



# Implementing and managing feature flags

How to effectively manage your feature flags in a monolithic Django application

# GitGuardian



# GitGuardian

## In numbers

### Inception

Founded in **2017**  
Automated secrets detection, in public and private repositories

1

### Growth

**62** employees  
**37** R&D  
**\$44M** raised in December 2021

3

2

### Tech

~**1B** public commits scanned/year  
~**2.5M** secrets detected in 2021  
**#1 GitHub app** for security  
~**1.3M** repositories scanned

4

### Some of our clients



# GitGuardian

## Products

### Public Monitoring

Auto-discover who your developers are on GitHub  
Monitor under-the-radar activity  
Be alerted in real-time

The Perimeter interface displays a list of developers with columns for Filter, Email Domain, Why in Perimeter, State, Name, Email(s), and Why in Perimeter. The list includes developers like Anthony Dong, Arthur Vervae, AymericGG, Bastien Arnout, CharlesGG, defunk, Clément Martinez, and David Héralut.

Filter	EMAIL DOMAIN	WHY IN PERIMETER	STATE	NAME	EMAIL(S)	WHY IN PERIMETER
	All	All	All			
				Anthony Dong	anthony.dong@gitguardian.com	Monitored email domain
				Arthur Vervae	arthur.vervae@gitguardian.com	Monitored email domain
				AymericGG	aymeric.sicard@gitguardian.com	Monitored email domain Member of organization
				Bastien Arnout	bastien.arnout@gitguardian.com	Monitored email domain
				CharlesGG	charles.riveau@gitguardian.com	Monitored email domain
				defunk	chris@github.com chris@ozmm.org	Manual addition
				Clément Martinez		Monitored email domain
				dherault	david.herault@gitguardian.com david.herault@pix.fr	Monitored email domain

The Incidents interface shows a table of secrets with columns for Filter, Date, Secret, Info, and Status. The table lists several Slack Bot Tokens (#2356) with 5 occurrences each. The status for these tokens is 'Regression' and 'Assigned'.

Filter	DATE	SECRET	INFO	STATUS
	Month 00, 2020	Slack Bot Token #2356	prm-dev-team/check-runs-test-repos... myfile_path/fileName.ext Name Surname name.surname@company.com	Regression Assigned
	Month 00, 2020	Slack Bot Token #2356	prm-dev-team/check-runs-test-repos... myfile_path/fileName.ext Name Surname name.surname@company.com	Regression Resolved Secret resolved
	Month 00, 2020	Slack Bot Token #2356	prm-dev-team/check-runs-test-repos... myfile_path/fileName.ext Name Surname name.surname@company.com	Regression Triggered Secret not resolved
	Month 00, 2020	Slack Bot Token #2356	prm-dev-team/check-runs-test-repos... myfile_path/fileName.ext Name Surname name.surname@company.com	Regression Assigned
	Month 00, 2020	Slack Bot Token #2356	prm-dev-team/check-runs-test-repos... myfile_path/fileName.ext Name Surname name.surname@company.com	Regression Assigned

