Remote Care Documentation

Release 1.0

ExampleInc

Aug 01, 2017

Contents

CHAPTER 1

Main

Remotecare is designed with the aim to automate the regular control of a patient with a chronic disease. Since such diseases can have long periods of low activity the patient might not need a regular (face to face) control appointment when he/she has no complications. Instead the patient could fill in some questionnaires and if necessary plan an e-consult.

CHAPTER 2

Install Remote Care

The steps below show the procedure for getting Remote Care running on a single machine. But with some knowledge of mysql and nginx/apache you should be able to get it running in an environment with separate database and webservers as well.

> **Note:** The install process has been successfully tested on Ubuntu 14.04 lts (x64). But should also work on newer Ubuntu versions and other Debian based Linux distributions. Others (RPM based, other) are also a possibilty but you need to figure out which packages correspond to the DPKG ones and probably need to search Google a lot for fixes concerning the differences between the two.

Step 1: Clone repository Setup directory structure and clone repository:

```
#Create directory, default to /srv/remotecare
sudo mkdir /srv/remotecare/
sudo chown $USER:$USER /srv/remotecare/
mkdir /srv/remotecare/default
cd /srv/remotecare/default

#install git
sudo apt-get install git

#replace #username# with your username for the repository
git clone #username#@10.101.139.250:/srv/git/remotecare.git ./
```

Step 2: Install prerequisites and dependencies Install all needed packages:

```
#Install virtualenv, nginx and uwsgi
#Remove nginx and uwsgi if you only are going to use
#the internal manage.py runserver
sudo apt-get install python-dev python-virtualenv
sudo apt-get install nginx
sudo apt-get install uwsgi uwsgi-plugin-python

#Install package dependencies for Remote Care
```

```
#Does also install yuglify for css/js compression
/srv/remotecare/default/package_dependencies.sh
```

Step 3: Setup virtualenv Setup virtualenv and install pip packages:

```
#create virtualenv
virtualenv /srv/remotecare/virtpy

#activate the virtual env and install requirements
source /srv/remotecare/virtpy/bin/activate
pip install -r /srv/remotecare/default/requirements.txt
```

Step 4: Setup database Very basic setup for getting a database running in mysql:

```
#Setup database (mysql)
mysql -u root -p

#Use different users & passwords and check collation.
CREATE DATABASE remote_care_db;
CREATE USER remote_care@localhost IDENTIFIED BY 'remote_care';
GRANT ALL PRIVILEGES ON remote_care_db.* to remote_care;
FLUSH PRIVILEGES;

#Return to the default user
exit

vim /srv/remotecare/default/remotecare/remotecare/server_settings.py
#Include the settings for the server::
-----------------------------------------------------
DEBUG = False
TEMPLATE_DEBUG = DEBUG

DATABASES = {
    'default': {
    # Add 'postgresql_psycopg2', 'mysql', 'sqlite3' or 'oracle'.
    'ENGINE': 'django.db.backends.mysql',
    # Or path to database file if using sqlite3.
    'NAME': 'remote_care_db',
    'USER': 'remote_care',                      # Not used with sqlite3.
    'PASSWORD': 'remote_care',                  # Not used with sqlite3.
    # Set to empty string for localhost. Not used with sqlite3.
    'HOST': '',
    # Set to empty string for default. Not used with sqlite3.
    'PORT': '',
    },
}
-----------------------------------------------------
```

Step 5: Sync Django models with database and collect static Sync de Django models to the database and/or setup tables:

```
#Note, you need to have the virtualenv activated
cd /srv/remotecare/default/remotecare

#Create datatables
python manage.py migrate

#Insert initial data (hospitals etc.)
```

```
python manage.py loaddata apps/lists/fixtures/initial_data.json
python manage.py loaddata apps/questionnaire/qol/fixtures/initial_data.json
python manage.py loaddata apps/questionnaire/ibd/fixtures/initial_data.json

#Also collect static files
python manage.py collectstatic --noinput

# insert the demo test data
# Manager user auto added email:manager@example.com, pssw:remotecare
python insert_test_data.py

# Django test/development server: python manage.py runserver 0:8000
```

Step 6: Setup uwsgi init script Setup an init script for uwsgi:

```
#Copy simple default uwsgi config & start uwsgi
sudo cp /srv/remotecare/default/remotecare/remotecare/uwsgi.ini /etc/uwsgi/apps-
↪available/remotecare.ini
sudo ln -s /etc/uwsgi/apps-available/remotecare.ini /etc/uwsgi/apps-enabled/
↪remotecare.ini
sudo service uwsgi restart

#See if uswgi runs: Check "ps fax" and "netstat -a"
#Error checking: tail -f /var/log/uwsgi/apps/remotecare.log
```

Step 7: Setup nginx Setup nginx with uswgi:

```
#Copy simple default nginx config & start nginx
sudo cp /srv/remotecare/default/remotecare/remotecare/nginx /etc/nginx/sites-
↪available/default
sudo service nginx restart

#See if nginx runs: Check "ps fax"
#Error checking: tail -f /var/log/nginx/error.log
```

Step 8: Check if website runs The website should now be running at the default IP of the server!

Packages:

CHAPTER 3

core

The core package provides core functionality which is (mostly) not Remote care specific:

- Encryption: provides encryption wrappers based on the Crypto package
- Unittest: defines a baseclass with handy functions for all unitests
- Templatetags: custom template tags
- Serializers: contains a custom JSON serializer
- Backends: e-mail authorization backend
- Forms: default form classes and widgets
- Widgets: default widgets
- Views: default views for auditing purposes
- run_csslint: include this runner if you want to run csslint
- context_processors: default context processor for datetime format
- models: base models and modelfields

Packages:

encryption

The encryption package contains functions for encryption, hashing and getting randomized values.

It makes use of the pycrypto library which needs to be in the requirements.txt.

The symmetric module contains wrapper functions for easy (symmetric) encryption and decryption. It uses AES_256_CBC by default which is included into the AES module.

Modules:

tests.py

Test the encryption and hash functions

Class definitions:

class core.encryption.tests.**Encryption**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Test cases definitions for the encryption functions

check_encryption()
Check the AES256CBC encryption

check_hash()
Check the hash/hmac functions

check_random()
Check that the random functions can be called without exceptions

test_encryption_functions()
Runs the checks

symmetric.py

Contains functions for symmetric encryption/decryption, by default with AES_256_CBC although this can be set to an other encryption algorithm or key length in a later stage.

Note: The encrypted values are returned with the cipher prepended, as following: AES256CBC$encrypted_value

Class and function definitions:

exception core.encryption.symmetric.**EncryptionException**
Bases: exceptions.Exception

This exception is thrown when the cipher is unknown.

core.encryption.symmetric.**get_max_length**(*max_length*, *cipher=<class core.encryption.AES.AES_256_CBC>*)
Get the max_length to set on a field based on the max_length you want to store on the field..

Args:

- max_length: the max_length of the text to store
- cipher: override the default cipher

Returns: The max length of the field to hold the AES encryption. based on the first: 2 ^ x which is high enough.

Raises: EncryptionException: Cipher is unknown

core.encryption.symmetric.**is_encrypted**(*value*)
Check if the value is encrypted by comparing the first characters of the value to the list of known ciphers.

Args:

- value: the value to check (AES256CBC$#encrypted_value#)

Returns: True if the cipher name could be found in the value else False

core.encryption.symmetric.**encrypt**(*plain_text*, *key*, *cipher=default_cipher*)
Encrypt wrapper function

Args:

- plain_text: the text to encrypt
- key: the key to use
- cipher: override the default cipher

Returns: encrypted plain_text by key with use of cipher

Raises: EncryptionException: Cipher is unknown

core.encryption.symmetric.**decrypt**(*cipher_and_encrypted_text*, *key*)

Decrypt wrapper function

the cipher_and_encrypted_text in ciphername$encrypted_text format with use of the key

Args:

- cipher_and_encrypted_text: encrypted value in ciphername$encrypted_text
- key: the key to use

Returns: plain text if key is correct else crap.

Raises: EncryptionException: Cipher is unknown

random.py

Provides a couple of usefull functions for generating random strings by randomly chosing values from a list of possible characters

Function definitions:

core.encryption.random.**randomint**(*minimum*, *maximum*)

Wrapper function for getting a random int

Args:

- minimum: the minimum value
- maximum: the maximum value

Returns: a random integer between (including) minumum, maximum

core.encryption.random.**randombase**(*length=None*, *choices='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopq*

Wrapper function for getting a random choice from a list of choices

Args:

- length: the length of the random string to return
- choices: a list of choices to choose from

Returns: A string with the specified length of random choices

core.encryption.random.**randomid**(*length=64*, *choices='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvv*
+.')

Wrapper function for getting a random id

Produces a 64 length random string with symbols by default.

Args:

- length: the length of the random string to return
- choices: a list of choices to choose from

Returns: A string with the specified length of random choices

core.encryption.random.**randompassword**(*length=12*, *choices='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmn*
+.')

Wrapper function for getting a random password

Produces a 12 length random string with symbols by default.

Args:

- length: the length of the random string to return
- choices: a list of choices to choose from

Returns: A string with the specified length of random choices

core.encryption.random.**randomkey**(*length=8*, *choices='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuv*
Wrapper function for getting a random key

Produces a 8 length random string with no symbols by default. Can be used to store longer values in the session.

Args:

- length: the length of the random string to return
- choices: a list of choices to choose from

Returns: A string with the specified length of random choices

AES.py

Provides a wrapper class on top of the AES implementation in Crypto which handles creating the IV and padding.

Class definitions:

class core.encryption.AES.**AES_256_CBC**
Class for AES_256_CBC encryption

encrypt(*plain_text*, *key*)
Encrypt plaintext with AES 256 CBC (32 bytes) by using the key into a base64 string including the random iv_bytes.

Args:

- plain_text: the text to encrypt
- key: the key to use for encryption

The IV is prepended to the encrypted bytes

The actual used key is the key hashed using SHA256

Returns: IV + plain_text_encrypted_with_key in base64 format

decrypt(*base64_AES_256_CBC_encrypted_text*, *key*)
decrypt base64_AES_256_CBC_encrypted_text with AES 256 CBC (32 bytes) by using the key

Args:

- base64_AES_256_CBC_encrypted_text: the encrypted value with the IV prepended.
- key: the key to use for encryption

The actual used key is the key hashed using SHA256

Returns: plain_text

hash.py

Contains functions for hashing values with use of HMAC or without.

An hash is created as: hash_algorithm$salt$hashed_value

an HMAC is created as: hash_algorirthm$hashed_value

All values are default stored as base64

Class and function definitions:

exception `core.encryption.hash.`**`HashException`**
Bases: `exceptions.Exception`

This exception is thrown when the hash algorithm is unknown.

`core.encryption.hash.`**`check_hmac`**(*secret*, *password*, *hmac_base64*)
Checks whether the hmac(password, secret) is the same as the hmac_base64.

Args:

- secret: the key to use
- password: the value to hash.
- hmac_base64: base64 representation in hash_algorirthm$hashed_value format

Returns: True/False

Raises: HashException: HMAC algorithm unknown

`core.encryption.hash.`**`check_hash`**(*password*, *hash_base64*)
Checks whether the hash(password) is the same as the hmac_base64.

Args:

- password: the value to hash.
- hmac_base64: base64 representation in hash_algorirthm$salt$hashed_value format

Returns: True/False

Raises: HashException: hash algorithm unknown

`core.encryption.hash.`**`create_hmac`**(*secret*, *password*, *hasher=default_hasher*, *iterations= default_iterations*)
Create a hmac from secret and password with ability to set the hasher and number of iterations.

Args:

- secret: the key to use
- password: the value to hash.
- hasher: override default hasher
- iterations: override default iterations.

Returns: hash_algorirthm$hashed_value in base64 format

Raises: HashException: HMAC algorithm unknown

`core.encryption.hash.`**`create_hash`**(*password*, *salt*, *hasher=default_hasher*, *iterations= default_iterations*)
Create a hash of password using salt, hasher and iterations

Args:

- password: the value to hash.
- salt: the salt value.
- hasher: override default hasher
- iterations: override default iterations.

Returns: hash_algorirthm$salt$hashed_value in base64 format

Raises: HashException: Hash algorithm unknown

`core.encryption.hash.`**`do_hash`**(*password*, *hasher*)
Shortcut for crypto hashing function

Args:

- password: the value to hash.
- hasher: override default hasher

Returns: password hashed by hasher

`core.encryption.hash.`**`do_hmac_hash`**(*secret*, *password*, *hasher*)
Shortcut for hmac function

Args:

- secret: the key to use
- password: the value to hash.
- hasher: the hasher to use

Returns: password hashed by hasher with HMAC

unittest

The unittest package includes a class which can be used by as baseclass for unittests.

Modules:

baseunittest.py

Module providing a baseclass for unittests based on 'TestCase'

class `core.unittest.baseunittest.`**`BaseUnitTest`**(*methodName='runTest'*)
Bases: `django.test.testcases.TestCase`

Base unit test based on TestCase includes functions for loading json data and automatically filling in forms by initializing forms with instances and retrieving the post_data from them.

`reset_stores`()
Reset/empty the mailbox and SMS store

`SMS_STORE`

Returns: The sms_store which is an array containing all catched SMS messages

`mail_outbox`

Returns: The mail outbox which contains all catched outgoing e-mails

load_data(*file_name*, *user_data=False*)
Loads the data into self.object using file_name

get(*url*, *status_code=200*, *check_status_code=True*)
Shortcut for self.client.get(url) Automatically checks for a proper status_code (200)

get_fields_of_form(*form*)

Returns: array of all the fields in the fieldset definition for form param

get_post_data(*form*, *prefix=''*)

Returns: a dictionairy with post_data based on what is set as initial on the given form.

post_form(*url*, *initial=None*, *instance=None*, *extra_data={}*, *check_status_code=True*)
Automatic post a form by calling an url and getting the form from the returned response.context

Args:

- url: the url containing the form
- initial: the initial values to set on the form
- instance: the instance to set on the form.
- extra_data: extra data to inlucde into the post_data
- check_status_code: check the response.code? (default=302)

Returns: the response object

get_session_key(*value*)
This method can be used to get the session_key for a healthperson.id

Returns: The session key for the given value.

post(*url*, *post_data*, *status_code=302*, *check_status_code=True*)
Shortcut function for posting data

Args:

- url: the url to post to
- post_data: the post_data to sent
- status_code: the status_code to check for (default=302)
- check_status_code: check the status_code? (default=true)

Returns: the response object

get_model_instance_by_class(*model_class*, *to_strip=None*)
Shortcut function for getting a model instance from the loaded test data based on the model_class.

Args:

- model_class: the class of the model to search for
- to_strip: optional fields that need to be stripped (currently not used)

Returns: the deserialized instance from the test data

login(*email*, *password='remotecare'*, *sms_code='1234'*, *do_sms_code='0'*)
Shortcut function for logging in a user

Args:

- email: the email address to use

- password: the password to use (default=remotecare)
- sms_code: the sms code to use (default=1234)
- do_sms_code: 0 means do not use

Note: In test modus the SMS_STORE is used to simulate the normal procedure of sms authentication.

Returns: the reponse

templatetags

This package contains extra custom filters for Django templates.

Modules:

customfilters.py

This module contains extra custom filters functions for Django templates.

Function definitions:

`core.templatetags.customfilters.`**`default`**(*value*, *arg*)
Function which overrides the default template tag. (Since the original tag would display zero as -)

`core.templatetags.customfilters.`**`comma`**(*value*)
Function which overrides the default template tag. (Since the original tag would display zero as -)

`core.templatetags.customfilters.`**`decrypt`**(*value*, *arg='None'*)
Try to decrypt the value with use of the personal key.

