django-e2ee-framework

Philipp S. Sommer

Oct 22, 2022

CONTENTS:

An end-to-end encryption framework for Django

Warning: This package is work in progress, especially it's documentation. Stay tuned for updates and discuss with us at https://gitlab.hzdr.de/hcdc/django/django-e2ee-framework

CHAPTER

ONE

INSTALLATION

To install the *django-e2ee-framework* package for your Django project, you need to follow two steps:

1. *Install the package*
2. *Add the app to your Django project*

1.1 Installation from PyPi

The recommended way to install this package is via pip and PyPi via:

```
pip install django-e2ee-framework
```

Or install it directly from the source code repository on Gitlab via:

```
pip install git+https://gitlab.hzdr.de/hcdc/django/django-e2ee-framework.git
```

The latter should however only be done if you want to access the development versions.

1.2 Install the Django App for your project

To use the *django-e2ee-framework* package in your Django project, you need to add the app to your *INSTALLED_APPS*, configure your *urls.py*, run the migration, add a login button in your templates. Here are the step-by-step instructions:

1. Add the *django_e2ee* app to your *INSTALLED_APPS*
2. in your projects urlconf (see **:setting:`ROOT_URLCONF`**), add include *`django_e2ee.urls`* via:

   ```
   from django.urls import include, path

   urlpatterns += [
       path("django-e2ee-framework/", include("django_e2ee.urls")),
    ]
   ```

3. Run `python manage.py migrate` to add models to your database
4. Configure the app to your needs (see *Configuration options*).

That's it! For further adaption to you Django project, please head over to the *Configuration options*. You can also have a look into the `testproject` in the source code repository for a possible implementation.

1.3 Installation for development

Please head over to our *contributing guide* for installation instruction for development.

CHAPTER

TWO

CONFIGURATION OPTIONS

2.1 Configuration settings

The following settings have an effect on the app.

CHAPTER

THREE

API REFERENCE

3.1 App settings

This module defines the settings options for the `django-e2ee-framework` app.

3.2 URL config

URL patterns of the django-e2ee-framework to be included via:

```
from django.urls import include, path

urlpatters = [
    path(
        "django-e2ee-framework",
        include("django_e2ee.urls"),
    ),
]
```

Data:

app_name	App name for the django-e2ee-framework to be used in calls to `django.urls.reverse()`

`django_e2ee.urls.`**`app_name = 'e2ee'`**

App name for the django-e2ee-framework to be used in calls to `django.urls.reverse()`

3.3 Models

Models for the django-e2ee-framework app.

Models:

EncryptionKey(*args, **kwargs)	A key that is used for encrypting content in the database.
EncryptionKeySecret(*args, **kwargs)	The secret of an encryption key, encrypted with a users master key.
MasterKey(*args, **kwargs)	A public key for a user that is used for encryption.
MasterKeySecret(*args, **kwargs)	The encrypted private key for a *MasterKey*
SessionKey(*args, **kwargs)	An encrypted private key for a *MasterKey* for one session.

Functions:

delete_session_key(request, **kwargs)	Deactivate the notification subscriptions for the session.
validate_public_key(value)	Try importing a public key with cryptography

class django_e2ee.models.**EncryptionKey**(**args*, ***kwargs*)

Bases: Model

A key that is used for encrypting content in the database.

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Model Fields:

created_by	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
uuid	A wrapper for a deferred-loading field.

Attributes:

created_by_id	
encryptionkeysecret_set	Accessor to the related objects manager on the reverse side of a many-to-one relation.
master_keys	Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.
objects	

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

created_by

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

created_by_id

encryptionkeysecret_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

master_keys

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

objects = <django.db.models.manager.Manager object>

uuid

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class django_e2ee.models.**EncryptionKeySecret**(**args*, ***kwargs*)

Bases: `Model`

The secret of an encryption key, encrypted with a users master key.

Miscellaneous:

DoesNotExist

MultipleObjectsReturned

Model Fields:

encrypted_with	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
encryption_key	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
id	A wrapper for a deferred-loading field.
secret	A wrapper for a deferred-loading field.
signature	A wrapper for a deferred-loading field.
signed_by	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

Attributes:

encrypted_with_id
encryption_key_id
objects
signed_by_id

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

encrypted_with

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

encrypted_with_id

encryption_key

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

encryption_key_id

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

secret

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

signature

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

signed_by

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

signed_by_id

class django_e2ee.models.**MasterKey**(**args*, ***kwargs*)

Bases: `Model`

A public key for a user that is used for encryption.

Miscellaneous:

DoesNotExist

MultipleObjectsReturned

Attributes:

encryptionkey_set	Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.
encryptionkeysecret_set	Accessor to the related objects manager on the reverse side of a many-to-one relation.
masterkeysecret_set	Accessor to the related objects manager on the reverse side of a many-to-one relation.
objects	
pubkey_loaded	The pubkey loaded via cryptography
sessionkey_set	Accessor to the related objects manager on the reverse side of a many-to-one relation.
signed_secrets	Accessor to the related objects manager on the reverse side of a many-to-one relation.
signing_pubkey_loaded	The pubkey loaded via cryptography
user_id	

Model Fields:

pubkey	A wrapper for a deferred-loading field.
signing_pubkey	A wrapper for a deferred-loading field.
user	Accessor to the related object on the forward side of a one-to-one relation.