### Internal Monitoring

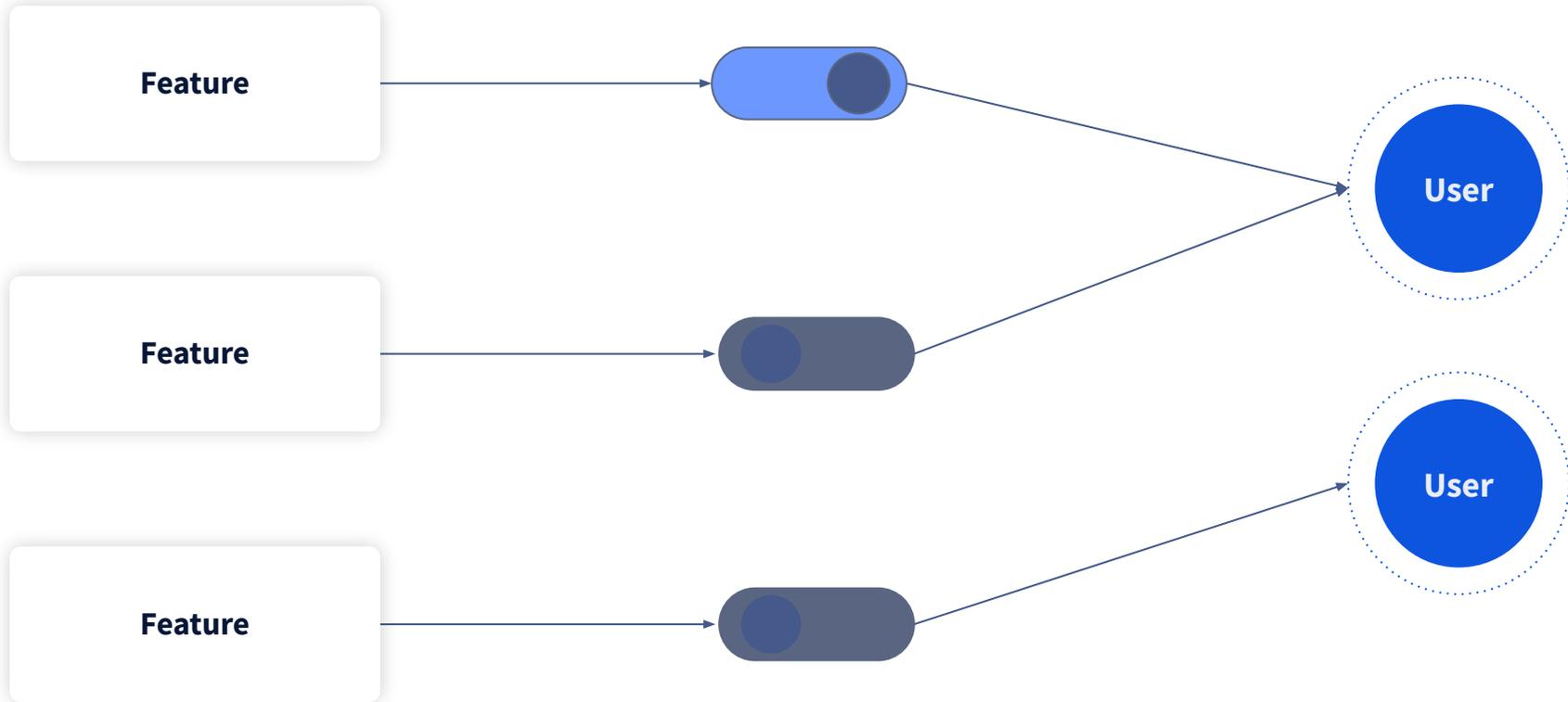
Monitor your repositories  
Detect secret leaks  
Be alerted in real-time  
Integrate remediation to your workflow

# Basics



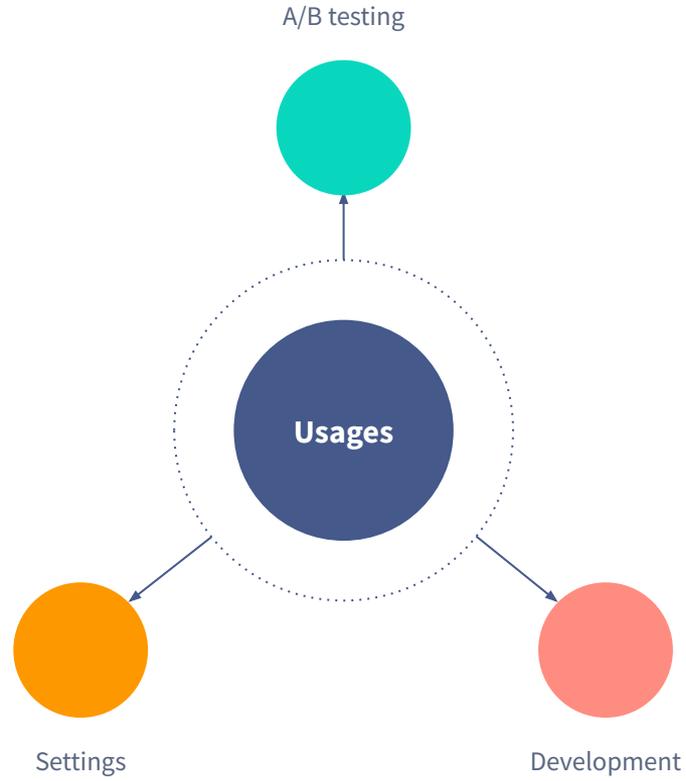
# Basics

## Feature Flag definition



# Basics

## Usages



# First implementations



# First Implementations

Django Solo

## Code

```
# models.py

from django.db import models
from solo.models import SingletonModel

class SiteConfiguration(SingletonModel):
    site_name = models.CharField(max_length=255, default='Site Name')
    maintenance_mode = models.BooleanField(default=False)

    def __str__(self):
        return "Site Configuration"

    class Meta:
        verbose_name = "Site Configuration"
```



### On the fly

Settings|preferences can be updated with the admin or an endpoint.



### User

Possible but all the logic would need to be coded.



### Database

Migrations are needed for every new feature flag.