`core.templatetags.customfilters.`**`classname`**(*value*, *arg='None'*)
Get the class_name of the value (=class instance)

`core.templatetags.customfilters.`**`moduleclassname`**(*value*, *arg='None'*)
Get the moduleclassname of the value (=class instance)

`core.templatetags.customfilters.`**`checkgroup`**(*value*, *arg='None'*)
Check which group the value (=user) is in.

`core.templatetags.customfilters.`**`get_random_session_key`**(*value*, *request=None*)
Store a value in the session with a random key used for hiding the id's of all healthpersons.

`core.templatetags.customfilters.`**`get_question_nr`**(*value*, *field=None*)
Get the question number of a question... #Note it would be a speed improvement to move this as a attribute on the fields in the forms.

Modules:

tests.py

Remote care specific test that tests things that were not covered by other tests in the apps.

Note: Needs to be extended with tests for all core code.

Class definitions:

class core.tests.**CoreTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Class with tests for modules in the core package

check_forms()
Checks parts from the forms module

check_models()
Checks parts from the models module

check_widgets()
Checks parts from the widgets module

test_core()
Only checks parts that are not covered by other Remote Care tests

serializers.py

Contains a special serializer with can be used for serializing all fields including inherited fields.

Class definitions:

class core.serializers.**AllFieldsSerializer**
Bases: django.core.serializers.python.Serializer

Supports serialization of fields on the model that are inherited (ie. non-local fields).

serialize(*queryset*, *fields*, ***options*)
Serialize a queryset.

models.py

Includes ModelField definitions.

Class definitions:

class core.models.**ManyToManyField**(**args*, ***kwargs*)
Bases: django.db.models.fields.related.ManyToManyField

Django modelfield for storing many to many relations updates the form_class

formfield(***kwargs*)
Update the formfield

class core.models.**ChoiceOtherField**(*other_field=<class 'django.forms.widgets.TextInput'>*, **args*, ***kwargs*)
Bases: django.db.models.fields.CharField

Django modelfield for allowing to choose from a selectbox or specify an other value

formfield(***kwargs*)
Update the formfield

class core.models.**CheckBoxIntegerField**(**args*, ***kwargs*)
Bases: django.db.models.fields.IntegerField

Integerfield with checkbox as widget

formfield(***kwargs*)
Update the formfield

class core.models.**CheckBoxCharField**(**args*, ***kwargs*)
Bases: django.db.models.fields.CharField

Charfield with checkbox as widget

formfield(***kwargs*)
Update the formfield

class core.models.**DateField**(**args*, ***kwargs*)
Bases: django.db.models.fields.DateField

Django modelfield for storing dates

to_python(*value*)
Remove the time part of the datetime in case unicode or string is given as value

formfield(***kwargs*)
Update the formfield

class core.models.**ImageField**(*verbose_name=None*, *name=None*, *width_field=None*, *height_field=None*, ***kwargs*)
Bases: django.db.models.fields.files.ImageField

Imagefield with maximum size validation

formfield(***kwargs*)
Update the formfield

class core.models.**YesNoChoiceField**(**args*, ***kwargs*)
Bases: django.db.models.fields.NullBooleanField

Django modelfield for storing yes/no choices

formfield(***kwargs*)
Update the formfield

exception core.models.**AuditUserNotDefinedError**
Bases: exceptions.Exception

Error class for showing errors of not supported lookups

class core.models.**ModelAuditMixin**(**args*, ***kwargs*)
Bases: object

A model mixin that tracks model fields' values and provide some useful functions to determine what fields have been changed.

class core.models.**AuditBaseModel**(**args*, ***kwargs*)
Bases: django.db.models.base.Model, *core.models.ModelAuditMixin*

Basemodel which automatically generates audit trails for models.

save(***kwargs*)
Override the save method to include the audit functions

exception core.models.**NotSupportedLookup**(*lookup*)
Bases: exceptions.Exception

Error class for showing errors of not supported lookups

class core.models.**SubfieldBase**
Bases: type

A metaclass for custom Field subclasses. This ensures the model's attribute has the descriptor protocol attached to it.

class core.models.**Creator**(*field*)
Bases: object

A placeholder class that provides a way to set the attribute on the model.

core.models.**make_contrib**(*superclass*, *func=None*)
Returns a suitable contribute_to_class() method for the Field subclass.

If 'func' is passed in, it is the existing contribute_to_class() method on the subclass and it is called before anything else. It is assumed in this case that the existing contribute_to_class() calls all the necessary superclass methods.

class core.models.**HMACField**(**args*, ***kwargs*)
Bases: django.db.models.fields.CharField

HMAC field which stores an HMAC version of the "associated_field" Automatically creates an HMAC hash when saving to the database and when looking up values in the database.

create_hmac(*value*)
Shortcut function for generating a HMAC

Args:

- value: the value to generate a HMAC from

Returns: The HMAC value

pre_save(*model_instance*, *add*)
Called before saving to the database, automatically creates an HMAC of the value of the "associated_field"

Args:

- model_instance: the model_instance to use
- add: newly added True/False

Returns: The value to store in the database

get_db_prep_lookup_old(*lookup_type*, *value*, *connection*, *prepared=False*)
Transform the lookup value in an HMAC. Only allow exact lookups.

Args:

- lookup_type: the name of the lookup
- value: the lookup argument
- connection: the database connection
- prepared: is the value prepared to be used?

Returns: The value to lookup in HMAC format.

Raises: NotSupportedLookup if the lookup_type != exact.

class core.models.**EncryptBaseField**(**args*, ***kwargs*)
Bases: object

Base model field for encrypting the value before sending it to the database and decrypting it before storing it on a model instance.

Provide a 'encryption_key' argument in the kwargs which is either a name of a property on the model instance or a function which returns the encryption_key.

is_encrypted(*value*)
Check if the value is encrypted by comparing the first characters of the value to the list of known ciphers.

Args:

- value: the value to check (AES256CBC$#encrypted_value#)

Returns: True if the cipher name could be found in the value else False

encryption_key(*model_instance*)

Args:

- **obj: the model instance to get the 'self.encryption_key_func'** attribute of.

Returns: The encryption/decryption key to use, either by using a property on a model instance or a function call.

get_db_prep_value(*value*, *connection*, *prepared*)
Override this function so to_python does not get called before saving, which would lead to decrypting the value.

to_python(*value*, *model_instance=None*)
Decrypts the value if it is encrypted

Args:

- value: The value to convert
- model_instance: the model_instance the function is run for, or None

Returns: The decrypted value if encrypted, else the value

pre_save(*model_instance*, *add*)
Encrypt the value if it is encrypted

Args:

- model_instance: the model_instance that is saved
- add: newly added True/False

Returns: The encrypted value to store.

Note: getattr(model_instance, self.attname) calls get_db_prep_value, which normally would call to_python. But to store an encrypted value, this function is overriden.

---

class core.models.**EncryptLookupBaseField**(**args*, ***kwargs*)
Bases: *core.models.EncryptBaseField*

Base modelfield combining both encryption and HMAC lookup. Automatically generates an hmac_#fieldname# modelfield on the model.

Init the field with an "hmac_key" function (for example: hmac_key=lambda:settings.HMAC_KEY) and the 'encryption_key' as used in the *EncryptBaseField*.

get_db_prep_lookup(*lookup_type*, *value*, *connection*, *prepared=False*)
Don't search on encrypted fields!

Raises: NotSupportedLookup

class core.models.**EncryptedHMACLookupCharField**(**args*, ***kwargs*)
Bases: *core.models.EncryptLookupBaseField*, django.db.models.fields.CharField

Encrypted charfield with HMAC lookup

class core.models.**EncryptedHMACLookupTextField**(**args*, ***kwargs*)
Bases: *core.models.EncryptLookupBaseField*, django.db.models.fields.TextField

Encrypted textfield with HMAC lookup

class core.models.**EncryptedHMACLookupEmailField**(**args*, ***kwargs*)
Bases: *core.models.EncryptLookupBaseField*, django.db.models.fields.EmailField

Encrypted emailfield with HMAC lookup

class core.models.**EncryptedCharField**(**args*, ***kwargs*)
Bases: *core.models.EncryptBaseField*, django.db.models.fields.CharField

Encrypted charfield

class core.models.**EncryptedTextField**(**args*, ***kwargs*)
Bases: *core.models.EncryptBaseField*, django.db.models.fields.TextField

Encrypted textfield

class core.models.**EncryptedEmailField**(**args*, ***kwargs*)
Bases: *core.models.EncryptBaseField*, django.db.models.fields.EmailField

Encrypted emailfield

run_csslint.py

forms.py

Standard baseclass form definitions & some widget definitions

Class definitions:

class core.forms.**QuerysetWrapper**(*objects*)
Bases: object

Queryset wrapper for caching the choices coupled to a ManyToManyField. The querysetwrapper provides functions which otherwise should be called on the queryset directly.

Dramatically reduces the amount of queries needed when rendering form fieldsets via a fieldset template.

all()

Returns: All stored objects

none()

Returns: An empty list

filter(***kwargs*)
Filter function which accepts pk and pk__in filters.

Returns: A list of objects

class core.forms.**BaseClassForm**
Bases: object

Base class form which holds the functions shared among all forms. Automatically adds placeholders

add_placeholders_to_fields()
Automatically add the 'placeholder' attribute to appropiate HTML input elements

get_fields()

Returns: array of all the fields in the fieldset definition for form param

fieldsets()
Instead of using fields on forms, fieldsets are used. This allows ordening the fields in sets and hiding/showing different sets based on selected values

queryset_speed_up()
Dramatically decreases the amount of queries necessary in template rendering of ManyToManyFields by replacing the queryset with a *QuerysetWrapper* instance.

If not replaced, every time the (BoundField) 'field' variable is used in the template the choices are retrieved from the database, resulting in some cases in 40+ queries.

Execute this function in the __init__ of forms that have one or more instances of *ModelMultipleChoiceField*

class core.forms.**BaseForm**(**args*, ***kwargs*)
Bases: django.forms.forms.Form, *core.forms.BaseClassForm*

Baseclass form which is based on forms.Form

class core.forms.**BaseModelForm**(**args*, ***kwargs*)
Bases: django.forms.models.ModelForm, *core.forms.BaseClassForm*

Baseclass form which is based on forms.ModelForm

class core.forms.**DisplayWidget**(*attrs=None*)
Bases: django.forms.widgets.Widget

Widget for displaying values

class core.forms.**ImageField**(**args*, ***kwargs*)
Bases: django.forms.fields.ImageField

Override ImageField to allow setting a maximum upload size and checking mime_type with help of the 'magic' package.

class core.forms.**MultipleChoiceField**(*choices=()*, *required=True*, *widget=None*, *label=None*, *initial=None*, *help_text=u''*, **args*, ***kwargs*)
Bases: django.forms.fields.MultipleChoiceField

Django formfield for multiple selections

class core.forms.**ModelMultipleChoiceField**(*queryset*, *required=True*, *widget=None*, *label=None*, *initial=None*, *help_text=u''*, **args*, ***kwargs*)
Bases: django.forms.models.ModelMultipleChoiceField

Django formfield for multiple selections

class core.forms.**ChoiceOtherWidget**(*choices*, *other_field*, *maxlength=128*, *attrs=None*)
Bases: django.forms.widgets.MultiWidget

Django widget for choice other fields

Displays a select box with the option for 'other' allowing to specify a value via a textinput

choices()

Returns: The widget choices

decompress(*value*)

Returns: an array with the value or ['other', value]

format_output(*rendered_widgets*)

Returns: the rendered widgets seperated by a breakline

class core.forms.**ChoiceOtherField**(*choices=[]*, **args*, ***kwargs*)
Bases: django.forms.fields.MultiValueField

Django formfield for choice other fields

fix_value_from_post(*post_data*, *field_name*)
Used for fixing post data so it can be temporarily stored in the database and later used as form initial data

get_value_from_post(*post_data*, *field_name*)
Get the value from the dropbox except when 'other' is selected if 'other' use the value from the textinput

clean(*value*)
Raise error messages if needed

compress(*data_list*)
data_list is a list of two items. The first item is the value from the TypeChoiceField and the second one is from the text input.

class core.forms.**FormRadioSelect**(*attrs=None*, *choices=()*)
Bases: django.forms.widgets.RadioSelect

Django formfield which overrides the radio select field

class core.forms.**SelectDateWidgetCustom**(*attrs=None*, *years=None*, *format=None*, *required=True*)
Bases: *core.widgets.SelectDateWidget*

Base class for SelectDateWidget, SelectDateTimeWidget and SelectTimeWidget.

render(*name*, *value*, *attrs=None*)
Return the html code of the widget.

class core.forms.**DateWidget**(*attrs=None*, *years=None*, *format=None*, *required=True*)
Bases: *core.forms.SelectDateWidgetCustom*

Django Widget for selecting dates

render(*name*, *value*, *attrs=None*)
Render datewidget

class core.forms.**FormDateField**(**args*, ***kwargs*)
Bases: django.forms.fields.DateField

Django Formfield for dates

widget
alias of *DateWidget*

fix_value_from_post(*post_data*, *field_name*)
Used for fixing post data so it can be temporarily stored in the database and later used as form initial data

get_value_from_post(*post_data*, *field_name*)
Try to get the value from post_data, used to temporarily store the value.

clean(*value*)
Date validation checks

widget_attrs(*widget*)
Add datefield CSS class by default

class core.forms.**YesNoChoiceField**(**args*, ***kwargs*)
Bases: django.forms.fields.NullBooleanField

Django formfield for yes/no selections

widget
alias of NullBooleanSelect

context_processors.py

Module with provides shortcuts for datetime definitions

Function definitions:

core.context_processors.**datetime_format**(*request*)
Custom datetime format definitions

widgets.py

Provides extra HTML Widget classes

class core.widgets.**SelectDateWidget**(*attrs=None*, *years=None*, *format=None*, *required=True*)
Bases: django.forms.widgets.Widget

A Widget that splits date input into three <select> boxes.

This also serves as an example of a Widget that has more than one HTML element and hence implements value_from_datadict.

classmethod id_for_label(*id_*)

Returns: the label for this widget

value_from_datadict(*data*, *files*, *name*)

Returns: the values from the post data based associated with this widget

parse_value(*val*)

Returns: An dict with month, day, year based on val or an empty dict

parse_format(*fmt*)
Parse the given format *fmt* and set the format property.

month_choices(*fmt*)
Return list of choices (tuple (key, value)) for monthes select.

day_choices(*fmt*)
Return list of choices (tuple (key, value)) for days select.

year_choices(*fmt*)
Return list of choices (tuple (key, value)) for years select.

backends.py

The standard email backend is replaced by a custom ModelBackend that supports getting the user based on the stored hmac email value.

Class definitions:

class core.backends.**EmailBackend**
Bases: django.contrib.auth.backends.ModelBackend

Custom authentication backend which uses the hmac email address rather than the username to authenticate.

authenticate(*email=None*, *password=None*, *username=None*, ***kwargs*)
Processes an authentication attempt

args:

- email: not used
- password: the password to check
- username: the plain-text email address to search for

Returns: the user if found and password correct else None.

views.py

Module containing a baseclass for all formviews and a function for serving files via the X-Sendfile directive.

Class and function definitions:

class core.views.**FormView**(***kwargs*)
Bases: django.views.generic.edit.FormView

Baseclass formview which automatically adds a 'changed_by_user' property to the form instance which is the logged in user.

Used for getting the logged in user from the view to model instance via a form.

get_form_kwargs()
Automatically add the changed_by_user to the form kwargs. This is picked up in the baseclass form and set on the form itselves and later in the save method of the form to the model.

This way the self.request.user is pushed via a form to the model to be used in the audit trail.

core.views.**xsendfileserve**(*request*, **args*, ***kwargs*)
Serve static files below a given point in the directory structure.

Uses X-Sendfile to sent the file to the client via the webserver, allowing authentication for media files.