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

encryptionkey_set

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

encryptionkeysecret_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

masterkeysecret_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

objects = <django.db.models.manager.Manager object>

pubkey

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property pubkey_loaded

The pubkey loaded via cryptography

sessionkey_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

signed_secrets

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

signing_pubkey

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property signing_pubkey_loaded

The pubkey loaded via cryptography

user

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

Restaurant.place is a ForwardOneToOneDescriptor instance.

user_id

class django_e2ee.models.**MasterKeySecret**(**args*, ***kwargs*)

Bases: Model

The encrypted private key for a *MasterKey*

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Model Fields:

identifier	A wrapper for a deferred-loading field.
iv	A wrapper for a deferred-loading field.
master_key	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
salt	A wrapper for a deferred-loading field.
secret	A wrapper for a deferred-loading field.
signing_secret	A wrapper for a deferred-loading field.
uuid	A wrapper for a deferred-loading field.

Attributes:

master_key_id
objects

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

identifier

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

iv

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

master_key

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

master_key_id

objects = <django.db.models.manager.Manager object>

salt

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

secret

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

signing_secret

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

uuid

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class django_e2ee.models.**SessionKey**(**args*, ***kwargs*)

Bases: `Model`

An encrypted private key for a *MasterKey* for one session.

Miscellaneous:

DoesNotExist

MultipleObjectsReturned

Methods:

get_absolute_url()

Model Fields:

ignore	A wrapper for a deferred-loading field.
iv	A wrapper for a deferred-loading field.
master_key	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
secret	A wrapper for a deferred-loading field.
session	Accessor to the related object on the forward side of a one-to-one relation.
session_secret	A wrapper for a deferred-loading field.
signing_secret	A wrapper for a deferred-loading field.

Attributes:

master_key_id
objects
session_id

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

get_absolute_url()

ignore

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

iv

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

master_key

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

master_key_id

objects = <django.db.models.manager.Manager object>

secret

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

session

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

session_id

session_secret

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

signing_secret

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`django_e2ee.models.`**delete_session_key**(*request*, ***kwargs*)

Deactivate the notification subscriptions for the session.

`django_e2ee.models.`**validate_public_key**(*value: str*)

Try importing a public key with cryptography

3.4 Views

Views of the django-e2ee-framework app to be imported via the url config (see *django_e2ee.urls*).

Classes:

E2ELoginViewMixin()	A mixin that gets a form to login or create an E2EE password.
EncryptionKeySecretView(**kwargs)	A view to create and get `e2ee.EncryptionKeySecret`.
EncryptionKeyViewSet(**kwargs)	A viewset for `e2ee.EncryptionKey`
MasterKeyRetrieveView(**kwargs)	Retreive the master key of the user.
MasterKeySecretViewSet(**kwargs)	A viewset for a users `e2ee.MasterKeySecret`
MasterKeyViewSet(**kwargs)	A view to create and retrieve a users `e2ee.MasterKey`
SessionKeyUpdateView(**kwargs)	Retreive the master key of the user.
SessionKeyViewSet(**kwargs)	A viewset for a users `e2ee.SessionKey`

Functions:

dummy_view(request, *args, **kwargs)	

`class django_e2ee.views.`**E2ELoginViewMixin**

Bases: `object`

A mixin that gets a form to login or create an E2EE password.

This mixin provides an additional `e2ee_login_form` context variable for the template. Depending on the fact whether the user already created an E2EE password, this can be a *django_e2ee.forms.PasswordCreateForm* to create a new E2EE password, or a *django_e2ee.forms.PasswordInputForm* to enter the password.

Additionally, it adds a context variable `e2ee_login_url` that points to the URL where the form should be submitted.

Methods:

get_context_data(**kwargs)	

get_context_data(***kwargs*)

class django_e2ee.views.**EncryptionKeySecretView**(***kwargs*)

Bases: `RetrieveAPIView`, `CreateAPIView`

A view to create and get `e2ee.EncryptionKeySecret`.

Methods:

get_object()	Returns the object the view is displaying.
get_queryset()	Get the list of items for this view.
get_serializer(*args, **kwargs)	Return the serializer instance that should be used for validating and deserializing input, and for serializing output.

Attributes:

permission_classes	
queryset	

Classes:

serializer_class	alias of *EncryptionKeySecretSerializer*

get_object()

Returns the object the view is displaying.

You may want to override this if you need to provide non-standard queryset lookups. Eg if objects are referenced using multiple keyword arguments in the url conf.