# First Implementations

## Poor man example

### Code

```
from django.db import models

class PoorManFeatureFlag(models.Model):
    settings = models.JSONField()
```



### On the fly

Settings|preferences can be updated with the admin or an endpoint.



### User

Possible but all the logic would need to be coded.



### Database

No migrations needed.



# First Implementations

## Django Dynamic Preferences

### Code

```
# we create some section objects to link related preferences together

general = Section('general')
discussion = Section('discussion')

# We start with a global preference
@global_preferences_registry.register
class SiteTitle(StringPreference):
    section = general
    name = 'title'
    default = 'My site'
    required = False

@global_preferences_registry.register
class MaintenanceMode(BooleanPreference):
    name = 'maintenance_mode'
    default = False

# now we declare a per-user preference
@user_preferences_registry.register
class CommentNotificationsEnabled(BooleanPreference):
    """Do you want to be notified on comment publication?"""
    section = discussion
    name = 'comment_notifications_enabled'
    default = True
```



### On the fly

Settings|preferences can be updated with the admin or an endpoint.



### User

Can be linked to the users.



### Database

No migration needed.



**Be Production Ready!**



# Be production Ready!

## Account Preferences

### Code

```
class AccountPreferenceModel(PerInstancePreferenceModel):  
    instance = models.ForeignKey(  
        Account, on_delete=models.CASCADE, verbose_name="  
    )  
  
    class Meta:  
        app_label = "app_configuration"  
        verbose_name = "Account preference"
```

```
class AccountPreferenceRegistry(PerInstancePreferenceRegistry)
```

```
account_preferences_registry = AccountPreferenceRegistry()
```

```
@account_preferences_registry.register  
class AllowLegacySSOUrl(AbstractAccountPreference, NullableBooleanPreference):
```

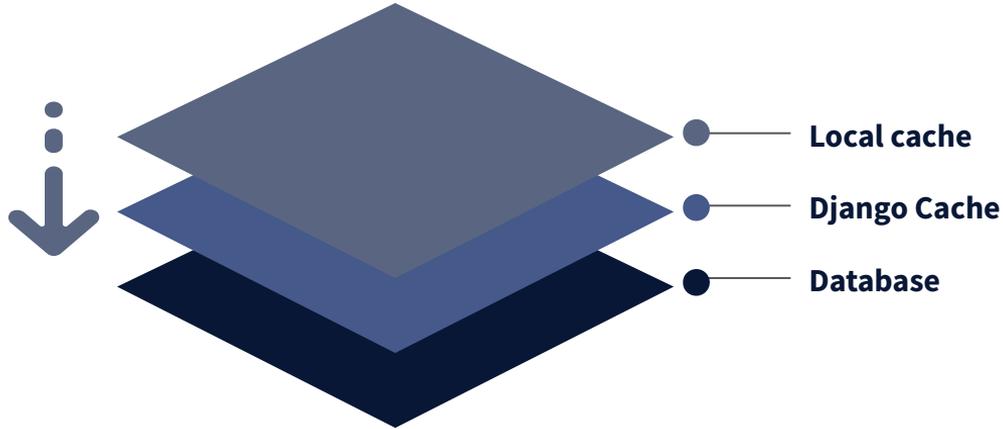
### PerInstancePreferenceModel

- Set your preferences per User/Account
- Finer grained preferences



# Be production Ready!

Improvements - Requests optimization



## Faster

Local Cache is 10x faster than Redis.

## Requests

Less network requests.  
Less database requests.

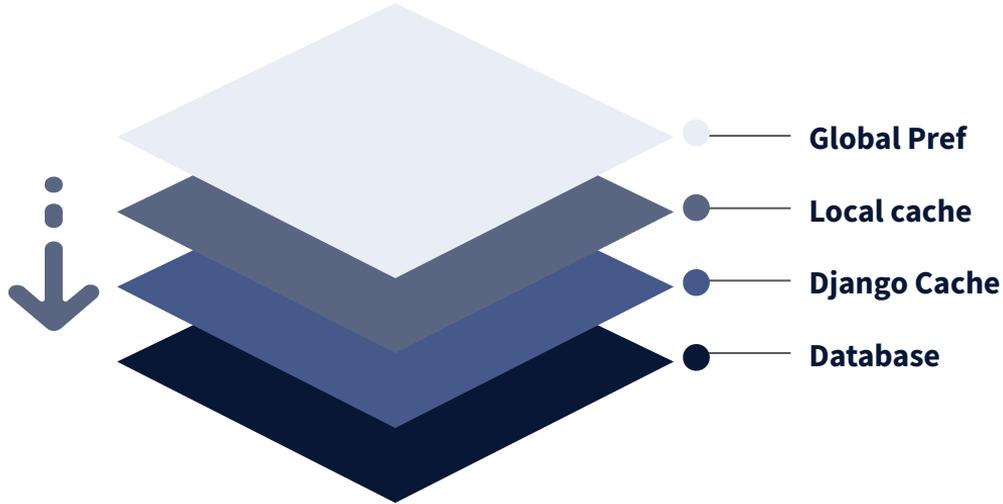
## Rewrite

Adapted most of the library methods.