To use, put a URL pattern such as:

```
(r'^(?P<path>.*)$', 'apps.core.views.xsendfileserve',
{'document_root' : '/path/to/my/files/'})
```

in your URLconf. You must provide the document_root param. You may also set show_indexes to True if you'd like to serve a basic index of the directory. This index view will use the template hardcoded below, but if you'd like to override it, you can create a template called static/directory_index.html.

Nginx default setup (include in sites-available config file(s)):

```
location /xsendmedia {
    internal;
    alias /full/path/to/remotecare/media;
}
```

compressors.py

class core.compressors.**CSSMinCompressor**(*verbose*)

Bases: pipeline.compressors.CompressorBase

Compress CSS wrapper for cssmin

compress_css(*css*)

Return compressed CSS

CHAPTER 4

apps

The “app” package contains all Remote Care apps divided accordingly to specific parts of Remote Care:

- Audit: contains a model for storing auditing information
- Information: general information pages, help and feedback
- Account: provides the custom user model
- Appointment: allows the secretary to add appointments
- Healthperson: contains all different roles and functions for these roles
- Service: reminder functions which should be executed periodically
- Utils: general class providing util functions
- Lists: hospital model for storing hospitals
- RCmessages: functions, views and model for messages in Remote care
- Report: model and views for adding/editing reports
- Questionnaire: all functions/models/views for filling in questionnaires

Packages:

audit

Contains a model for storing auditing information.

The audit information, which include the datetime, edited by user and a json containing the actually changes made is automatically stored when a model is saved. For this purpose the logged in user is automatically passed from the view to the form and eventually to the instance that is saved.

By using the baseclass *core.models.AuditBaseModel* models can be automatically set to store audit trails.

Modules:

models.py

class apps.audit.models.**LogEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Stores a change to a model instance in json format.

Parameters

- **id** (*AutoField*) –
- **added_on** (*DateTimeField*) –
- **added_by_id** (ForeignKey to *User*) –
- **encryption_key_id** (ForeignKey to *EncryptionKey*) –
- **json** (*TextField*) –

information

This Remote Care app contais views for displaying all kinds of information to an user and a form for giving feedback.

The help functionality is included as well.

Modules:

tests.py

Test the feedback system and check that the predefined information pages give a 200

Class definitions:

class apps.information.tests.**InformationAndFeedbackTest**(*methodName='runTest'*)
Bases: *core.unittest.baseunittest.BaseUnitTest*

Information and feedback tests

check_feedback()
Check if the feedback process/form works as expected

test_information_and_feedback()
Simple test for information pages (result_code=200)

forms.py

Provides a feedback form

Class definitions:

class apps.information.forms.**FeedBackAddEditForm**(**args*, ***kwargs*)
Bases: *core.forms.BaseForm*

Form for adding Feedback

views.py

Class based view definitions for showing information and submitting feedback

Class definitions:

class apps.information.views.**BaseInformationPageView**(***kwargs*)
Bases: django.views.generic.base.TemplateView

Basic view for information based template views accepts a list of templates and picks the correct template based on the page parameter

class apps.information.views.**InformationPageView**(***kwargs*)
Bases: *apps.information.views.BaseInformationPageView*

Information page view class, used for displaying all kinds of information in Remote Care

class apps.information.views.**AboutSecurityView**(***kwargs*)
Bases: django.views.generic.base.TemplateView

Shows a page with information about security

class apps.information.views.**InformationFeedBack**(***kwargs*)
Bases: django.views.generic.edit.FormView

Feedback form class, e-mails feedback to info@example.com

form_class
alias of FeedBackAddEditForm

send_feedback_email(*email_content*)
Sent a feedback email to info@example.com by default

form_valid(*form*)
Sent feedback and redirect user

class apps.information.views.**InformationFeedbackSentView**(***kwargs*)
Bases: django.views.generic.base.TemplateView

Shows a feedback has been succesfully sent page

account

All account information, including the User model is stored in this package. It also contains base profile edit forms which provide validation for e-mail address and mobile numbers.

Modules:

tests.py

Contains all tests for login and User model.

Class definitions:

class apps.account.tests.**LoginTest**(*methodName='runTest'*)
Bases: *core.unittest.baseunittest.BaseUnitTest*

Tests the login functions, login block after to many incorrect logins and functions on the User model that were not covered.

check_logins()
Checks the basic tests logins

check_block_system()
Checks login block system which temporarily blocks the user after to many invalid login attempts.

check_invalid_login()
Check that an user cannot login with a faulty password

check_model()
Check extra property and functions on the User model

test_check_logins()
Test the login features

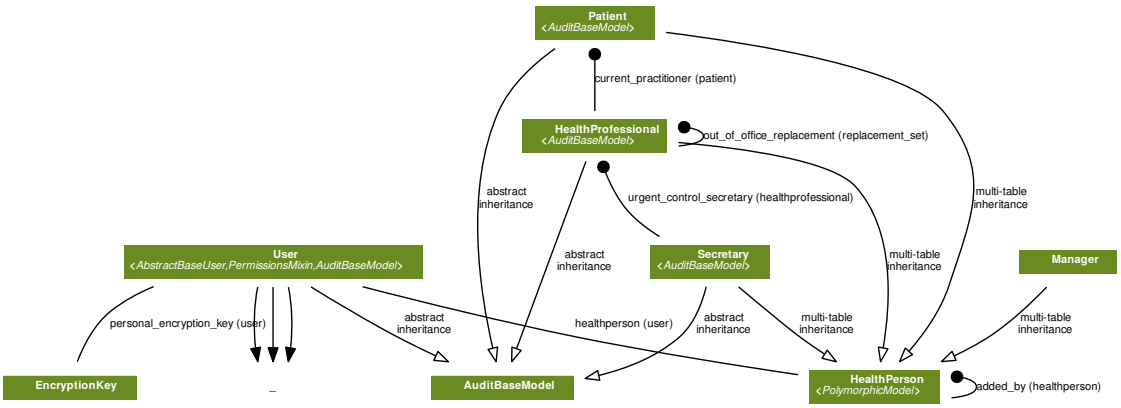
models.py

This module contains the User model and other models for storing login attempts, a list of old hashed passwords, password change request and temporarily storages of the (hashed) login sms code.

An user is coupled to one of the 4 different possible roles in Remote Care:

- Patient
- Healthprofessional
- Secretary
- Manager

Via the polymorphicmodel: *apps.healthperson.models.HealthPerson* as can be seen in the next model relationship diagram:



All personal user data is stored encrypted and, if searchable, also hashed. This key for encryption/decryption is the encryption_key of the user which is encrypted/decrypted with the MASTER_KEY in the settings file. The hashes are in HMAC format. The HMAC secrets are stored in the settings file.

Users are hospital 'bound' meaning that an healthprofessional/secretary can only find patients within the same hospital during searching.

Class definitions:

class apps.account.models.**UserManager**
Bases: django.contrib.auth.base_user.BaseUserManager

Custom user manager which allows adding an user via the manage.py command using email as unique key and filling in other required information

create_user(*email=None*, *password=None*, ***extra_fields*)
Creates and saves a User with the given username, email and password. note: encrypted_email is not encrypted yet here, will be encrypted when the user is saved.

create_superuser(*email*, *password*, ***extra_fields*)
Creates and saves a super user with the same parameters as create_user

class apps.account.models.**EncryptionKey**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Stores all encryption keys of the users.

Encrypts/decrypts them with the MASTER_KEY setting.

Parameters

- **id** (*AutoField*) –
- **key** (*EncryptedCharField*) –

class apps.account.models.**User**(**args*, ***kwargs*)
Bases: django.contrib.auth.base_user.AbstractBaseUser, django.contrib.auth.models.PermissionsMixin, *core.models.AuditBaseModel*

Custom user model which saves basic information about the user. All private information is encrypted.

Private information is stored encrypted in the database via encrypted model fields. See the *core.models.EncryptBaseField* for more information on encryption.

Private information that should also be searchable is represented by both an encrypted field and an HMAC field. See the *core.models.EncryptLookupBaseField* for more information on encryption and HMAC lookup.

Parameters

- **id** (*AutoField*) –
- **password** (*CharField*) –
- **last_login** (*DateTimeField*) –
- **is_superuser** (*BooleanField*) –
- **personal_encryption_key_id** (OneToOneField to *EncryptionKey*) –
- **hmac_first_name** (*HMACField*) –
- **first_name** (*EncryptedHMACLookupCharField*) –
- **hmac_last_name** (*HMACField*) –
- **last_name** (*EncryptedHMACLookupCharField*) –
- **hmac_email** (*HMACField*) –
- **email** (*EncryptedHMACLookupEmailField*) –
- **title** (*CharField*) – choices=[mr, ms, dr, prof]
- **initials** (*CharField*) –
- **prefix** (*CharField*) –

- **mobile_number** (*EncryptedCharField*) –
- **gender** (*CharField*) – choices=[male, female]
- **hospital_id** (ForeignKey to *Hospital*) –
- **hmac_local_hospital_number** (*HMACField*) –
- **local_hospital_number** (*EncryptedHMACLookupCharField*) –
- **hmac_BSN** (*HMACField*) –
- **BSN** (*EncryptedHMACLookupCharField*) –
- **date_of_birth** (*DateField*) –
- **healthperson_id** (OneToOneField to *HealthPerson*) –
- **is_staff** (*BooleanField*) –
- **is_active** (*BooleanField*) –
- **date_joined** (*DateTimeField*) –
- **account_blocked** (*YesNoChoiceField*) –
- **deleted_on** (DateField) –

new_questionnaire_request
Returns true if the patient has no questionnaire requests

new_message_count
Returns the amount of unread messages

full_name
Returns the full_name of an user which is: first_name + prefix + last_name

professional_name
Returns the full_name of an user which is: initials + prefix + last_name

is_deleted
Returns True if the user has been set for deletion.

audit_encryption_key_id
Get the EncryptionKey id so it can be coupled to the log item in the audit.

Returns: The id of the EncryptionKey that is used to encrypt the model instance.

encryption_key
Get the encryption key of the user.

Returns: The encryption key of the user instance.

class apps.account.models.**LoginAttempt**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Stores all login attempts for administration purposes. Login attempts with extra info and an hash of the username = email address

Parameters

- **id** (*AutoField*) –
- **succesfull** (*YesNoChoiceField*) –
- **ipaddress** (*CharField*) –
- **useragent** (*TextField*) –

- **extra_info** (*TextField*) –
- **session_id** (*TextField*) –
- **date** (*DateTimeField*) –
- **username_hash** (*TextField*) –

class apps.account.models.**LoginSMSCode**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Temporarily stores the hmac_sms_code used during login

Parameters

- **id** (*AutoField*) –
- **user_id** (ForeignKey to *User*) –
- **hmac_sms_code** (*CharField*) –

class apps.account.models.**OldPassword**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Stores previous passwords and the current password. Can both be used to validate that the password is different from the # last passwords and check if the password is expired.

Parameters

- **id** (*AutoField*) –
- **user_id** (ForeignKey to *User*) –
- **password_hash** (*CharField*) –
- **date_added** (*DateField*) –

class apps.account.models.**PasswordChangeRequest**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Temporarly stores the sms_code and key for resetting the password. Email and sms HMAC thus when set only can check using HMAC secret. The attempt_nr field is used to limit the total amount of attempts possible.

Parameters

- **id** (*AutoField*) –
- **user_id** (ForeignKey to *User*) –
- **hmac_email_key** (*CharField*) –
- **hmac_sms_code** (*CharField*) –
- **added_on** (*DateField*) –
- **attempt_nr** (*IntegerField*) –

class apps.account.models.**AgreedwithRules**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Stores whether the user agreed with the rules for using Remotecare.

Parameters

- **id** (*AutoField*) –
- **user_id** (OneToOneField to *User*) –
- **dateofagreement** (*DateTimeField*) –

forms.py

The forms in this module can be used as a profile editing form baseclass for easy including e-mail and mobile number validation.

Class definitions:

class apps.account.forms.**BaseProfileEditForm**(**args*, ***kwargs*)
Bases: *core.forms.BaseModelForm*

Base class profile-editing-form including (repeat) validators for mobile number and e-mail and clean method including validation logic.

Note: Does not include a fieldsets definition in the meta class, this needs to be set in the form that uses this class as baseclass.

---

class apps.account.forms.**SetPasswordForm**(**args*, ***kwargs*)
Bases: *core.forms.BaseForm*

This form class let's users set their password after logging in automatically via the API. (Used by Healthprofessionals)

class apps.account.forms.**BasePasswordProfileEditForm**(**args*, ***kwargs*)
Bases: *apps.account.forms.BaseProfileEditForm*

This form class is an extension of the BaseProfileEditForm which provides change password options

class apps.account.forms.**AgreeWithRulesForm**(**args*, ***kwargs*)
Bases: django.forms.forms.Form

Form which shows an 'agree with the rules' select box which can be used for accepting the rules for using the application.

appointment

Appointments can be requested by a patient during filling in the questionnaires or by a healthprofessional.

The appointment is created by a secretary. The patient is automatically notified when the appointment has been created.

This package contains all logic for adding/editing and viewing appointments

Modules:

tests.py

Module provides tests for adding appointments

Class definitions:

class apps.appointment.tests.**AppointmentTest**(*methodName='runTest'*)
Bases: *core.unittest.baseunittest.BaseUnitTest*

Test the appointment functionality by adding a appointment for a controle and an urgent controle

check_appointment()
Add an appointment and check that it has been saved correctly

test_adding_appointment()
Run the check for both a non-urgent control and urgent-control

models.py

Appointments can be requested by a patient during filling in the questionnaires or by a healthprofessional.

The appointment is created by a secretary and coupled to a healthprofessional. The patient is automatically notified when the appointment has been created.

Class definitions:

class apps.appointment.models.**Appointment**(**args*, ***kwargs*)
Bases: *core.models.AuditBaseModel*

An appointment is coupled to a questionnaire_request which couples it to a patient. The appointment is created by a secretary and takes place with a healthprofessional.

Parameters

- **id** (*AutoField*) –
- **questionnaire_request_id** (ForeignKey to *QuestionnaireRequest*) –
- **created_by_id** (ForeignKey to *Secretary*) –
- **created_on** (DateField) –
- **appointment_date** (*DateField*) –
- **appointment_hour** (*CharField*) – choices=[01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]
- **appointment_minute** (*CharField*) – choices=[00, 05, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55]
- **appointment_healthprofessional_id** (ForeignKey to *HealthProfessional*) –

get_day
Returns te day of the week (monday, tuesday, etc)

forms.py

Contains the form for adding and editing appointments

Class and function definitions:

apps.appointment.forms.**change_empty_choice**(*field*, *to_set*)
Change the empty choice (first entry) of a Select field

class apps.appointment.forms.**AppointmentAddEditForm**(**args*, ***kwargs*)
Bases: *core.forms.BaseModelForm*

Form class for adding/editing appointments

views.py

Module containing a view for adding/editing appointments

Class definitions:

class apps.appointment.views.**AppointmentEdit**(**kwargs)
Bases: *apps.healthperson.patient.views.PatientBaseView*, *core.views.FormView*

Class based view for adding/editing appointment information by a secretary

form_class
alias of AppointmentAddEditForm

render_sms_and_template(*appointment*, *message_template*, *sms_template*)
Renders both sms and template at once since the context is the same.

Args:

- appointment: the appointment instance
- message_template: the template_name to load for the message
- sms_template: the template_name to load for the sms

Returns: [message_template, sms_template]

get_default_urgent_message_and_sms(*appointment*)
Renders both sms and template at once for urgent controls

Args:

- appointment: the appointment instance

Returns: [message_template, sms_template]

get_default_message_and_sms(*appointment*)
Renders both sms and template at once for non-urgent controls

Args:

- appointment: the appointment instance

Returns: [message_template, sms_template]

form_valid(*form*)
Save the appointment and notify the patient of the planned appointment

healthperson

This package includes the different user roles within Remote Care.

- Patient
- Healthprofessional
- Secretary
- Manager

The different models, views, urls and forms specific for every role are included in the different packages below.