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

get_serializer(**args*, ***kwargs*)

Return the serializer instance that should be used for validating and deserializing input, and for serializing output.

permission_classes = [<class 'rest_framework.permissions.IsAuthenticated'>, <class 'django_e2ee.permissions.HasEncryptionSecretPermission'>]

queryset

serializer_class

alias of *EncryptionKeySecretSerializer* **Classes:**

Meta()

Methods:

create(validated_data)	We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

class django_e2ee.views.**EncryptionKeyViewSet**(***kwargs*)

Bases: ModelViewSet

A viewset for e2ee.EncryptionKey

Attributes:

basename
description
detail
name
permission_classes
queryset
suffix

Methods:

get_queryset()	Get the list of items for this view.

Classes:

serializer_class	alias of *EncryptionKeySerializer*

basename = None

description = None

detail = None

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

name = None

permission_classes = [<class 'rest_framework.permissions.IsAuthenticated'>]

queryset

serializer_class

alias of *EncryptionKeySerializer* **Classes:**

Meta()

Methods:

create(validated_data)	We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

suffix = None

class django_e2ee.views.**MasterKeyRetrieveView**(***kwargs*)

Bases: RetrieveAPIView

Retreive the master key of the user.

Methods:

get_object()	Returns the object the view is displaying.

Attributes:

permission_classes

Classes:

serializer_class	alias of *MasterKeySerializer*

get_object()

Returns the object the view is displaying.

You may want to override this if you need to provide non-standard queryset lookups. Eg if objects are referenced using multiple keyword arguments in the url conf.

permission_classes = [<class 'rest_framework.permissions.IsAuthenticated'>]

serializer_class

alias of *MasterKeySerializer* **Classes:**

Meta()

Methods:

create(validated_data)	We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

class django_e2ee.views.**MasterKeySecretViewSet**(***kwargs*)

Bases: ModelViewSet

A viewset for a users e2ee.MasterKeySecret

Attributes:

basename
description
detail
name
permission_classes
queryset
suffix

Methods:

get_queryset()	Get the list of items for this view.

Classes:

serializer_class	alias of *MasterKeySecretSerializer*

basename = None

description = None

detail = None

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

name = None

permission_classes = [<class 'rest_framework.permissions.IsAuthenticated'>]

queryset

serializer_class

alias of *MasterKeySecretSerializer* **Classes:**

Meta()

Methods:

create(validated_data)	We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

suffix = None

class django_e2ee.views.**MasterKeyViewSet**(***kwargs*)

Bases: ModelViewSet

A view to create and retrieve a users e2ee.MasterKey

Attributes:

basename
description
detail
name
permission_classes
queryset
suffix

Classes:

serializer_class	alias of *MasterKeySerializer*

basename = None

description = None

detail = None

name = None

permission_classes = [<class 'rest_framework.permissions.IsAuthenticated'>, <class 'rest_framework.permissions.DjangoModelPermissions'>]

queryset

serializer_class

alias of *MasterKeySerializer* **Classes:**

Meta()

Methods:

create(validated_data)	We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

suffix = None

class django_e2ee.views.**SessionKeyUpdateView**(***kwargs*)

Bases: RetrieveAPIView, UpdateAPIView

Retreive the master key of the user.

Methods:

get_object()	Returns the object the view is displaying.

Attributes:

permission_classes

Classes:

serializer_class	alias of *SessionKeySerializer*

get_object()

Returns the object the view is displaying.

You may want to override this if you need to provide non-standard queryset lookups. Eg if objects are referenced using multiple keyword arguments in the url conf.

permission_classes = [<class 'rest_framework.permissions.IsAuthenticated'>]

serializer_class

alias of *SessionKeySerializer* **Classes:**

Meta()

Methods:

create(validated_data)	We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

class django_e2ee.views.**SessionKeyViewSet**(***kwargs*)

Bases: ModelViewSet

A viewset for a users e2ee.SessionKey

Attributes:

basename
description
detail
name
permission_classes
queryset
suffix

Methods:

get_queryset()	Get the list of items for this view.

Classes:

serializer_class	alias of *SessionKeySerializer*

basename = None

description = None

detail = None

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

name = None

permission_classes = [<class 'rest_framework.permissions.IsAuthenticated'>]

queryset

serializer_class

alias of *SessionKeySerializer* **Classes:**

Meta()

Methods:

create(validated_data)	We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

suffix = None

django_e2ee.views.**dummy_view**(*request*, **args*, ***kwargs*)

3.5 django_e2ee package

Django End-to-End Encryption Framework

An end-to-end encryption framework for Django

3.5.1 Subpackages

django_e2ee.templatetags package

Submodules

E2EE template tags

Template tags for the django-e2ee-framework.

Functions:

e2ee_enabled(session)	Test if a session has e2e enabled.
e2ee_ignored(session)	Test if a session has e2e enabled.
get_password_create_form()	
get_session_key(session)	Get the `e2ee.SessionKey` for a session.
get_session_key_form(context, session_key)	Get a form to enable the session verification.
get_session_keys(context)	
has_session_key(session)	Test if a session has e2e enabled.

`django_e2ee.templatetags.e2ee.`**`e2ee_enabled`**(*session: Union[str, SessionStore, Session]*) → bool

Test if a session has e2e enabled.