# Be production Ready!

## Improvements - Hierarchy



### Dependency

Create a hierarchy with `depends_on`.

### Inheritance - Custom property

`has_global_preference_fallback`.

### Rewrite

Adapted most of the library methods.



# Be production Ready!

## Improvements

### Cache

- Faster than DB
- Local Cache

### Database Requests

- Adapt manager to decrease requests

### Models

- Adapt models for fallback
- Adapt managers for optimization
- Nullable Preferences

### Hierarchy

- Dependency between preferences
- Fallback on the global when necessary

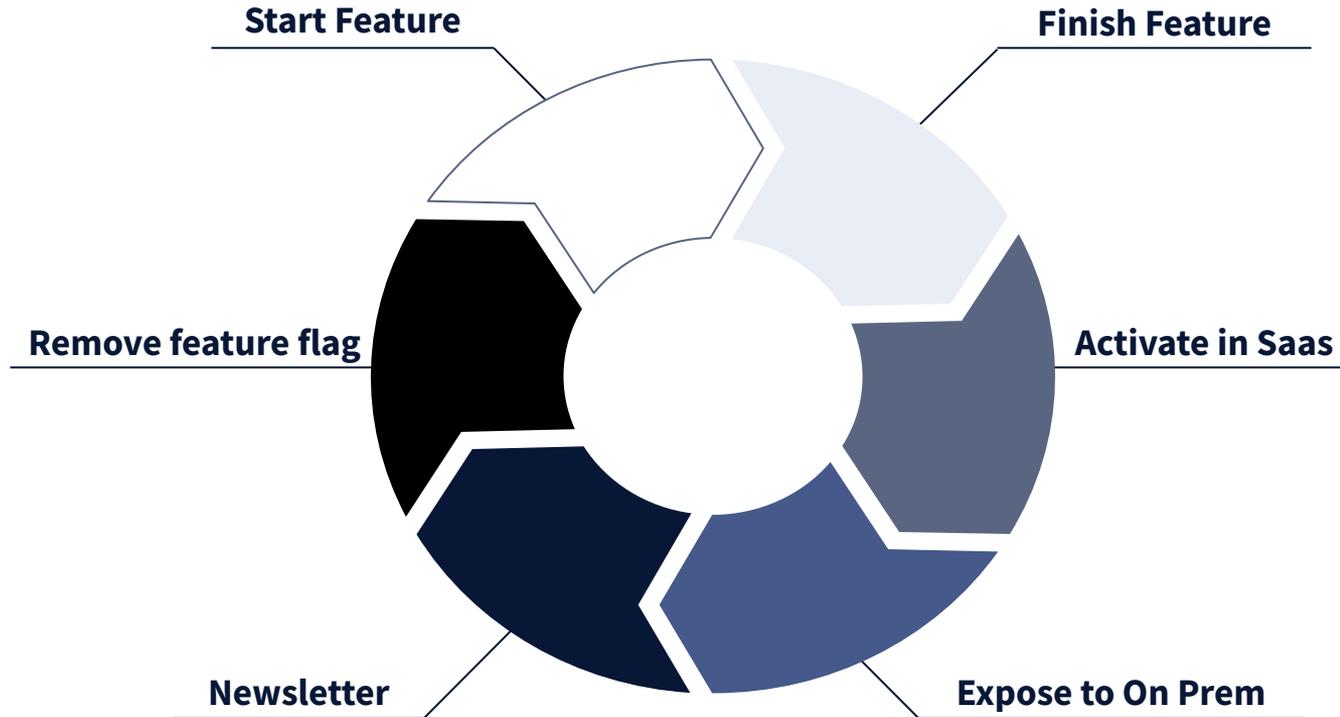
### Creation

- Create Preference only when necessary
- No overflow of the database



# Be production Ready!

Lifecycle





**Contact us - We are hiring!**

[careers@gitguardian.com](mailto:careers@gitguardian.com)

[www.gitguardian.com](http://www.gitguardian.com)



## Links

<https://github.com/lazybird/django-solo/>

<https://django-dynamic-preferences.readthedocs.io/en/latest/>