Packages:

healthprofessional

Remote Care allows healthprofessionals to automate the periodically control of a chronic patient. The process includes adding new patients and configuring the period used for control and which questionnaires should be used.

When a patient has filled in the questionnaires for a control, the healthprofessional can login and view the information filled in. The control can be finished by editing an automatic generated report and message and clicking the 'finish' button. After this step the message is sent to the patient and the report can be exported in docX or PDF.

Modules:

tests.py

Unittest for checking the healthprofessional functionality

Class definitions:

class apps.healthperson.healthprofessional.tests.**HealthprofessionalTest**(*methodName='runTest'*)
Bases: *core.unittest.baseunittest.BaseUnitTest*

Test message sending, searching, and all patient related pages like reports/messages etc.

check_sent_messages()
Checks if healthprofessional can sent a message to a patient and if it can be found via the search page.

check_search()
Check general search for patients

check_patient_pages()
Check pages which can be opened by an healhtprofessional for a patient including controls. appointments, reports and filled in questionnaire information

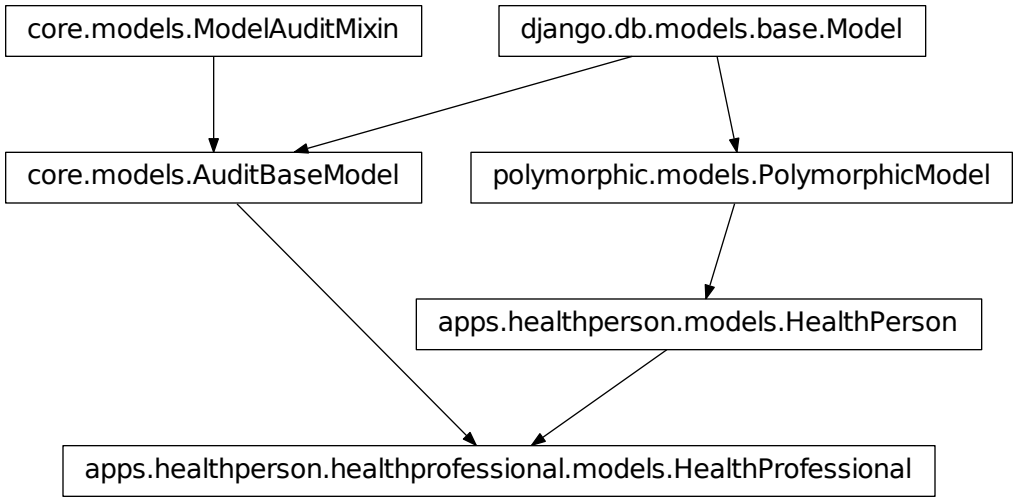
test_healthprofessional_functions()
Test runner for all healhtprofessional checks

> **Note:** Adding and editing reports, messages etc. are not tested here but in their own apps.

models.py

This module contains the healthprofessional model definition. The healthprofessional is coupled to a *apps.account.models.User* instance via the *apps.healthperson.models.HealthPerson* baseclass.

Inheritance-diagram:



Class definitions:

class apps.healthperson.healthprofessional.models.**HealthProfessional**(**args*, ***kwargs*)

Bases: *apps.healthperson.models.HealthPerson*, *core.models.AuditBaseModel*

Stores specific information healthprofessional including photo, function, specialism and notifications and out of office settings.

Parameters

- **id** (*AutoField*) –
- **polymorphic_ctype_id** (ForeignKey to ContentType) –
- **added_on** (DateField) –
- **added_by_id** (ForeignKey to *HealthPerson*) –
- **healthperson_ptr_id** (OneToOneField to *HealthPerson*) –
- **photo_location** (*ImageField*) –
- **function** (*CharField*) – choices=[specialist, assistant, specializednurse, nurse, dietician]
- **specialism** (*CharField*) – choices=[gastro_liver_disease, rheumatology, surgery, internal_medicine, orhopedie]
- **telephone** (*CharField*) –
- **urgent_control_notification** (*CharField*) – choices=[sms_and_email, sms_only, email_only, to_secretary]
- **urgent_control_secretary_id** (ForeignKey to *Secretary*) –
- **out_of_office_start** (*DateField*) –
- **out_of_office_end** (*DateField*) –

- **out_of_office_replacement_id** (ForeignKey to *HealthProfessional*) –

forms.py

This module provides all the forms necessary for the functionality for an healthprofessional.

Class definitions:

class apps.healthperson.healthprofessional.forms.**HealthProfessionalSearchForm**(**args*, ***kwargs*)

Bases: *core.forms.BaseForm*

Search for a healthprofessional by either first_name, last_name, function or specialism Only provides the form & fields, all logic is in the view.

class apps.healthperson.healthprofessional.forms.**HealthProfessionalPhotoForm**(**args*, ***kwargs*)

Bases: *core.forms.BaseModelForm*

Provides a form for adding/editing the photo, using an ImageField

class apps.healthperson.healthprofessional.forms.**HealthProfessionalOutOfOfficeEditForm**(**args*, ***kwarg*

Bases: *core.forms.BaseModelForm*

Set the out-of-office information, this includes a period and an replacement healthprofessional during that period.

class apps.healthperson.healthprofessional.forms.**HealthProfessionalNotificationEditForm**(**args*, ***kwa*

Bases: *core.forms.BaseModelForm*

Edit the notification of new messages/finished controls

class apps.healthperson.healthprofessional.forms.**HealthProfessionalEditForm**(**args*, ***kwargs*)

Bases: *apps.account.forms.BasePasswordProfileEditForm*

Edit healthprofessional information. Add specific healthprofessionals fields on top of the default User fields.

class apps.healthperson.healthprofessional.forms.**HealthProfessionalAddForm**(**args*, ***kwargs*)

Bases: *apps.account.forms.BaseProfileEditForm*

Add healthprofessional information. Add specific healthprofessionals fields on top of the default User fields.

views.py

This module contains all the views which are used by the manager to add/edit healthprofessionals and the views used by the healthprofessional itselves.

Class definitions:

class apps.healthperson.healthprofessional.views.**HealthProfessionalBaseView**(***kwargs*)

Bases: django.views.generic.base.View

Base view which adds the healthprofessional by using the healthprofessional_session_id or logged in user

dispatch(**args*, ***kwargs*)

Adds healthprofessional to the view class

get_context_data(***kwargs*)

Base context, include the healthprofessional by default

class apps.healthperson.healthprofessional.views.**HealthProfessionalIndexView**(***kwargs*)
Bases: apps.base.views.BaseIndexTemplateView, *apps.healthperson.healthprofessional.views.HealthProfessionalBaseView*

This view shows the homepage of the healthprofessional

get_controles_for_healthprofessional(*healthprofessional*)
Adds controles and urgent_patient_controles to the view so they can be shown in the overview.

Args:

- healthprofessional: The healthprofessional to get all all controles for

class apps.healthperson.healthprofessional.views.**SearchView**(***kwargs*)
Bases: django.views.generic.base.TemplateView

Generic search page as available in the homepage

dispatch(**args*, ***kwargs*)
Init default values to be used in the context

post(*request*, **args*, ***kwargs*)
Search for patients

get_context_data(***kwargs*)
Return the found patients in a context

class apps.healthperson.healthprofessional.views.**HealthProfessionalCropPhoto**(***kwargs*)
Bases: *apps.healthperson.healthprofessional.views.HealthProfessionalBaseView*, django.views.generic.base.TemplateView

View allows to crop the photo of the healthprofessional, if necessary.

class apps.healthperson.healthprofessional.views.**HealthProfessionalEditPhoto**(***kwargs*)
Bases: *apps.healthperson.healthprofessional.views.HealthProfessionalBaseView*, *core.views.FormView*

Add/edit or remove the photo of an healthprofessional

form_class
alias of HealthProfessionalPhotoForm

class apps.healthperson.healthprofessional.views.**HealthProfessionalPhotoView**(***kwargs*)
Bases: *apps.healthperson.healthprofessional.views.HealthProfessionalBaseView*, django.views.generic.base.TemplateView

Show photo of healthprofessional

class apps.healthperson.healthprofessional.views.**HealthProfessionalNotificationView**(***kwargs*)
Bases: *apps.healthperson.healthprofessional.views.HealthProfessionalBaseView*, django.views.generic.base.TemplateView

Show notification settings

class apps.healthperson.healthprofessional.views.**HealthProfessionalOutOfOfficeView**(***kwargs*)
Bases: *apps.healthperson.healthprofessional.views.HealthProfessionalBaseView*, django.views.generic.base.TemplateView

Show out of office settings

class apps.healthperson.healthprofessional.views.**HealthProfessionalOutOfOfficeEdit**(***kwargs*)
Bases: *apps.healthperson.healthprofessional.views.HealthProfessionalBaseView*, *core.views.FormView*

Edit the out of office settings for an healthprofessional.

These settings are used to configure an out of office period with a replacement.

form_class
alias of HealthProfessionalOutOfOfficeEditForm

class apps.healthperson.healthprofessional.views.**HealthProfessionalNotificationEdit**(***kwargs*)
Bases: *apps.healthperson.healthprofessional.views.HealthProfessionalBaseView*, *core.views.FormView*

Edit the notification settings for an healthprofessional.

These settings are used for sending notifications of unhandeld (urgent) controls.

form_class
alias of HealthProfessionalNotificationEditForm

class apps.healthperson.healthprofessional.views.**HealthProfessionalPersonaliaView**(***kwargs*)
Bases: *apps.healthperson.healthprofessional.views.HealthProfessionalBaseView*, django.views.generic.base.TemplateView

Shows the personalia of an healhtprofessional which is the information stored in the coupled *apps.account.models.User* instance.

class apps.healthperson.healthprofessional.views.**HealthProfessionalSetPassword**(***kwargs*)
Bases: *apps.healthperson.healthprofessional.views.HealthProfessionalBaseView*, *core.views.FormView*

Displays a form to set a password. Used to initialize password for an healthprofessional

form_class
alias of SetPasswordForm

class apps.healthperson.healthprofessional.views.**HealthProfessionalPersonaliaEdit**(***kwargs*)
Bases: *apps.healthperson.healthprofessional.views.HealthProfessionalBaseView*, *core.views.FormView*

Edit the personalia of an healhtprofessional which is the information stored in the coupled *apps.account.models.User* instance.

form_class
alias of HealthProfessionalEditForm

class apps.healthperson.healthprofessional.views.**HealthProfessionalSearchView**(***kwargs*)
Bases: *core.views.FormView*

Search for an healthprofessional as a manager

form_class
alias of HealthProfessionalSearchForm

get_context_data(***kwargs*)
Add the search results to the context

get_initial()
Get initial data, used for showing the old results and form data when the user uses the back button to return to the form

get(*request*, **args*, ***kwargs*)
Re-execute the search if the user has used the back button

form_valid(*form*)
Perform the search action, search for a healthprofessional

class apps.healthperson.healthprofessional.views.**HealthProfessionalAddView**(***kwargs*)
Bases: *apps.healthperson.views.BaseAddView*

Class based view for adding a new healthprofessional

form_class
alias of HealthProfessionalAddForm

class apps.healthperson.healthprofessional.views.**HealthProfessionalRemove**(***kwargs*)
Bases: *apps.healthperson.healthprofessional.views.HealthProfessionalBaseView*, django.views.generic.base.TemplateView

Remove the healthprofessional by setting the deleted_on attribute on the coupled *apps.account.models.User* instance.

post(*request*, **args*, ***kwargs*)
Remove the healthprofessional by setting the user to inactive

management

Managers can add, edit and remove patients, healthprofessionals and secretariat in Remote Care. They have no access to patient information other than their personalia.

Modules:

tests.py

This module contains tests which can be performed by a manager. Since the manager has access to all add/edit/remove user functionality these functions are all tested here.

Class definitions:

class apps.healthperson.management.tests.**ManagementTest**(*methodName='runTest'*)
Bases: *core.unittest.baseunittest.BaseUnitTest*

Provides the tests performed by a logged in manager

check_search(*url*, *post_data*, *context_key*, *full_name*)
Check the default search page for different search criteria

Args:

- url: the url to open
- post_data: the data to post to that url
- context_key: the key to look voor in the context after posting
- full_name: the full_name of the user that should be found

Returns: The response after posting the data

check_personalia()
Checks the personalia view/edit pages of the manager

check_healthperson_pages(*base_url*, *view_edit_pages*)
Check edit pages for different user types, try to post with default values that are retrieved with a 'get' request.

Args:

- base_url: the base_url to build up the url to call

- view_edit_pages: a list of parts of urls to use to get the view and post the edit page.

check_secretary_pages(*secretary*)
Check the pages accessible by a manager for a secretary

Args:

- secretary: the secretary to check the pages for

check_healthprofessional_pages(*healthprofessional*)
Check the pages accessible by a manager for a healthprofessional

Args:

- healthprofessional: the healthprofessional to check the pages for

check_patient_pages(*patient*)
Check the pages accessible by a manager for a patient

Args:

- patient: the patient to check the pages for

add_new_patient()
Try adding a new patient

add_new_healthprofessional()
Try adding a new healthprofessional

add_new_secretary()
Try adding a new secretary

remove_healthprofessional(*healthprofessional*)
Remove an healthprofessional

remove_patient(*patient*)
Remove a patient

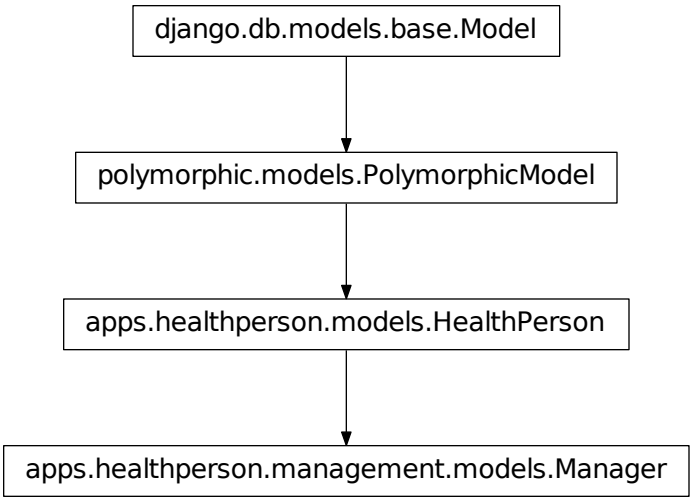
remove_secretary(*secretary*)
Remove a secretary

test_management()
Manager checks runner, performs all check definitions.

models.py

This module contains the manager model definition. The manager is coupled to a *apps.account.models.User* instance via the *apps.healthperson.models.HealthPerson* baseclass.

Inheritance-diagram:



Class definitions:

class apps.healthperson.management.models.**Manager**(**args*, ***kwargs*)
Bases: *apps.healthperson.models.HealthPerson*

Stores the manager with no extra information.

Parameters

- **id** (*AutoField*) –
- **polymorphic_ctype_id** (ForeignKey to ContentType) –
- **added_on** (DateField) –
- **added_by_id** (ForeignKey to *HealthPerson*) –
- **healthperson_ptr_id** (OneToOneField to *HealthPerson*) –

forms.py

This module only contains a form for changing the profile information of the manager itselves.

Class definitions:

class apps.healthperson.management.forms.**ProfileEditForm**(**args*, ***kwargs*)
Bases: *apps.account.forms.BasePasswordProfileEditForm*

Edit manager profile (personalia) form

views.py

This module contains all views used by a manager.