`django_e2ee.templatetags.e2ee.`**`e2ee_ignored`**(*session: Union[str, SessionStore, Session]*) → bool

Test if a session has e2e enabled.

`django_e2ee.templatetags.e2ee.`**`get_password_create_form`**()

`django_e2ee.templatetags.e2ee.`**`get_session_key`**(*session: Union[str, SessionStore, Session]*) → Optional[*models.SessionKey*]

Get the `e2ee.SessionKey` for a session.

`django_e2ee.templatetags.e2ee.`**`get_session_key_form`**(*context*, *session_key*) → *E2EESessionForm*

Get a form to enable the session verification.

`django_e2ee.templatetags.e2ee.`**`get_session_keys`**(*context*) → QuerySet[*models.SessionKey*]

`django_e2ee.templatetags.e2ee.`**`has_session_key`**(*session: Union[str, SessionStore, Session]*) → bool

Test if a session has e2e enabled.

django_e2ee.tests package

Tests for the *django_e2ee* app.

Submodules

Tests for the django-e2ee-framework models

Functions:

test_model()	Place holder test to make sure pytest finds at least one.

`django_e2ee.tests.test_models.`**`test_model`**()

Place holder test to make sure pytest finds at least one.

Delete this once you really start to write some tests.

Tests for the django-e2ee-framework views

Classes:

TestEncryptionKey()	Unit tests for the EncryptionKey class in JavaScript.
TestEncryptionKeyStore()	Unit tests for the EncryptionKey class in JavaScript.

Functions:

test_encryption_decryption_post(e2e_browser, ...)	Test encrypting a message through the service worker.
test_master_key_creation(e2e_browser, test_user)	Test the creation of the master key.
test_master_key_secret_creation(e2e_browser, ...)	Test the creation of another masterkey secret.
test_remote_password_authentication(...)	Test authentication of a new session via other session and password.
test_remote_session_authentication(...)	Test authentication of a new session via other session.
test_session_authentication(e2e_browser, ...)	Test authentication of a new session via password.

class django_e2ee.tests.test_views.**TestEncryptionKey**

Bases: object

Unit tests for the EncryptionKey class in JavaScript.

Methods:

get_encryption_key_js(driver)	
get_existing_encryption_key_js(driver, uuid)	
test_encrypt_decrypt(e2e_browser)	Test encryption and decryption of a message.
test_key_generation(e2e_browser)	Test generating an encryption key.
test_key_upload(e2e_browser)	
test_key_upload_for_multiple_users(...)	Test the encryption key upload for multiple users.
test_key_upload_for_other_user(...)	

get_encryption_key_js(*driver: Remote*)

get_existing_encryption_key_js(*driver: Remote*, *uuid*)

test_encrypt_decrypt(*e2e_browser: Remote*)

Test encryption and decryption of a message.

test_key_generation(*e2e_browser: Remote*)

Test generating an encryption key.

test_key_upload(*e2e_browser: Remote*)

test_key_upload_for_multiple_users(*test_user_factory: Callable[[], User]*, *e2e_browser_factory: Callable[[User], Remote]*)

Test the encryption key upload for multiple users.

test_key_upload_for_other_user(*test_user_factory: Callable[[], User]*, *e2e_browser_factory: Callable[[User], Remote]*)

class django_e2ee.tests.test_views.**TestEncryptionKeyStore**

Bases: object

Unit tests for the EncryptionKey class in JavaScript.

Methods:

get_encryption_key_store_js(driver)	

test_decryption(e2e_browser, test_user)	Test decrypting messages
test_key_creation(e2e_browser)	Test the createKey method of an EncryptionKeyStore
test_key_retrieval(e2e_browser)	Test the getKey method of an EncryptionKeyStore

get_encryption_key_store_js(*driver: Remote*)

test_decryption(*e2e_browser: Remote*, *test_user: User*)

Test decrypting messages

test_key_creation(*e2e_browser: Remote*)

Test the createKey method of an EncryptionKeyStore

test_key_retrieval(*e2e_browser: Remote*)

Test the getKey method of an EncryptionKeyStore

django_e2ee.tests.test_views.**test_encryption_decryption_post**(*e2e_browser: Remote*, *test_user: User*, *make_request: Callable[[Remote, str, str, Optional[Dict]], Dict]*)

Test encrypting a message through the service worker.

django_e2ee.tests.test_views.**test_master_key_creation**(*e2e_browser: Remote*, *test_user: User*)

Test the creation of the master key.

django_e2ee.tests.test_views.**test_master_key_secret_creation**(*e2e_browser: Remote*, *make_request: Callable[[Remote, str, str, Optional[Dict]], Dict]*, *authenticated_browser_factory: Callable[[User], Remote]*, *test_user: User*, *dummy_password: str*)

Test the creation of another masterkey secret.

django_e2ee.tests.test_views.**test_remote_password_authentication**(*e2e_browser: Remote*, *test_user: User*, *authenticated_browser_factory: Callable[[User], Remote]*, *make_request: Callable[[Remote, str, str, Optional[Dict]], Dict]*, *dummy_password: str*)

Test authentication of a new session via other session and password.

django_e2ee.tests.test_views.**test_remote_session_authentication**(*e2e_browser: Remote*, *test_user: User*, *authenticated_browser_factory: Callable[[User], Remote]*, *make_request: Callable[[Remote, str, str, Optional[Dict]], Dict]*)

Test authentication of a new session via other session.

django_e2ee.tests.test_views.**test_session_authentication**(*e2e_browser: Remote*, *test_user: User*, *authenticated_browser_factory: Callable[[User], Remote]*, *make_request: Callable[[Remote, str, str, Optional[Dict]], Dict]*, *dummy_password: str*)

Test authentication of a new session via password.

