend.js`.

Recommended icon properties:

- PNG format;
- small dimensions, for example 24×24 or 32×32 pixels;
- transparent background when possible;
- visually distinct per category.

4.3 Categories and types

Categories and types can be created directly in PostgreSQL or through any custom administrative tooling you add around the models.

Example category:

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

Example type:

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

If a type has a `wms_layer`, the public form JavaScript can add that WMS layer to the map when the user selects the type.

4.4 Texts and translations

The application registers translations from:

```
getitfixed:locale
```

Default available languages are configured in the `.ini` file:

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

The locale negotiator uses Pyramid's default locale negotiation first, then falls back to `Accept-Language` matching against the configured languages.

Useful Make targets:

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

`update-catalog` extracts and merges translation strings. `compile-catalog` compiles `.po` files to `.mo` files.

4.5 404 page

The application registers a default 404 view rendered with:

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

To customise it, provide a custom layout or register your own Pyramid not-found view in an integrating application.

5 Process flow

5.1 High-level flow

```
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 Public issue submission

1. The reporter opens `/getitfixed/issues`.
2. The map view displays existing issues that are both:
    - not `new`;
    - not `private`.
3. The reporter creates a new issue.
4. The form captures:
    - category;
    - type;
    - map position;
    - description;
    - localisation text;
    - photos;

    - firstname;
    - lastname;
    - phone;
    - email.
5. The issue is saved with default status `new`.
6. Two emails are sent:
    - `new_issue_email` to the reporter;
    - `admin_new_issue_email` to the category email.
7. The reporter is redirected to the private issue page with a success message.

5.3 Public map display

The public query filters out:

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

Therefore:

- new issues are hidden until moderated;
- private issues never appear on the public map;
- resolved issues are not explicitly filtered out by the public query in the reviewed revision, but admin open-list filtering treats `resolved` as closed.

5.4 Category/type selection flow

On the public issue form:

1. The browser fetches category/type data from the `categories.json` route.
2. Categories are loaded in the category select field.
3. Selecting a category filters available types.
4. Selecting a type can add a configured WMS layer to the map.
5. On mobile, the map/category/type choice is presented as a focused first step before completing the rest of the form.

The JSON route returns objects shaped like:

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

5.5 Administrator flow

1. The administrator opens `/getitfixed_admin/issues`.
2. The admin list defaults to open issues, meaning issues whose status is not `resolved`.
3. The list can be filtered by status and category.
4. The administrator opens an issue.
5. They can:
    - inspect issue details;
    - inspect photos;
    - add an event;
    - update status;
    - add public or private comments;
    - mark the issue private or public.
6. Public admin events can notify the reporter through `update_issue_email`.
7. Private admin comments stay visible to administrators only.

5.6 Reporter private follow-up flow

1. The reporter receives an email containing a private issue link.
2. The link points to `/getitfixed_private/issues/<hash>`.
3. The issue form is rendered read-only.

4. Public events are displayed.
5. The reporter can add a comment.
6. Reporter comments create an event authored by `customer`.
7. A notification is sent to the category email using `update_issue_email`.

5.7 Event visibility

- Admin interface: shows all issue events.
- Private reporter interface: shows public events only.
- Reporter comments are authored as `customer`.
- Administrator comments are authored as `admin`.
- Private admin events should not notify the reporter.

5.8 Privacy flow

Administrators can toggle `Issue.private`.

When `private = true`:

- the issue is hidden from the public map;
- the issue is inaccessible through the public numeric URL;
- it remains accessible in the admin interface;
- it remains accessible through the private hash URL for the reporter.

5.9 Email notification matrix

Trigger	Sender template	Recipient	Link target
New issue submitted	`new_issue_email`	Reporter email	Private reporter issue URL.
New issue submitted	`admin_new_issue_email`	Category email	Admin issue URL.
Reporter adds comment	`update_issue_email`	Category email	Admin issue URL anchored to events.
Admin adds non-private event	`update_issue_email`	Reporter email	Private reporter issue URL anchored to events.
Issue resolved	`resolved_issue_email`	Reporter email	Intended for resolved notification; verify implementation note above.

6 Operations and maintenance

6.1 Logs

During development, `make meacoffee` ends by following application logs:

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

For all services:

```
docker compose logs -f
```

6.2 Database access

```
make psql
```

Equivalent command:

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

6.3 Pyramid shell

```
make pshell
```

The shell provides a request-bound SQLAlchemy session as `s`.

6.4 Running tests

```
make test
```

This starts `db_tests` and runs the acceptance tests in the `test` service.

6.5 Code checks

```
make check
```

This runs Black in check mode and Flake8.

Format code:

```
make black
```

7 Integration with GeoMapFish

GetItFixed can be integrated in a GeoMapFish project.

Typical integration steps:

1. Add GetItFixed configuration to the GeoMapFish `vars.yaml`.
2. Create/update the `getitfixed` database schema with Alembic.
3. Grant the `getitfixed_admin` permission to the appropriate GeoMapFish role.
4. Expose the public and admin routes.

Migration command example inside a GeoMapFish container:

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

Example ACL entry:

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

Expected routes:

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

8 Appendix: quick configuration example

```
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
```