Class definitions:

class apps.healthperson.management.views.**ManagerBaseView**(***kwargs*)
Bases: django.views.generic.base.View

Base class for manager views, automatically add manager to view

set_manager()
Add the manager to the view

class apps.healthperson.management.views.**ManagerIndexView**(***kwargs*)
Bases: *apps.healthperson.management.views.ManagerBaseView*, apps.base.views.BaseIndexTemplateView

Manager homepage view

class apps.healthperson.management.views.**ManagerPersonaliaView**(***kwargs*)
Bases: *apps.healthperson.management.views.ManagerBaseView*, django.views.generic.base.TemplateView

Manager personalia view

class apps.healthperson.management.views.**SearchView**(***kwargs*)
Bases: *apps.healthperson.management.views.ManagerBaseView*, django.views.generic.base.TemplateView

Manager search view for in the homepage

class apps.healthperson.management.views.**ManagerPersonaliaEdit**(***kwargs*)
Bases: *apps.healthperson.management.views.ManagerBaseView*, *core.views.FormView*

Manager personalia edit view

form_class
alias of ProfileEditForm

patient

Remote Care allows chronic patients to digitalize their periodic checkups/controls by filling in questionnaires and providing optional extra information.

Patients are automatically informed when they need to fill in questionnaires. The results of previous filled in questionnaires can be viewed and the patient can read messages sent by an healthprofessional.

When the patient is feeling ill (but no emergency) the 'urgent control' questionnaires can be filled in advance of planning an appointment with the healthprofessional.

The information section shows disease specific information to the patient.

Modules:

tests.py

This module defines the tests done patient functionality that is not part of other apps

Class definitions:

class apps.healthperson.patient.tests.**PatientTest**(*methodName='runTest'*)
Bases: *core.unittest.baseunittest.BaseUnitTest*

Unittest definitions for patients

check_account (*base_url*)
Check viewing/editing the account (profile) information for a patient

check_notification (*base_url*)
Check viewing/editing the notification settings for a patient

check_messages (*base_url*)
Check viewing messages

check_search ()
Check the search function in the homepage

check_questionnaire_helper (*url*, *count*, *test_value*)
Helper function for testing the filled-in questionnaire details

check_questionnaire_details (*base_url*)
Check the filled-in questionnaire detail pages

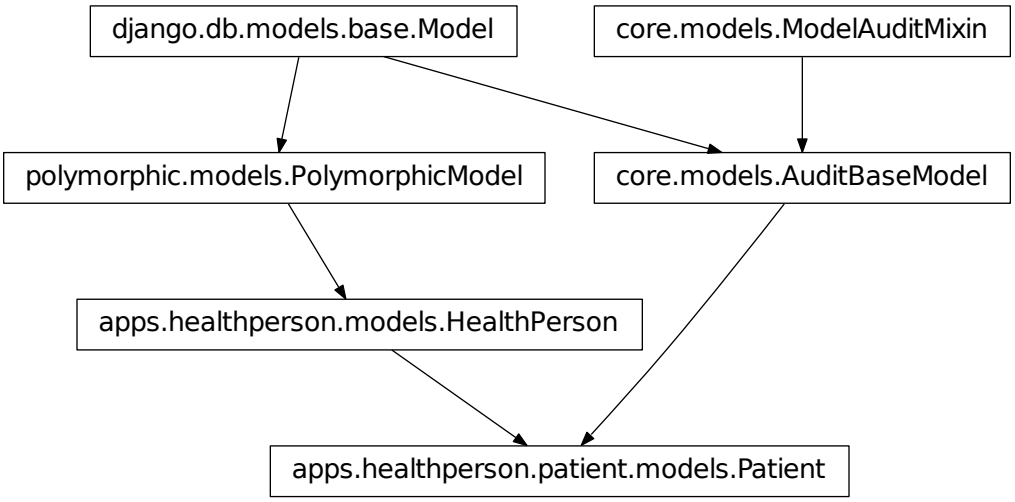
check_model ()
Check functions in model.py not covered by other tests but very important for correctly adding new questionnaires etc.

test_patient_functions ()
Run all checks for a patient

models.py

This module contains the patient model definition. The patient is coupled to a *apps.account.models.User* instance via the *apps.healthperson.models.HealthPerson* baseclass.

Inheritance-diagram:



Class and function definitions:

apps.healthperson.patient.models.**add_weken**(*number*)
Adds 'week' or 'weken' to the provided number

Args:

- number: the number of weeks

Returns: the number with 'week' or 'weken' appended

class apps.healthperson.patient.models.**Patient**(**args*, ***kwargs*)
Bases: *apps.healthperson.models.HealthPerson*, *core.models.AuditBaseModel*

Stores patient specific information.

Parameters

- **id** (*AutoField*) –
- **polymorphic_ctype_id** (ForeignKey to ContentType) –
- **added_on** (DateField) –
- **added_by_id** (ForeignKey to *HealthPerson*) –
- **healthperson_ptr_id** (OneToOneField to *HealthPerson*) –
- **rc_registration_number** (*CharField*) –
- **diagnose** (*CharField*) – choices=[rheumatoid_arthritis, chron, colitis_ulcerosa, intestinal_transplantation]
- **excluded_questionnaires** (*TextField*) –
- **current_practitioner_id** (ForeignKey to *HealthProfessional*) –
- **regular_control_frequency** (*ChoiceOtherField*) – choices=[never, 3_months, 6_months, 12_months, other]
- **blood_sample_frequency** (*ChoiceOtherField*) – choices=[sameasregular, 3_months, 6_months, 12_months, other]
- **last_blood_sample** (DateField) –
- **always_clinic_visit** (*CharField*) – choices=[alwaysclinicvisit, onlyifwanted]
- **regular_control_start_notification** (*CharField*) – choices=[sms_and_email, sms_only, email_only]
- **regular_control_reminder_notification** (*CharField*) – choices=[sms_and_email, sms_only, email_only]
- **healthprofessional_handling_notification** (*CharField*) – choices=[sms_and_email, sms_only, email_only]
- **message_notification** (*CharField*) – choices=[sms_and_email, sms_only, email_only]

last_questionnaire_date

Returns: The last questionnaire date or None

timedelta_since_last_questionnaire

Returns: The timedelta since the last questionnaire date or None

days_since_last_questionnaire

Returns: The days since the last questionnaire date or None

next_questionnaire_ready

Returns: True if the next questionnaire is ready, which means it should be filled in. False otherwise.

next_questionnaire_date

Returns: The next questionnaire date based on the last_questionnaire_date and the control_frequency setting or None

always_appointment

Returns: True if there should always follow an appointment after the periodic control

display_regular_control_frequency

Returns: The regular control frequency in readable format.

display_blood_sample_frequency

Returns: The blood sample frequency in readable format.

include_blood_taken_questions

Returns: True if the blood taken questions should be included, which is based on the blood sample frequency and the last last_blood_taken_date. False otherwise.

blood_taken_freq_display

Returns: Returns the blood taken frequency in readable format

last_blood_taken_date

Returns: The last date when blood was taken or None

forms.py

This module only contains the forms for a patient for searching filled-in information, editing notifications settings, and editing personalia. For managers/secretary/healthprofessionals forms are included for administration of patients.

Class definitions:

class apps.healthperson.patient.forms.**PatientSearchForm**(**args*, ***kwargs*)
Bases: *core.forms.BaseForm*

Search for a patient. Used by all healthpersons except for patients.

class apps.healthperson.patient.forms.**PatientDiagnoseControleEditForm**(**args*, ***kwargs*)
Bases: *core.forms.BaseModelForm*

Edit the diagnose and controle settings. Used by a healthprofessional or manager.

class apps.healthperson.patient.forms.**PatientNotificationEditForm**(**args*, ***kwargs*)
Bases: *core.forms.BaseModelForm*

Edit the notification settings. Used by a patient.

class apps.healthperson.patient.forms.**PatientProfileEditForm**(**args*, ***kwargs*)
Bases: *apps.account.forms.BasePasswordProfileEditForm*

Edit patient profile form used by the patient self

class apps.healthperson.patient.forms.**PatientPersonaliaEditForm**(**args*, ***kwargs*)
Bases: *apps.account.forms.BasePasswordProfileEditForm*

Edit patient profile form used by an healthprofessional and secretary

class apps.healthperson.patient.forms.**PatientPersonaliaEditFormManager**(**args*, ***kwargs*)

Bases: *apps.account.forms.BasePasswordProfileEditForm*

Edit patient profile form used by an manager.

class apps.healthperson.patient.forms.**PatientAddForm**(**args*, ***kwargs*)

Bases: *apps.account.forms.BaseProfileEditForm*

Add new patient form

views.py

This module contains all views used by the patient self or by other healhtpersons that administrate patients.

Class definitions:

apps.healthperson.patient.views.**get_patient**(*request*, *patient_session_id*)

Helper function for getting the patient from a patient_session_id

Args:

- request: the request instance
- patient_session_id: the patient_session_id coupled to the patient

Returns: the patient instance

class apps.healthperson.patient.views.**PatientBaseView**(***kwargs*)

Bases: django.views.generic.base.View

Base view which adds the patient by using the patient_session_id or logged in user

dispatch(**args*, ***kwargs*)

Add patient to the view class

get_context_data(***kwargs*)

Base context, include the patient by default

class apps.healthperson.patient.views.**PatientIndexView**(***kwargs*)

Bases: apps.base.views.BaseIndexTemplateView

View which shows the homepage of the patient

get_all_messages_for_patient_index(*patient*)

Get all messages for a patient

Args:

- patient: the patient for with the messages should be returned

Returns: A list with all messages for a patient

class apps.healthperson.patient.views.**PatientGenericView**(***kwargs*)

Bases: *apps.healthperson.patient.views.PatientBaseView*, django.views.generic.base.TemplateView

Generic view based on the patient baseview and TemplateView

class apps.healthperson.patient.views.**PatientNotificationView**(***kwargs*)

Bases: *apps.healthperson.patient.views.PatientGenericView*

Show the notification settings used by a different roles

class apps.healthperson.patient.views.**PatientNotificationEditView**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, *core.views.FormView*

Edit the notification settings used by a patient

form_class
alias of PatientNotificationEditForm

class apps.healthperson.patient.views.**PatientProfileView**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientGenericView*

Shows the profile information of a patient used by a multiple roles

class apps.healthperson.patient.views.**PatientProfileEditView**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, *core.views.FormView*

Edit the profile information of a patient used by a patient

form_class
alias of PatientProfileEditForm

class apps.healthperson.patient.views.**PatientAppointmentsView**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, django.views.generic.base.TemplateView

Show the appointments information of a patient used by a healthprofessional

class apps.healthperson.patient.views.**PatientPersonaliaView**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, django.views.generic.base.TemplateView

Show the profile information of a patient used by a patient admins

class apps.healthperson.patient.views.**PatientMessagesView**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, django.views.generic.base.TemplateView

Show the messages sent to a patient used by healthprofessionals

class apps.healthperson.patient.views.**PatientReportsView**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, django.views.generic.base.TemplateView

Show the report for a patient used by healthprofessionals

class apps.healthperson.patient.views.**PatientControlesView**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, django.views.generic.base.TemplateView

Show the finished controls for a patient used by healthprofessionals

class apps.healthperson.patient.views.**PatientFreqControlView**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, django.views.generic.base.TemplateView

Show the control and blood taken frequencies for a patient used by patient admins

class apps.healthperson.patient.views.**PatientPersonaliaEditView**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, *core.views.FormView*

Edit the personalia of a patient used by patient admins.

..Note:: Managers use a different form for editing, including options for resetting the password.

form_class
alias of PatientPersonaliaEditForm

class apps.healthperson.patient.views.**QuestionnaireForDiagnose**(***kwargs*)
Bases: django.views.generic.base.View

Returns a JSON list of questionnaires available for a diagnose

get_questionnaires_for_diagnose_helper(*selected_questionnaire*)
Helper function for getting the questionnaires

Returns: The list of questionnaires for the selected_questionnaire

class apps.healthperson.patient.views.**PatientFreqControlEditView**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, *core.views.FormView*

Edit the patient control and blood take frequency used by patient admins

form_class
alias of PatientDiagnoseControleEditForm

class apps.healthperson.patient.views.**PatientSearchView**(***kwargs*)
Bases: *core.views.FormView*

Search for patients view used by patient admins

form_class
alias of PatientSearchForm

get_initial()
Get initial data, used for showing the old results and form data when the user uses the back button to return to the form

get(*request*, **args*, ***kwargs*)
Re-execute the search if the user has used the back button

form_valid(*form*)
Perform the search action, search for a healthprofessional

class apps.healthperson.patient.views.**SearchView**(***kwargs*)
Bases: django.views.generic.base.TemplateView

Search view for in the homepage of the patient allows searching for answers on questionnaires and messages

get_inner_context(*request*, *patient*, *model_display_name*)
Gets the questionnaires to be included into the results.

Args:

- request: the current request
- patient: the patient who is searching
- model_display_name: the model_display_name of the questionnaire to search for.

Returns: a list of questionnaires

post(*request*, **args*, ***kwargs*)
Execute the search for questionnaires and messages based on 'searchterm'

class apps.healthperson.patient.views.**HealthPatientAddView**(***kwargs*)
Bases: *apps.healthperson.views.BaseAddView*

Class based view for adding a new patient used by patient admins

form_class
alias of PatientAddForm

dispatch(**args*, ***kwargs*)
Check permissions

class apps.healthperson.patient.views.**PatientRemoveView**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, django.views.generic.base.TemplateView

Class based view for removing a patient used by patient admins

post(*request*, **args*, ***kwargs*)
Remove the patient by setting the user to inactive

secretariat

The secretary is responsible for making appointments with patients when a patient has finished a control and requested an appointment or an healthprofessional requested an appointment.

Modules:

tests.py

Definitions for tests for secretary. Currently only includes tests for checking the search page

Class definitions:

class apps.healthperson.secretariat.tests.**SecretaryTest**(*methodName='runTest'*)
Bases: *core.unittest.baseunittest.BaseUnitTest*

Test the search function on the homepage

check_search()
Check homepage search for patients

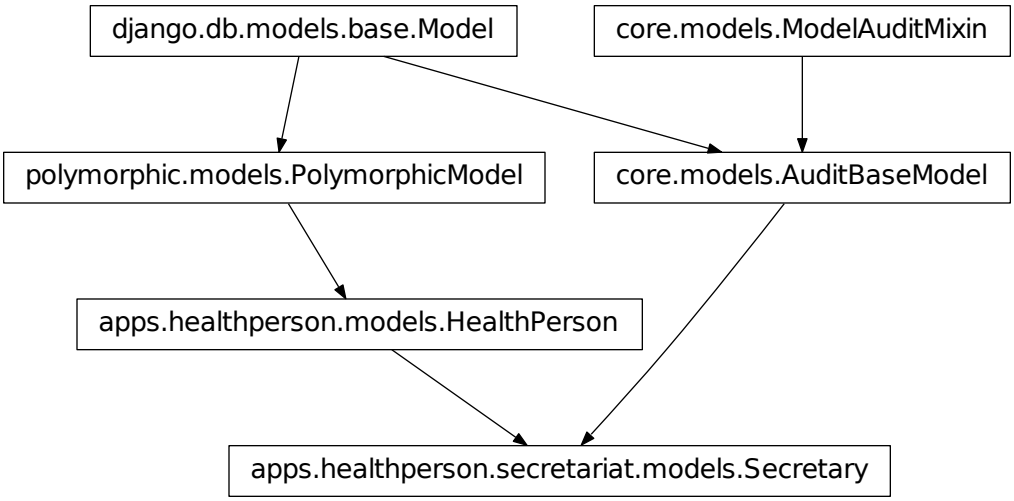
test_secretary_functions()
Run the checks for a secretary

> **Note:** testing adding/editing appointments is included into the appointments app

models.py

This module contains the secretary model definition. The secretary is coupled to a *apps.account.models.User* instance via the *apps.healthperson.models.HealthPerson* baseclass.