3.5.2 Submodules

Admin interfaces

This module defines the django-e2ee-framework Admin interfaces.

Classes:

EncryptionKeyAdmin(model, admin_site)	An admin for encryption keys.
EncryptionKeySecretInline(parent_model, ...)	An inline for an encryption key secret.
MasterKeyAdmin(model, admin_site)	An admin for a master key
MasterKeySecretInline(parent_model, admin_site)	An inline for master key secrets.
SessionKeyInline(parent_model, admin_site)	An inline for session keys.

class django_e2ee.admin.**EncryptionKeyAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

An admin for encryption keys.

Methods:

has_add_permission(request)	Return True if the given request has permission to add an object.

Attributes:

inlines
list_display
media

has_add_permission(*request*) → bool

Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

inlines = [<class 'django_e2ee.admin.EncryptionKeySecretInline'>]

list_display = ['uuid', 'created_by']

property media

class django_e2ee.admin.EncryptionKeySecretInline(*parent_model*, *admin_site*)

Bases: StackedInline

An inline for an encryption key secret.

Attributes:

extra
media
readonly_fields

Methods:

has_add_permission(request, obj)	Return True if the given request has permission to add an object.

Models:

model	alias of *EncryptionKeySecret*

extra = 0

has_add_permission(*request*, *obj*) → bool

Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

property media

model

alias of *EncryptionKeySecret* **Miscellaneous:**

DoesNotExist
MultipleObjectsReturned

Model Fields:

encrypted_with	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
encryption_key	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
id	A wrapper for a deferred-loading field.
secret	A wrapper for a deferred-loading field.
signature	A wrapper for a deferred-loading field.
signed_by	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

Attributes:

encrypted_with_id
encryption_key_id
objects
signed_by_id

readonly_fields = ['encrypted_with', 'secret', 'signature', 'signed_by']

class django_e2ee.admin.**MasterKeyAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

An admin for a master key

Methods:

has_add_permission(request)	Return True if the given request has permission to add an object.

Attributes:

inlines
list_display
media
readonly_fields

has_add_permission(*request*) → bool

Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

inlines = [<class 'django_e2ee.admin.MasterKeySecretInline'>, <class 'django_e2ee.admin.SessionKeyInline'>]

list_display = ['user']

property media

readonly_fields = ['pubkey', 'signing_pubkey']

class django_e2ee.admin.**MasterKeySecretInline**(*parent_model*, *admin_site*)

Bases: StackedInline

An inline for master key secrets.

Attributes:

extra
media
readonly_fields

Methods:

has_add_permission(request, obj)	Return True if the given request has permission to add an object.

Models:

model	alias of *MasterKeySecret*

extra = 0

has_add_permission(*request*, *obj*) → bool

Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

property media

model

alias of *MasterKeySecret* **Miscellaneous:**

DoesNotExist
MultipleObjectsReturned

Model Fields:

identifier	A wrapper for a deferred-loading field.
iv	A wrapper for a deferred-loading field.
master_key	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
salt	A wrapper for a deferred-loading field.
secret	A wrapper for a deferred-loading field.
signing_secret	A wrapper for a deferred-loading field.
uuid	A wrapper for a deferred-loading field.

Attributes:

master_key_id
objects

readonly_fields = ['uuid', 'secret', 'signing_secret', 'salt', 'iv']

class django_e2ee.admin.**SessionKeyInline**(*parent_model*, *admin_site*)

Bases: StackedInline

An inline for session keys.

Attributes:

exclude
extra
media
readonly_fields

Methods:

has_add_permission(request, obj)	Return True if the given request has permission to add an object.

Models:

model	alias of *SessionKey*

exclude = ['session']

extra = 0

has_add_permission(*request*, *obj*) → bool

Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

property media

model

alias of *SessionKey* **Miscellaneous:**

DoesNotExist
MultipleObjectsReturned

Methods:

get_absolute_url()

Model Fields:

ignore	A wrapper for a deferred-loading field.
iv	A wrapper for a deferred-loading field.
master_key	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
secret	A wrapper for a deferred-loading field.
session	Accessor to the related object on the forward side of a one-to-one relation.
session_secret	A wrapper for a deferred-loading field.
signing_secret	A wrapper for a deferred-loading field.