Inheritance-diagram:



Class definitions:

class apps.healthperson.secretariat.models.**Secretary**(**args*, ***kwargs*)
Bases: *apps.healthperson.models.HealthPerson*, *core.models.AuditBaseModel*

Stores extra information for a secretary

Parameters

- **id** (*AutoField*) –
- **polymorphic_ctype_id** (ForeignKey to ContentType) –
- **added_on** (DateField) –
- **added_by_id** (ForeignKey to *HealthPerson*) –
- **healthperson_ptr_id** (OneToOneField to *HealthPerson*) –
- **specialism** (*CharField*) – choices=[gastro_liver_disease, rheumatology, surgery, internal_medicine, orhopedie]

forms.py

This module contains the forms for profile editing by a secretary and als for the manager to search for a secretary.

Class definitions:

class apps.healthperson.secretariat.forms.**SecretarySearchForm**(**args*, ***kwargs*)
Bases: *core.forms.BaseForm*

Search for a secretary by last_name or specialism, used by a manager.

class apps.healthperson.secretariat.forms.**SecretaryEditForm**(**args*, ***kwargs*)
Bases: *apps.account.forms.BasePasswordProfileEditForm*

Edit secretary profile/personalia form

class apps.healthperson.secretariat.forms.**SecretaryAddForm**(**args*, ***kwargs*)
Bases: *apps.account.forms.BaseProfileEditForm*

Add new secretary form

views.py

A secretary can add/edit/remove patients and make appointments. This module contains the views needed to perform those functions.

Class definitions:

class apps.healthperson.secretariat.views.**SecretaryBaseView**(***kwargs*)
Bases: django.views.generic.base.View

Base view which adds the secretary by using the secretary_session_id or logged in user

dispatch(**args*, ***kwargs*)
Add secretary to the view class

get_context_data(***kwargs*)
Base context, include the secretary by default

class apps.healthperson.secretariat.views.**SecretaryIndexView**(***kwargs*)
Bases: apps.base.views.BaseIndexTemplateView, *apps.healthperson.secretariat.views.SecretaryBaseView*

Class based view for secretary homepage

class apps.healthperson.secretariat.views.**SecretariatSearchView**(***kwargs*)
Bases: *core.views.FormView*

Search secretary view page used by manager

form_class
alias of SecretarySearchForm

get_initial()
Get initial data, used for showing the old results and form data when the user uses the back button to return to the form

get(*request*, **args*, ***kwargs*)
Re-execute the search if the user has used the back button

form_valid(*form*)
Perform the search action, search for a secretary

class apps.healthperson.secretariat.views.**SearchView**(***kwargs*)
Bases: django.views.generic.base.TemplateView

Generic search page as available in the homepage can be used by a secretary or manager (second is not used)

post(*request*, **args*, ***kwargs*)
Search for patients

class apps.healthperson.secretariat.views.**SecretariatAddView**(***kwargs*)
Bases: *apps.healthperson.views.BaseAddView*

Class based view for adding a new secretary used by a manager

form_class
alias of SecretaryAddForm

class apps.healthperson.secretariat.views.**SecretariatPersonaliaView**(***kwargs*)
Bases: *apps.healthperson.secretariat.views.SecretaryBaseView*, django.views.generic.base.TemplateView

Class based view for showing secretary personalia used by the manager and secretary

class apps.healthperson.secretariat.views.**SecretariatPersonaliaEdit**(***kwargs*)
Bases: *apps.healthperson.secretariat.views.SecretaryBaseView*, *core.views.FormView*

Class based view for editing secretary personalia used by a manager and secretary

form_class
alias of SecretaryEditForm

form_valid(*form*)
Save the person and secretary information

class apps.healthperson.secretariat.views.**SecretariatRemove**(***kwargs*)
Bases: *apps.healthperson.secretariat.views.SecretaryBaseView*, django.views.generic.base.TemplateView

Remove a secretary, sets the user deleted_on date and is_active to False

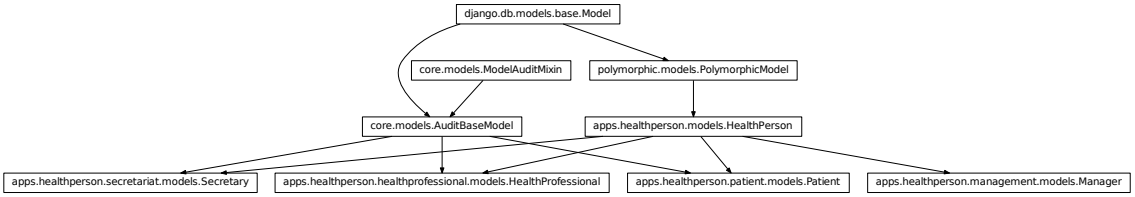
post(*request*, **args*, ***kwargs*)
Remove the secretary by setting the user to inactive

Modules:

models.py

This module includes the definition for the HealthPerson model.

The polymorphic model *HealthPerson* is used as a baseclass for thedifferent models coupled to Remote Care roles:



The *HealthPerson* is set as a "ForeignKey" on the *apps.account.models.User* user model. This allows every user to have a different model corresponding to their role.

Class definitions:

class apps.healthperson.models.**HealthPerson**(**args*, ***kwargs*)
Bases: polymorphic.models.PolymorphicModel

The healthperson is the polymorphic model baseclass for the different user models in Remote Care:

- Healthprofessional
- manager
- patient

- secretary

The *HealthPerson* is set as a "ForeignKey" on the *apps.account.models.User* user model. This allows every user to have a different model corresponding to their role.

Parameters

- **id** (*AutoField*) –
- **polymorphic_ctype_id** (ForeignKey to ContentType) –
- **added_on** (DateField) –
- **added_by_id** (ForeignKey to *HealthPerson*) –

health_person_id

The health_person_id (=self.id) is not directly used in url's of healthperson's but is stored in the session with a random key instead. This random key is included in the url's and used in the views to get the id of the healthperson. This way the id's are completely invisible for users and makes it impossible to easy switch to data of another user.

In an older version the HealhPerson had a seperate health_person_id field, this has been replaced with the 'id' field. In order to not have to change all the templates it has been added as a property.

Returns: self.id

views.py

Since all users use the same *apps.account.models.User* model, this module provides a baseclass for adding new users.

Function definitions:

class apps.healthperson.views.**BaseAddView**(***kwargs*)

Bases: *core.views.FormView*

Baseclass for adding user, used for adding users of all 3 different role types. (managers cannot be added)

get_user_for_form(*form*)

Args:

- form: The modelform with model=User

Returns: Saved user with some default values set.

url_prefix

Returns: The url prefix to be included in the e-mail

utils.py

This util module provides help functions for checking if an user is in the appropiate group.

Function definitions:

apps.healthperson.utils.**is_allowed**(*user*, *white_list=[]*)

Checks if the user's group is in provided list with white listed groups.

Args:

- user: the user instance to check
- while_list: a list with group names the user should be in.

Returns: True is the user is in provided group white_list

apps.healthperson.utils.**is_allowed_patient**(*user*)
Shortcut for checking if the user is a patient

apps.healthperson.utils.**is_allowed_secretary**(*user*)
Shortcut for checking if the user is a secretary

apps.healthperson.utils.**is_allowed_manager**(*user*)
Shortcut for checking if the user is a manager

apps.healthperson.utils.**is_allowed_manager_and_secretary**(*user*)
Shortcut for checking if the user is a manager or secretary

apps.healthperson.utils.**is_allowed_manager_and_healthprofessional**(*user*)
Shortcut for checking if the user is a manager or healthprofessional

apps.healthperson.utils.**is_allowed_healthprofessional**(*user*)
Shortcut for checking if the user is an healthprofessional

apps.healthperson.utils.**is_allowed_patient_admins**(*user*)
Shortcut for checking if the user is a manager, secretary or healthprofessional

service

The service package contains functions which should be run daily via a deamon around 9:00. (since sms and e-mail notifications are sent to Remote Care users)

The service functions automatically sent e-mail and/or sms notifications for due controls to patients or due handling of filled in controls to healthprofessionals. Patient that are set for deleting are automatically deleted after two weeks.

Modules:

tests.py

Unittests for the service functions

Class definitions:

class apps.service.tests.**ServiceTest**(*methodName='runTest'*)
Bases: *core.unittest.baseunittest.BaseUnitTest*

Test the service functions

reminder_checks(*healthperson*, *body_test*, *check_email*, *check_sms*)
Checks if reminders have been sent via the SMS and e-mail stores that catch SMS and e-mail traffic during testing

Args:

- healthperson: healthperson to pick the e-mail address to test from
- body_test: the body of the message to test for
- check_email: do need to check e-mail?
- check_sms: do need to check sms?

do_reminder_check(*body_test*, *function*, *attr_name*, *function_instance*, *attr_instance*, *check_instance*)
Runner for checking reminders, highly abstracted

Args:

- body_test: the body to test for
- function: the function to call with function_instance
- attr_name: the attr_name to set on attr_instance
- function_instance: the param to sent to the function
- check_instance: the instance to check, should be a healthperson

See example in 'do_test_questionnaire_fillin_sms' to see what is sent. for the specific args

do_questionnaire_check(*body_test*, *function*, *attr_name*)
Helper function for checking questionnaire reminder functions

Args:

- body_test: the body to test for
- function: the function to call before checking
- attr_name: the attr_name to set before checking

do_report_check(*body_test*, *function*, *attr_name*)
Helper function for checking report reminder functions

Args:

- body_test: the body to test for
- function: the function to call before checking
- attr_name: the attr_name to set before checking

do_test_questionnaire_fillin_sms()
Check if the questionnaire fillin sms/email function works correctly

do_test_questionnaire_reminder_sms()
Check if the questionnaire reminder sms/email function works correctly

do_test_urgent_report_reminder()
Check if the urgent report reminder sms/email function works correctly

do_test_report_reminder()
Check if the report reminder sms/email function works correctly

do_test_remove_deleted_patients()
Check if patients are automatically deleted

do_test_check_questionnaire_fillin_deadlines()
Check if the questionnaire fillin deadlines function works

create_questionnaire_request(*patient*, *date_time*, *is_handled*)
Creates a questionnaire request

Args:

- patient: the patient to couple to the questionnaire request
- date_time: the date & time to set as finished date
- is_handled: set the handled_on to date_time if True

do_test_insert_new_questionnaire_requests()
Checks if insert new questionnaire request functions works correctly by testing a range of different scenario's

unhandeld_questionnaires_helper(*urgent*)
Helper functions for testing unhandeld questionnaires

Args:

- urgent: check urgent or not urgent

do_test_unhandeld_questionnaires()
Check if reminder are sent to an healhprofessional for unhandeld filled-in questionnaires

test_service_functions()
Test runner which performs all checks

utils.py

Module providing reminder functions.

Note: The main function on the bottom should be run daily via a deamon or other solution around 9:00.

Function definitions:

apps.service.utils.**send_questionnaire_reminder_sms**(*patient*)
Send a sms reminder to a patient who has not filled in a questionnaire

apps.service.utils.**send_questionnaire_fillin_sms**(*patient*)
Send a sms and/or email message to a patient that a new series of questionnaires that should be filled in

apps.service.utils.**send_urgent_report_reminder**(*urgent_questionnaire_request*)
Send a message to healthprofessional about an urgent control which he/she has not created a report for.

apps.service.utils.**send_report_reminder**(*questionnaire_request*)
Send a message to healthprofessional about an controle which he/she has not created a report for.

apps.service.utils.**remove_deleted_patients**()
Automatically remove patients that are set for deletion after 2 weeks

apps.service.utils.**check_questionnaire_fillin_deadlines**()
Check if deadlines for filling in questionnaires are passed

apps.service.utils.**insert_new_questionnaire_request_for_patient**(*patient*)
Add a new questionnaire request for a patient for periodic checking a patient.

apps.service.utils.**insert_new_questionnaire_requests**()
Helper function for adding a new questionnaire request

apps.service.utils.**check_unhandled_questionnaires**()
Check if there are unhandeld controls by healthprofessionals

apps.service.utils.**check_unhandled_urgent_questionnaires**()
Check if there are unhandeld urgent controls by healthprofessionals

apps.service.utils.**main_run_daily**()
Function which can be called daily to perform all necessary checks.

Note: Should be run around 09:00. !!NOT AT MIDNIGHT!! since people are going to get SMS notifications.

utils

Package that provides helper functions for Remote Care for:

- Sending email and sms messages (utils)
- Exporting to PDF (pdf)
- Exporting to DocX (docxhelper)

Modules:

tests.py

This module contains tests for the export functionality and the functions in utils that are not covered by other tests in Remote Care.

Class definitions:

class apps.utils.tests.**Exports**(*methodName='runTest'*)

Bases: django.test.testcases.TestCase

Test class for testing docX & PDF export

get_request()

Generate a fake request

Returns: A fake request instance

get_source()

Get the HTML source to test

Returns: The HTML source to test with

test_docx()

Test the DocX export functionality

test_pdf()

Test the PDF export functionality

pdf.py

This module contains the functions for converting HTML to PDF.

Function definitions:

apps.utils.pdf.**convertHtmlToPdf**(*sourceHtml*)

Converts the HTML to PDF

Args:

- sourceHtml: the HTML (string) to convert

Returns: An HttpResponse instance with the PDF included or an HttpResponse instance with the HTML included.

apps.utils.pdf.**render_to_PDF**(*request*, *body*)

Shortcut for rendering the HTML template and create PDF

Args:

- request: the initial request

- body: the body to include in the PDF in HTML format

Returns: An HttpResponse instance with the PDF included or an HttpResponse instance with the HTML included.

docxhelper.py

This module contains classes and functions for generating DocX documents from HTML used for exporting reports.

Function definitions:

class apps.utils.docxhelper.**HTMLToDocXParser**
Bases: HTMLParser.HTMLParser

Class for parsing HTML to Docx

set_body(*body*)
Initalize the DocX document by setting the body

Args:

- body: the DocX body

handle_starttag(*tag*, *attrs*)
Handles a starttag in the HTML. Converts a predefined list of tags to the corresponding DocX definitions.

Args:

- tag: the name of the HTML tag
- attr: optional attrs coupled to the HTML tag

handle_endtag(*tag*)
Handles a endtag in the HTML. Converts a predefined list of tags to the corresponding DocX definitions:

Args:

- tag: the name of the HTML tag

handle_data(*data*)
Adds text to the DocX document

Args:

- data: the data/text to add to the document

handle_entityref(*name*)
Handle special HTML entities

Args:

- name: the name of the HTML entity

apps.utils.docxhelper.**convertHtmlToDocX**(*sourceHtml*)
Wrapper function for converting HTML to DocX

Args:

- sourceHtml: the HTML to convert

Returns: A response instance with the DocX document included

apps.utils.docxhelper.**render_to_DocX**(*request*, *body*)
Shortcut function for rendering the HTML template and create DocX

Args:

- request: the initial request
- body: the body with the text to be exported

Returns: A response instance with the DocX document included

utils.py

This module contains functions for sending email and SMS messages to Remote Care users.

Function definitions:

apps.utils.utils.**send_email_to**(*email_adres*, *email_content*, *subject='Remote Care - notificatie'*)
Generic function for sending email to email_adres with email_content

Args:

- email_adres: the receiver of the email
- email_content: the body of the email

apps.utils.utils.**send_authorisation_sms_to**(*mobile_number*, *auth_code*)
Wrapper function for later usage

Args:

- mobile_number: the mobile number to sent auth_code to
- auth_code: the SMS message to sent

apps.utils.utils.**send_sms_to_patient**(*patient*, *content*)
Generic function for sending an SMS message to a patient Automatic strips the content so it fits within 160 characters.

Args:

- patient: the patient to sent the SMS to
- content: The content of the SMS message

apps.utils.utils.**send_sms_to**(*mobile_number*, *auth_code*)
Generic function which sents the sms using Mollie services.

Args:

- mobile_number: the mobile_number to sent the SMS message to
- auth_code: The content of the SMS message

apps.utils.utils.**generic_sent_notification_to_patient**(*patient*, *sms_template*, *email_template*, *notification_setting*)
Generic function which is used for sending regular control reminders to a patient

Args:

- patient: the patient to sent the message(s) to
- sms_template: the sms message to sent to a patient
- email_template: the email message to sent to a patient

apps.utils.utils.**send_notification_of_new_report**(*patient*)
Send a notification of a new report to a patient

Args:

- patient: the patient to sent the message(s) to

apps.utils.utils.**send_notification_of_new_message**(*patient*)
Send a notification of a new message to a patient

Args:

- patient: the patient to sent the message(s) to

apps.utils.utils.**get_password_change_request**(*email_key*)
Try to get the current password request and automatically remove expired ones.

Args:

- email_key: The original email_key stored in HMAC format in the passwordchangerequest.

Returns: The associated PasswordChangeRequest instance or None

apps.utils.utils.**get_user_for_password_change_request**(*email_key*)
Retrieves the user coupled to a password change_request

Args:

- email_key: The original email_key stored in HMAC format in the passwordchangerequest.

Returns: The User for the associated PasswordChangeRequest instance or None

apps.utils.utils.**change_password**(*email_key*, *password*, *sms_code*)
Change password by checking email_key, password and sms_code

Args:

- email_key: The original email_key stored in HMAC format in the passwordchangerequest.
- password: the new password to set
- sms_code: the SMS authorization code to check

Returns: True if the password has been changed else False

apps.utils.utils.**sent_password_change_sms**(*email_key*, *email*, *date_of_birth*, *sent_sms=True*, *sms_code=None*)
Sent password change sms, last two parameters are for debugging which can be removed in production stage.

Args:

- email_key: The original email_key stored in HMAC format in the passwordchangerequest.
- date_of_birth: the date_of_birth to check
- sent_sms: need to sent SMS authorization code? (default=True)
- sms_code: override sms authorization code

Returns: True if the password change authorization SMS has been sent

apps.utils.utils.**sent_password_change_request**(*user*, *url_prefix*, *change_request_by_manager=False*, *new_account=False*)
Sent an e-mail with a password change request to an users e-mail address

Args:

- user: the user to sent the password change request email to.
- url_prefix: the url prefix to use in the email

- change_request_by_manager: is this change initialized by a manager?
- new_account: is this a new user?

apps.utils.utils.**create_login_sms_code**(*user*, *send_sms=True*, *sms_code=None*)
Create and sent a login SMS code to an user the sms_code is stored in HMAC format.

Args:

- user: the user to sent the login SMS authorization code to.
- send_sms: do sent a SMS authorization code? (default=True)
- sms_code: override the random sms_code

apps.utils.utils.**check_login_sms_code**(*sms_code*)
Check if the sms_code is correct, returns the user coupled to this sms_code which can be further validated.

Args:

- sms_code: the SMS authorization code sent during login, stored in HMAC form in a LoginSMSCode instance.

Returns: The User instance for the login SMS authorization code or None

lists

The list app currently only contains the Hospital model which stores a list of hospitals but can contain other lists in the future.

Modules:

models.py

This module contains the Hospital model definition. All users are hospital 'bound' meaning that an healhtprofessional can only search patients within his/hers hospital.

Class definitions:

class apps.lists.models.**Hospital**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Stores a list of hospitals

Parameters

- **id** (*AutoField*) –
- **abbreviation** (*CharField*) –
- **full_name** (*CharField*) –
- **city** (*CharField*) –
- **created** (*DateTimeField*) –
- **modified** (*DateTimeField*) –
- **tags** (*TextField*) –

classmethod json_add_url()
Function which returns a json_url for this object

classmethod search_keys()
Function which returns the search keys for this object

classmethod icon_url()
Function which returns the icon to be used for this object

rcmessages

Healthprofessionals and secretaries can sent messages patients. Messages are also sent after the healthprofessional finished handling a (urgent) control or a secretary planned an appointment. All messages are stored encrypted with the encryption_key of the patient.

Modules:

tests.py

Module containing message tests.

Class definitions:

class apps.rcmessages.tests.**RCMessageTest**(*methodName='runTest'*)
Bases: *core.unittest.baseunittest.BaseUnitTest*

Test adding and viewing messages.

test_messages()
Test adding messages with different senders and test if encryption works correctly.

models.py

This module contains the RCMessage model definition.

Class definitions:

class apps.rcmessages.models.**RCMessage**(**args*, ***kwargs*)
Bases: *core.models.AuditBaseModel*

Stores messages which can be sent from healthprofessional/secretary to patient with AES encryption

Note: It's named RCMessage instead of just Message to avoid confusion with the Django build in Message

Parameters

- **id** (*AutoField*) –
- **patient_id** (ForeignKey to *Patient*) –
- **healthprofessional_id** (ForeignKey to *HealthProfessional*) –
- **secretary_id** (ForeignKey to *Secretary*) –
- **related_to_id** (ForeignKey to *QuestionnaireRequest*) –
- **subject** (*EncryptedCharField*) –
- **internal_message** (*EncryptedTextField*) –
- **added_on** (DateField) –

- **read_on** (DateField) –

message()
Shortcut to the internal_message field

Returns: self.internal_message

sender
Shortcut to get the sender of the message

Returns: self.healthprofessional or, if not set, self.secretary

audit_encryption_key_id
Get the EncryptionKey id so it can be coupled to the log item in the audit.

Returns: The id of the EncryptionKey that is used to encrypt the model instance.

forms.py

This module contains the forms for messages.

Class definitions:

class apps.rcmessages.forms.**MessageAddForm**(**args*, ***kwargs*)
Bases: *core.forms.BaseModelForm*

Form for adding a new message

class apps.rcmessages.forms.**MessageSearchForm**(**args*, ***kwargs*)
Bases: *core.forms.BaseForm*

Healthprofessional can search his/hers sent messages based on the BSN or/and last_name of the patient

views.py

This module contains the class based views and functions for messages.

Class and function definitions:

apps.rcmessages.views.**remove_not_handled_messages**(*rc_messages*)
Function which removes messages that are coupled to a questionnaire which is not finished by a healthprofessional

Args:

- rc_messages: list of RCMessage instances to strip

Returns: list of RCMessage instances

apps.rcmessages.views.**get_all_messages_for_secretary**(*secretary*)

Args:

- secretary: the secretary to get all messages for

Returns: all messages for a secretary

apps.rcmessages.views.**get_all_messages_for_healthprofessional**(*healthprofessional*)

Args:

- healthprofessional: the healthprofessional to get all messages for

Returns: all messages for a healthprofessional

apps.rcmessages.views.**get_all_messages_for_patient**(*patient*)

Args:

- patient: the patient to get all messages for

Returns: all messages for a patient

class apps.rcmessages.views.**MessageAdd**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, *core.views.FormView*

Class based view for adding a new message by a secretary or healthprofessional

form_class
alias of MessageAddForm

class apps.rcmessages.views.**SentMessageSearch**(***kwargs*)
Bases: *core.views.FormView*

Search through sent messages either as an healthprofessional or secretary

form_class
alias of MessageSearchForm

get_context_data(***kwargs*)
Return the found patients in a context

class apps.rcmessages.views.**MessageSent**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, django.views.generic.base.TemplateView

Shows a confirmation message that the message has been sent.

class apps.rcmessages.views.**SentMessageOverview**(***kwargs*)
Bases: django.views.generic.base.TemplateView

Generates an overview of all sent messages of either a healthprofessional or secretary when there are no messages, otherwise the sentmessagedetails view is used.

class apps.rcmessages.views.**SentMessageDetails**(***kwargs*)
Bases: django.views.generic.base.TemplateView

Shows the contents of a sent message

class apps.rcmessages.views.**MessageOverview**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, django.views.generic.base.TemplateView

Shows an overview of all messages of a patient

class apps.rcmessages.views.**MessageDetails**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*, django.views.generic.base.TemplateView

Shows the contents of a message to a patient

report

Automatic generated reports can be edited by an healthprofessional during handling filled in controls (questionnaires) by a patient. The report can be exported in docX and PDF format.

Modules:

tests.py

Test the complete procedure of handling a filled in control by a healthprofessional.

Class definitions:

class apps.report.tests.**ReportTest**(*methodName='runTest'*)
Bases: *core.unittest.baseunittest.BaseUnitTest*

Test the report functionality by adding a report & message for a controle and an urgent controle

report_for_controle(*questionnaire*, *urgent=False*)
Test runner which tries to create a report, message and finishes the handling of a control

Args:

- questionnaire: the questionnaire_request to use for testing
- urgent: test the urgent control or default control procedure.

test_report()
Test a report for a normal control and a urgent control

models.py

Module which defines the Report model.

Class definitions:

class apps.report.models.**Report**(**args*, ***kwargs*)
Bases: *core.models.AuditBaseModel*

Model for saving reports of filled in questionnaires. Standard templates are used to generate the report which can be edited by the healthprofessional.

The report is encrypted with use of the personal key of the created_by healthprofessional

Parameters

- **id** (*AutoField*) –
- **questionnaire_request_id** (ForeignKey to *QuestionnaireRequest*) –
- **created_by_id** (ForeignKey to *HealthProfessional*) –
- **created_on** (DateField) –
- **finished_on** (DateField) –
- **invalid** (*BooleanField*) –
- **sent_to_doctor** (*BooleanField*) –
- **patient_needs_appointment** (*BooleanField*) –
- **report** (*EncryptedTextField*) –

filled_in

Returns: True if finished_on is set

audit_encryption_key_id
Get the EncryptionKey id so it can be coupled to the log item in the audit.

Returns: The id of the EncryptionKey that is used to encrypt the model instance.

forms.py

Module providing the forms for adding a report & message during handling a filled-in control by an healthprofessional.

Class definitions:

class apps.report.forms.**MessageAddEditForm**(**args*, ***kwargs*)
Bases: *core.forms.BaseModelForm*

Add/Edit a message for a patient (as part of the report function)

class apps.report.forms.**UrgentReportAddEditForm**(**args*, ***kwargs*)
Bases: *core.forms.BaseModelForm*

Add edit an urgent report, checks that items that are included in the template as placeholder are removed.

class apps.report.forms.**ReportAddEditForm**(**args*, ***kwargs*)
Bases: *core.forms.BaseModelForm*

Add/Edit an normal report. Is merely the same as the urgent one but needs some extra initialization.

views.py

This module contains a class for generating report templates and views for adding/editing and exporting reports.

Class definitions:

class apps.report.views.**ReportTemplateGenerator**
Class for generating a template of a report to be edited by an healthprofessional resulting in the final report.

get_questionnaires(*questionnaire_request*)
Get the questionnaires for a questionnaire_request

Args:

- questionnaire_request: the questionnaire_request to get the questionnaires for.

Returns: A list of questionnaires for the given questionnaire_request

get_context(*patient*, *healthprofessional*, *questionnaire_request*)
Generate the default context used to render all report templates

Args:

- patient: the patient instance to include
- healthprofessional: the healthprofessional instance to include
- questionnaire_request: the questionnaire_request instance to include

Returns: the context dict

render_template(*request*, *patient*, *healthprofessional*, *questionnaire_request*, *template_name*)
Render the template based on the given arguments

Args:

- request: the current request
- patient: the patient coupled to the report
- healthprofessional: the healthprofessional of the patient
- questionnaire_request: the questionnaire_request coupled to this report
- template_name: the (relative) path & name of the template

Returns: The rendered template

class apps.report.views.**ReportBaseView**(***kwargs*)
Bases: *apps.healthperson.patient.views.PatientBaseView*

Base class based view for all report generating views. Conditionally adds report, message to the view if present.

class apps.report.views.**QuestionnaireView**(***kwargs*)
Bases: *apps.report.views.ReportBaseView*, django.views.generic.base.TemplateView

Class based view for showing all filled in data of a questionnaire. Uses the same templates as used for the patient questionnaire views.

class apps.report.views.**ReportView**(***kwargs*)
Bases: *apps.report.views.ReportBaseView*, django.views.generic.base.TemplateView

Shows the (decrypted) report

class apps.report.views.**ReportExportBase**(***kwargs*)
Bases: *apps.report.views.ReportBaseView*

Class based view which is the base for all export views (currently DocX and PDF)

get_body()

Returns: The body in HTML to export.

class apps.report.views.**ReportDocX**(***kwargs*)
Bases: *apps.report.views.ReportExportBase*

Returns the report in DocX format

class apps.report.views.**ReportPDF**(***kwargs*)
Bases: *apps.report.views.ReportExportBase*

Returns the report in PDF format

class apps.report.views.**PrintReport**(***kwargs*)
Bases: *apps.report.views.ReportExportBase*

Returns the report in HTML for easy printing (Not used on the moment)

class apps.report.views.**ReportEditbase**(***kwargs*)
Bases: *apps.report.views.ReportBaseView*, *core.views.FormView*

Class based view which is the base for the editing/adding reports views

class apps.report.views.**UrgentReportEdit**(***kwargs*)
Bases: *apps.report.views.ReportEditbase*

Edit reports for urgent controls

form_class
alias of UrgentReportAddEditForm

class apps.report.views.**ReportEdit**(***kwargs*)
Bases: *apps.report.views.ReportEditbase*

Edit reports for normal controls

form_class
alias of ReportAddEditForm

class apps.report.views.**MessageView**(***kwargs*)
Bases: *apps.report.views.ReportView*

Show the (decrypted) message

class apps.report.views.**MessageEdit**(***kwargs*)
Bases: *apps.report.views.ReportBaseView*, *core.views.FormView*

Edit an existing message

form_class
alias of MessageAddEditForm

get_form_kwargs()
Include the healthprofessional instance

class apps.report.views.**HandlingFinish**(***kwargs*)
Bases: *apps.report.views.ReportView*

When a report and message has been added the questionnaire handling can be 'finished' by the healthprofessional. During this step the message is sent to the patient and the report becomes read-only.

questionnaire

The periodic controls to be filled in by patients consists of one or more disease specific questionnaires. Currently the questionna

- How are you doing? (start questionnaire)
- Disease activity
- Quality of life
- Quality of health care (optional, included once per year)
- Appointment, blood taken (finish questionnaire)

Some diseases don't have questionnaires for the disease activity step. For the urgent questionnaires the first two steps are replaced with a 'direct appointment' and a 'description of the problem' step.

Every questionnaire is stored as one model and multiple forms coupled to the questionnaire model are included into the *apps.questionnaire.wizards.QuestionnaireWizard* to display the forms after eachother.

The wizard uses a *apps.questionnaire.storage.DatabaseStorage* instance to store the filled in information. All posted form data is saved automatically to allow the user to stop and later return to finish the questionnaire. The storage allows saving unclean (post) data for this purpose. When all steps/questionnaires are succesfully filled in all unclean data is run through the clean methods of the forms and the cleaned data is stored in the questionnaire models.

Packages:

default

Package containing all default questionnaire forms & models

Modules:

models.py

This module contains all the default questionnaire models

Class definitions:

class apps.questionnaire.default.models.**StartQuestionnaire**(**args*, ***kwargs*)
Bases: *apps.questionnaire.models.QuestionnaireBase*

The start/first questionnaire for a normal control

Parameters

- **id** (*AutoField*) –
- **request_step_id** (ForeignKey to *RequestStep*) –
- **current_status** (*CharField*) – choices=[good, less_good, bad, really_bad, terrible]
- **problems** (*TextField*) –
- **problem_severity** (*IntegerField*) – choices=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
- **hospital_visit** (*CharField*) – choices=[no, yes_urgent_care, yes_intestional_clinic, yes_other_specialist]

display_template

Returns: The template path & name to be used for showing the details page for the filled in data for this questionnaire.

class apps.questionnaire.default.models.**StartUrgentQuestionnaire**(**args*, ***kwargs*)
Bases: *apps.questionnaire.models.QuestionnaireBase*

The start/first questionnaire for a urgent control

Parameters

- **id** (*AutoField*) –
- **request_step_id** (ForeignKey to *RequestStep*) –
- **appointment_period** (*CharField*) – choices=[tomorrow, within_3_days, this_week]

display_template

Returns: The template path & name to be used for showing the details page for the filled in data for this questionnaire.

class apps.questionnaire.default.models.**UrgentProblemQuestionnaire**(**args*, ***kwargs*)
Bases: *apps.questionnaire.models.QuestionnaireBase*

The second part for the urgent control procedure.