Attributes:

master_key_id
objects
session_id

readonly_fields = ['session_secret', 'secret', 'signing_secret', 'iv']

App config

App config for the django_e2ee app.

Classes:

DjangoHelmholtzAaiConfig(app_name, app_module)

class django_e2ee.apps.**DjangoHelmholtzAaiConfig**(*app_name*, *app_module*)

Bases: AppConfig

Attributes:

default_auto_field
label
name

default_auto_field = 'django.db.models.BigAutoField'

label = 'e2ee'

name = 'django_e2ee'

Context processors

Context processors for the django_e2ee app.

Functions:

get_e2ee_login_context_data(request)

django_e2ee.context_processors.**get_e2ee_login_context_data**(*request*) → Dict

Reusable forms of the django-e2ee-framework.

Classes:

E2EESessionForm([data, files, auto_id, ...])	A form to setup E2EE for a session
PasswordCreateForm([data, files, auto_id, ...])	A form for creting the E2EE-password.
PasswordInputForm([data, files, auto_id, ...])	A form for entering the E2EE-password.

class django_e2ee.forms.**E2EESessionForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Bases: *PasswordInputForm*

A form to setup E2EE for a session

Methods:

add_prefix(field_name)	Return the field name with a prefix appended, if this Form has a prefix set.

Attributes:

base_fields	
declared_fields	
media	Return all media required to render the widgets on this form.

add_prefix(*field_name*)

Return the field name with a prefix appended, if this Form has a prefix set.

Subclasses may wish to override.

base_fields = {'method': <django.forms.fields.CharField object>, 'password': <django.forms.fields.CharField object>, 'session_secret': <django.forms.fields.CharField object>, 'uuid': <django.forms.models.ModelChoiceField object>, 'verification_number': <django.forms.fields.IntegerField object>}

declared_fields = {'method': <django.forms.fields.CharField object>, 'password': <django.forms.fields.CharField object>, 'session_secret': <django.forms.fields.CharField object>, 'uuid': <django.forms.models.ModelChoiceField object>, 'verification_number': <django.forms.fields.IntegerField object>}

property media

Return all media required to render the widgets on this form.

class django_e2ee.forms.**PasswordCreateForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Bases: Form

A form for creting the E2EE-password.

Classes:

Media()	

Attributes:

base_fields	
declared_fields	
media	Return all media required to render the widgets on this form.

class Media

Bases: object

Attributes:

js

```
js = ('js/e2ee/submit_form_as_json.js', 'js/e2ee/password_generator.js',
'js/e2ee/fill_random_password.js')
```

```
base_fields = {'identifier':  <django.forms.fields.CharField object>, 'password':
<django.forms.fields.CharField object>, 'show_password':
<django.forms.fields.BooleanField object>}
```

```
declared_fields = {'identifier':  <django.forms.fields.CharField object>,
'password':  <django.forms.fields.CharField object>, 'show_password':
<django.forms.fields.BooleanField object>}
```

property media

Return all media required to render the widgets on this form.

class django_e2ee.forms.**PasswordInputForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Bases: Form

A form for entering the E2EE-password.

Classes:

Media()

Attributes:

base_fields	
declared_fields	
media	Return all media required to render the widgets on this form.

class Media

Bases: object

Attributes:

js

```
js = ('js/e2ee/submit_form_as_json.js',)
```

```
base_fields = {'password':  <django.forms.fields.CharField object>, 'uuid':
<django.forms.models.ModelChoiceField object>}
```

declared_fields = {'password': <django.forms.fields.CharField object>, 'uuid': <django.forms.models.ModelChoiceField object>}

property media

Return all media required to render the widgets on this form.

Permissions

Custom permissions for the restAPI.

Classes:

HasEncryptionSecretPermission()	Check if the user has access to the `e2ee.EncryptionKey`.

class django_e2ee.permissions.**HasEncryptionSecretPermission**

Bases: `BasePermission`

Check if the user has access to the `e2ee.EncryptionKey`.

Methods:

has_object_permission(request, view, obj)	Return *True* if permission is granted, *False* otherwise.

has_object_permission(*request*, *view*, *obj*)

Return *True* if permission is granted, *False* otherwise.

Serializers

Serializers for the django-e2ee-framework app.

Classes:

BulkCreateListSerializer(*args, **kwargs)	
EncryptionKeySecretSerializer(*args, **kwargs)	A serialializer for `e2ee.EncryptionKeySecret`.
EncryptionKeySerializer(*args, **kwargs)	A serialializer for `e2ee.EncryptionKey`.
MasterKeySecretSerializer(*args, **kwargs)	A serialializer for `e2ee.MasterKeySecret`.
MasterKeySerializer(*args, **kwargs)	A serialializer for `e2ee.MasterKey`.
SessionKeySerializer(*args, **kwargs)	A serialializer for `e2ee.SessionKey`.

class django_e2ee.serializers.**BulkCreateListSerializer**(**args*, ***kwargs*)

Bases: `ListSerializer`

Methods:

create(validated_data)

create(*validated_data*)

class django_e2ee.serializers.**EncryptionKeySecretSerializer**(**args*, ***kwargs*)

Bases: ModelSerializer

A serialializer for e2ee.EncryptionKeySecret.

Classes:

Meta()	

Methods:

create(validated_data)	We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

class Meta

Bases: object

Attributes:

fields	
read_only_fields	

Classes:

list_serializer_class	alias of *BulkCreateListSerializer*

Models:

model	alias of *EncryptionKeySecret*

fields = ['encryption_key', 'encrypted_with', 'secret', 'signature', 'signed_by']

list_serializer_class

alias of *BulkCreateListSerializer* **Methods:**

create(validated_data)

model

alias of *EncryptionKeySecret* **Miscellaneous:**

DoesNotExist
MultipleObjectsReturned

Model Fields:

encrypted_with	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
encryption_key	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
id	A wrapper for a deferred-loading field.
secret	A wrapper for a deferred-loading field.
signature	A wrapper for a deferred-loading field.
signed_by	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

Attributes:

encrypted_with_id
encryption_key_id
objects
signed_by_id

```
read_only_fields = ['encryption_key', 'signed_by']
```

create(*validated_data*)

We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

return ExampleModel.objects.create(**validated_data)

If there are many to many fields present on the instance then they cannot be set until the model is instantiated, in which case the implementation is like so:

example_relationship = validated_data.pop('example_relationship') instance = ExampleModel.objects.create(**validated_data) instance.example_relationship = example_relationship return instance

The default implementation also does not handle nested relationships. If you want to support writable nested relationships you'll need to write an explicit *.create()* method.

class django_e2ee.serializers.**EncryptionKeySerializer**(**args*, ***kwargs*)

Bases: ModelSerializer

A serialializer for e2ee.EncryptionKey.

Classes:

Meta()

Methods:

create(validated_data)	We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

class Meta

Bases: object

Attributes:

fields

Models:

model	alias of *EncryptionKey*

fields = ['uuid']

model

alias of *EncryptionKey* **Miscellaneous:**

DoesNotExist
MultipleObjectsReturned

Model Fields:

created_by	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
uuid	A wrapper for a deferred-loading field.

Attributes:

created_by_id	
encryptionkeysecret_set	Accessor to the related objects manager on the reverse side of a many-to-one relation.
master_keys	Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.
objects	

create(*validated_data*)

We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

return ExampleModel.objects.create(**validated_data)

If there are many to many fields present on the instance then they cannot be set until the model is instantiated, in which case the implementation is like so:

example_relationship = validated_data.pop('example_relationship') instance = ExampleModel.objects.create(**validated_data) instance.example_relationship = example_relationship return instance

The default implementation also does not handle nested relationships. If you want to support writable nested relationships you'll need to write an explicit *.create()* method.

class django_e2ee.serializers.**MasterKeySecretSerializer**(**args*, ***kwargs*)

Bases: ModelSerializer

A serialializer for e2ee.MasterKeySecret.

Classes:

Meta()	

Methods:

create(validated_data)	We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

class Meta

Bases: object

Attributes:

fields	

Models:

model	alias of *MasterKeySecret*

fields = ['identifier', 'secret', 'signing_secret', 'salt', 'iv', 'uuid']

model

alias of *MasterKeySecret* **Miscellaneous:**

DoesNotExist
MultipleObjectsReturned

Model Fields:

identifier	A wrapper for a deferred-loading field.
iv	A wrapper for a deferred-loading field.
master_key	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
salt	A wrapper for a deferred-loading field.
secret	A wrapper for a deferred-loading field.
signing_secret	A wrapper for a deferred-loading field.
uuid	A wrapper for a deferred-loading field.

Attributes:

master_key_id
objects

create(*validated_data*)

We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

return ExampleModel.objects.create(**validated_data)

If there are many to many fields present on the instance then they cannot be set until the model is instantiated, in which case the implementation is like so:

example_relationship = validated_data.pop('example_relationship') instance = ExampleModel.objects.create(**validated_data) instance.example_relationship = example_relationship return instance

The default implementation also does not handle nested relationships. If you want to support writable nested relationships you'll need to write an explicit *.create()* method.

class django_e2ee.serializers.**MasterKeySerializer**(**args*, ***kwargs*)

Bases: ModelSerializer

A serialializer for e2ee.MasterKey.

Classes:

Meta()

Methods:

create(validated_data)	We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

class Meta

Bases: object

Attributes:

fields
read_only_fields

Models:

model	alias of *MasterKey*

fields = ['pubkey', 'signing_pubkey', 'user']

model

alias of *MasterKey* **Miscellaneous:**

DoesNotExist
MultipleObjectsReturned

Attributes:

encryptionkey_set	Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.
encryptionkeysecret_set	Accessor to the related objects manager on the reverse side of a many-to-one relation.
masterkeysecret_set	Accessor to the related objects manager on the reverse side of a many-to-one relation.
objects	
pubkey_loaded	The pubkey loaded via cryptography
sessionkey_set	Accessor to the related objects manager on the reverse side of a many-to-one relation.
signed_secrets	Accessor to the related objects manager on the reverse side of a many-to-one relation.
signing_pubkey_loaded	The pubkey loaded via cryptography
user_id	

Model Fields:

pubkey	A wrapper for a deferred-loading field.
signing_pubkey	A wrapper for a deferred-loading field.
user	Accessor to the related object on the forward side of a one-to-one relation.

read_only_fields = ['user']

create(*validated_data*)

We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

return ExampleModel.objects.create(**validated_data)

If there are many to many fields present on the instance then they cannot be set until the model is instantiated, in which case the implementation is like so:

example_relationship = validated_data.pop('example_relationship') instance = ExampleModel.objects.create(**validated_data) instance.example_relationship = example_relationship return instance

The default implementation also does not handle nested relationships. If you want to support writable nested relationships you'll need to write an explicit *.create()* method.

class django_e2ee.serializers.**SessionKeySerializer**(**args*, ***kwargs*)

Bases: ModelSerializer

A serialializer for e2ee.SessionKey.

Classes:

Meta()	

Methods:

create(validated_data)	We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

class Meta

Bases: object

Attributes:

fields	
read_only_fields	

Models:

model	alias of *SessionKey*

fields = ['session', 'session_secret', 'secret', 'signing_secret', 'iv', 'ignore']

model

alias of *SessionKey* **Miscellaneous:**

DoesNotExist
MultipleObjectsReturned

Methods:

get_absolute_url()

Model Fields:

ignore	A wrapper for a deferred-loading field.
iv	A wrapper for a deferred-loading field.
master_key	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
secret	A wrapper for a deferred-loading field.
session	Accessor to the related object on the forward side of a one-to-one relation.
session_secret	A wrapper for a deferred-loading field.
signing_secret	A wrapper for a deferred-loading field.

Attributes:

master_key_id
objects
session_id

```
read_only_fields = ['session']
```

create(*validated_data*)

We have a bit of extra checking around this in order to provide descriptive messages when something goes wrong, but this method is essentially just:

return ExampleModel.objects.create(**validated_data)

If there are many to many fields present on the instance then they cannot be set until the model is instantiated, in which case the implementation is like so:

example_relationship = validated_data.pop('example_relationship') instance = ExampleModel.objects.create(**validated_data) instance.example_relationship = example_relationship return instance

The default implementation also does not handle nested relationships. If you want to support writable nested relationships you'll need to write an explicit *.create()* method.

CHAPTER

FOUR

CONTRIBUTION AND DEVELOPMENT HINTS

The django-e2ee-framework project is developed by the Helmholtz-Zentrum Hereon. It is open-source as we believe that this package can be helpful for multiple other django applications, and we are looking forward for your feedback, questions and especially for your contributions.

- If you want to ask a question, are missing a feature or have comments on the docs, please open an issue at the source code repository
- If you have suggestions for improvement, please let us know in an issue, or fork the repository and create a merge request. See also *Contributing in the development*.

4.1 Contributing in the development

Thanks for your wish to contribute to this app!! The source code of the django-e2ee-framework package is hosted at https://gitlab.hzdr.de/hcdc/django/django-e2ee-framework. It's an open gitlab where you can register via GitHub, or via the Helmholtz AAI. Once you created an account, you can fork this repository to your own user account and implement the changes. Afterwards, please make a merge request into the main repository. If you have any questions, please do not hesitate to create an issue on gitlab and contact the developers.

Warning: For local development, you need a postgres and redis server available. They can be configured via environment variables (e.g. `POSTGRES_HOST` `REDIS_HOST`, see the `settings.py` file in the testproject).

Once you created you fork, you can clone it via

```
git clone https://gitlab.hzdr.de/<your-user>/django-e2ee-framework.git
```

and install it in development mode with the *[dev]* option via:

```
pip install -e ./django-e2ee-framework/[dev]
```

Once you installed the package, run the migrations:

```
cd django-e2ee-framework/
python manage.py migrate
```

which will setup the database for you.

4.1.1 Fixing the docs

The documentation for this package is written in restructured Text and built with sphinx and deployed on readthedocs.

If you found something in the docs that you want to fix, head over to the `docs` folder and build the docs with *make html* (or *make.bat* on windows). The docs are then available in `docs/_build/html/index.html` that you can open with your local browser.

Implement your fixes in the corresponding `.rst`-file and push them to your fork on gitlab.

4.1.2 Contributing to the code

We use automated formatters (see their config in `pyproject.toml` and `setup.cfg`), namely

- Black for standardized code formatting
- blackdoc for standardized code formatting in documentation
- Flake8 for general code quality
- isort for standardized order in imports.
- mypy for static type checking on type hints

We highly recommend that you setup pre-commit hooks to automatically run all the above tools every time you make a git commit. This can be done by running:

```
pre-commit install
```

from the root of the repository. You can skip the pre-commit checks with `git commit --no-verify` but note that the CI will fail if it encounters any formatting errors.

You can also run the pre-commit step manually by invoking:

```
pre-commit run --all-files
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V