Parameters

- **id** (*AutoField*) –
- **request_step_id** (ForeignKey to *RequestStep*) –
- **problems** (*TextField*) –
- **problem_severity** (*IntegerField*) – choices=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

display_template

Returns: The template path & name to be used for showing the details page for the filled in data for this questionnaire.

class apps.questionnaire.default.models.**FinishQuestionnaire**(**args*, ***kwargs*)
Bases: *apps.questionnaire.models.QuestionnaireBase*

Finish/last questionnaire for the normal control

Parameters

- **id** (*AutoField*) –
- **request_step_id** (ForeignKey to *RequestStep*) –
- **appointment** (*CharField*) – choices=[no, yes_phone_nurse, yes_phone_doctor, yes_nurse, yes_doctor]
- **appointment_period** (*CharField*) – choices=[within_4_weeks, within_2_weeks, this_week]
- **appointment_preference** (*ChoiceOtherField*) – choices=[None, other]
- **blood_sample** (*ChoiceOtherField*) – choices=[umcg, martini_ziekenhuis, lab_noord, other]
- **blood_sample_date** (*DateField*) –

display_template

Returns: The template path & name to be used for showing the details page for the filled in data for this questionnaire.

forms.py

This module contains all the default questionnaire forms

See the forms.py in the questionnaire app for documentation on how to manage the forms.

Class definitions:

class apps.questionnaire.default.forms.**StartUrgentQuestionnaireForm**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: appointment_period

class apps.questionnaire.default.forms.**UrgentProblemQuestionnaireForm**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: problems, problem_severity

class apps.questionnaire.default.forms.**StartQuestionnaireForm**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- current_status: current_status
- None: problems

class apps.questionnaire.default.forms.**StartQuestionnaireForm1**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: problem_severity, hospital_visit

class apps.questionnaire.default.forms.**FinishQuestionnaireForm**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: appointment
- appointment: appointment_period, appointment_preference

class apps.questionnaire.default.forms.**FinishQuestionnaireForm1**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fields: blood_sample, blood_sample_date, date_unknown, date_display

rheumatism

This package contains all the forms and models for the rheumatism questionnaires.

Modules:

models.py

This module contains all the rheumatism questionnaire models

Class definitions:

class apps.questionnaire.rheumatism.models.**RADAIQuestionnaire**(**args*, ***kwargs*)
Bases: *apps.questionnaire.models.QuestionnaireBase*

Rheumatoid arthritis Disease Activity Ondex (RADAI) Questionnaire

Parameters

- **id** (*AutoField*) –
- **request_step_id** (ForeignKey to *RequestStep*) –
- **activity_six_month_score** (*CharField*) – choices=[none, almost_unnoticeable, minimal_to_mild, mild, mild_to_moderate, moderate, moderate_to_severe, severe, severe_to_extreme, extreme]
- **activity_today_score** (*CharField*) – choices=[none, almost_unnoticeable, minimal_to_mild, mild, mild_to_moderate, moderate, moderate_to_severe, severe, severe_to_extreme, extreme]
- **pain_today_score** (*CharField*) – choices=[none, almost_unnoticeable, minimal_to_mild, mild, mild_to_moderate, moderate, moderate_to_severe, severe, severe_to_extreme, extreme]
- **stiffness_today_score** (*CharField*) – choices=[none, less_30min, 30_to_60min, 1_to_2hours, 2_to_4hours, 4_to_less1day, all_day]
- **left_shoulder_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]
- **left_elbow_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]

- **left_wrist_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]
- **left_vingers_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]
- **left_hip_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]
- **left_knee_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]
- **left_ankle_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]
- **left_toes_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]
- **right_shoulder_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]
- **right_elbow_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]
- **right_wrist_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]
- **right_vingers_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]
- **right_hip_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]
- **right_knee_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]
- **right_ankle_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]
- **right_toes_pain_score** (*CharField*) – choices=[none, mild, moderate, severe]

graphic_score_display

Returns: The graphic score display to use for the graphic

display_template

Returns: The template path & name to be used for showing the details page for the filled in data for this questionnaire.

class apps.questionnaire.rheumatism.models.**RheumatismSF36**(**args*, ***kwargs*)
Bases: *apps.questionnaire.models.QuestionnaireBase*

Rheumatoid arthritis SF36 questionnaire

Parameters

- **id** (*AutoField*) –
- **request_step_id** (ForeignKey to *RequestStep*) –
- **health_general** (*CharField*) – choices=[excellent, very_good, good, poor, bad]
- **health_changes** (*CharField*) – choices=[much_better, slighty_better, about_same, slighty_worse, much_worse]
- **high_effort_impact** (*CharField*) – choices=[severe, a_bit, not]
- **poor_effort_impact** (*CharField*) – choices=[severe, a_bit, not]
- **carrying_impact** (*CharField*) – choices=[severe, a_bit, not]
- **walking_stairs_impact** (*CharField*) – choices=[severe, a_bit, not]
- **walking_one_stair_impact** (*CharField*) – choices=[severe, a_bit, not]
- **bent_over_impact** (*CharField*) – choices=[severe, a_bit, not]
- **walk_km_impact** (*CharField*) – choices=[severe, a_bit, not]

- **walk_halfkm_impact** (*CharField*) – choices=[severe, a_bit, not]
- **walk_tenthkm_impact** (*CharField*) – choices=[severe, a_bit, not]
- **wash_cloth_impact** (*CharField*) – choices=[severe, a_bit, not]
- **work_less_problem** (*CharField*) – choices=[yes, no]
- **achieve_problem** (*CharField*) – choices=[yes, no]
- **work_limitation_problem** (*CharField*) – choices=[yes, no]
- **work_effort_problem** (*CharField*) – choices=[yes, no]
- **work_less_emotional_problem** (*CharField*) – choices=[yes, no]
- **achieve_emotional_problem** (*CharField*) – choices=[yes, no]
- **accurate_emotional_problem** (*CharField*) – choices=[yes, no]
- **social_impact** (*CharField*) – choices=[nothing, a_bit, a_lot, much, verry_much]
- **pain_score** (*CharField*) – choices=[none, really_light, light, a_lot, severe, really_severe]
- **pain_impact** (*CharField*) – choices=[none, a_bit, a_lot, much, verry_much]
- **cheerful_score** (*CharField*) – choices=[non_stop, often, frequently, sometimes, rarely, never]
- **nervious_score** (*CharField*) – choices=[non_stop, often, frequently, sometimes, rarely, never]
- **blues_score** (*CharField*) – choices=[non_stop, often, frequently, sometimes, rarely, never]
- **calm_score** (*CharField*) – choices=[non_stop, often, frequently, sometimes, rarely, never]
- **energetic_score** (*CharField*) – choices=[non_stop, often, frequently, sometimes, rarely, never]
- **depressed_score** (*CharField*) – choices=[non_stop, often, frequently, sometimes, rarely, never]
- **burnout_score** (*CharField*) – choices=[non_stop, often, frequently, sometimes, rarely, never]
- **happiness_score** (*CharField*) – choices=[non_stop, often, frequently, sometimes, rarely, never]
- **tired_score** (*CharField*) – choices=[non_stop, often, frequently, sometimes, rarely, never]
- **social_visit_impact** (*CharField*) – choices=[non_stop, often, sometimes, rarely, never]
- **easier_ill_score** (*CharField*) – choices=[correct, partly_correct, dontknow, party_incorrect, incorrect]
- **even_healthy_score** (*CharField*) – choices=[correct, partly_correct, dontknow, party_incorrect, incorrect]
- **health_drop_score** (*CharField*) – choices=[correct, partly_correct, dontknow, party_incorrect, incorrect]

- **excellent_health_score** (*CharField*) – choices=[correct, partly_correct, dont-know, party_incorrect, incorrect]

display_template

Returns: The template path & name to be used for showing the details page for the filled in data for this questionnaire.

graphic_score_display

Returns: the score for in the graphic

forms.py

This module contains all the forms for the rheumatism questionnaires.

See the forms.py in the questionnaire app for documentation on how to manage the forms.

Class definitions:

class apps.questionnaire.rheumatism.forms.**RheumatismSF36Form**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: health_general, health_changes, high_effort_impact, poor_effort_impact

class apps.questionnaire.rheumatism.forms.**RheumatismSF36Form1**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: carrying_impact, walking_stairs_impact, walking_one_stair_impact, bent_over_impact

class apps.questionnaire.rheumatism.forms.**RheumatismSF36Form2**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: walk_km_impact, walk_halfkm_impact, walk_tenthkm_impact, wash_cloth_impact

class apps.questionnaire.rheumatism.forms.**RheumatismSF36Form3**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- Combined: work_less_problem, achieve_problem, work_limitation_problem, work_effort_problem

class apps.questionnaire.rheumatism.forms.**RheumatismSF36Form4**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- Combined: work_less_emotional_problem, achieve_emotional_problem, accurate_emotional_problem

class apps.questionnaire.rheumatism.forms.**RheumatismSF36Form5**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: social_impact, pain_score, pain_impact

class apps.questionnaire.rheumatism.forms.**RheumatismSF36Form6**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- Combined: cheerful_score, nervious_score, blues_score, calm_score, energetic_score

class apps.questionnaire.rheumatism.forms.**RheumatismSF36Form7**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- Combined: depressed_score, burnout_score, happiness_score, tired_score

class apps.questionnaire.rheumatism.forms.**RheumatismSF36Form8**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: social_visit_impact, easier_ill_score, even_healthy_score

class apps.questionnaire.rheumatism.forms.**RheumatismSF36Form9**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: health_drop_score, excellent_health_score

class apps.questionnaire.rheumatism.forms.**RADAIQuestionnaireForm**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: activity_six_month_score, activity_today_score, pain_today_score, stiffness_today_score

class apps.questionnaire.rheumatism.forms.**RADAIQuestionnaireForm1**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- patient_image_right: right_shoulder_pain_score, right_elbow_pain_score, right_wrist_pain_score, right_vingers_pain_score
- patient_image_left: left_shoulder_pain_score, left_elbow_pain_score, left_wrist_pain_score, left_vingers_pain_score

class apps.questionnaire.rheumatism.forms.**RADAIQuestionnaireForm2**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- patient_image_right: right_hip_pain_score, right_knee_pain_score, right_ankle_pain_score, right_toes_pain_score
- patient_image_left: left_hip_pain_score, left_knee_pain_score, left_ankle_pain_score, left_toes_pain_score

ibd

This package contains the inflammatory bowel disease (IBD) questionnaire forms and models.

Modules:

models.py

This module contains all the inflammatory bowel disease (IBD) questionnaire models

Class definitions:

class apps.questionnaire.ibd.models.**IBDNauseaVomitTime**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

List of IBD nausea vomit times

Parameters

- **id** (*AutoField*) –
- **name** (*CharField*) –

class apps.questionnaire.ibd.models.**IBDQuestionnaire**(**args*, ***kwargs*)
Bases: *apps.questionnaire.models.QuestionnaireBase*

IBD questionnaire

Parameters

- **id** (*AutoField*) –
- **request_step_id** (ForeignKey to *RequestStep*) –
- **has_stoma** (*CharField*) – choices=[yes, no]
- **has_pouch** (*CharField*) – choices=[yes, no]
- **has_pouch_problems** (*CharField*) – choices=[yes, no]
- **pouch_problems** (*TextField*) –
- **stool_freq** (*CharField*) – choices=[1_3_times, 4_6_times, 7_9_times, more_than_9_times]
- **stool_thickness** (*CharField*) – choices=[hard, normal, mushy, liquid]
- **stool_liquid_freq** (*CharField*) – choices=[less_than_1_time_per_day, 1_2_times_per_day, 3_6_times_per_day, more_than_6_times_per_day]
- **diarrhea_at_night** (*CharField*) – choices=[yes, no]
- **stool_has_blood** (*CharField*) – choices=[no, sometimes, often_or_daily]
- **stool_has_slime** (*CharField*) – choices=[no, sometimes, often_or_daily]
- **stool_urgency** (*CharField*) – choices=[no, yes_sometimes, yes_severe]
- **stool_planning** (*CharField*) – choices=[no_never, yes_sometimes, yes_mostly, yes_always]
- **stool_continence** (*CharField*) – choices=[no_never, yes_sometimes, yes_regulary]
- **stoma_version** (*CharField*) – choices=[colostoma, ileostoma, dont_know]
- **stoma_empty_freq** (*PositiveSmallIntegerField*) –
- **stoma_has_problems** (*CharField*) – choices=[yes, no]
- **stoma_problems** (*TextField*) –
- **nausea_vomit** (*CharField*) – choices=[yes, no]
- **has_fistel** (*CharField*) – choices=[yes, no, dont_know]

- **fistel_location** (*TextField*) –
- **anal_pain** (*CharField*) – choices=[no, yes_mild, yes_severe]
- **anal_problems** (*CharField*) – choices=[yes, no]
- **appetite** (*CharField*) – choices=[good, moderate, bad]
- **patient_weight** (*DecimalField*) –
- **patient_length** (*DecimalField*) –
- **stomach_ache** (*CharField*) – choices=[no, yes_sometimes_mild, yes_sometimes_severe, yes_always_mild, yes_always_severe]
- **fatigue** (*CharField*) – choices=[yes, no]
- **fever** (*CharField*) – choices=[yes, no]
- **fever_specify** (*TextField*) –
- **joint_pain** (*CharField*) – choices=[yes, no]
- **joint_pain_complaints** (*TextField*) –
- **eye_inflammation** (*CharField*) – choices=[yes, no]
- **eye_inflammation_complaints** (*TextField*) –
- **skin_disorder** (*CharField*) – choices=[yes, no]
- **skin_disorder_complaints** (*TextField*) –
- **does_smoke** (*CharField*) – choices=[yes, no]
- **smoke_freq** (*PositiveSmallIntegerField*) –
- **question_remarks** (*TextField*) –

display_template

Returns: The template path & name to be used for showing the details page for the filled in data for this questionnaire.

BMI

Function for calculating the BMI

forms.py

This module contains all the forms for the inflammatory bowel disease (IBD) questionnaires.

See the forms.py in the questionnaire app for documentation on how to manage the forms.

Class definitions:

class apps.questionnaire.ibd.forms.**IDBQuestionnaireForm**(**args*, ***kwargs*)

Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

IBDQuestionnaireForm

Fieldsets:

- None: has_stoma, has_pouch
- has_pouch: has_pouch_problems
- has_pouch_problems: pouch_problems

class apps.questionnaire.ibd.forms.**IDBQuestionnaireForm2A**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

IBDQuestionnaireForm2A

Fieldsets:

- None: stool_freq, stool_thickness
- stool_thickness: stool_liquid_freq
- None: diarrhea_at_night, stool_has_blood, stool_has_slime

class apps.questionnaire.ibd.forms.**IDBQuestionnaireForm3A**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

IBDQuestionnaireForm3A

Fieldsets:

- None: stool_urgency, stool_planning, stool_continence

class apps.questionnaire.ibd.forms.**IDBQuestionnaireForm2B**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

IBDQuestionnaireForm2B

Fieldsets:

- None: stoma_version, stoma_empty_freq, stoma_has_problems
- stoma_has_problems: stoma_problems

class apps.questionnaire.ibd.forms.**IDBQuestionnaireForm4**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

IBDQuestionnaireForm4

Fieldsets:

- None: nausea_vomit
- nausea_vomit: nausea_vomit_time

class apps.questionnaire.ibd.forms.**IDBQuestionnaireForm5**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

IBDQuestionnaireForm5

Fieldsets:

- None: has_fistel
- has_fistel: fistel_location
- None: anal_pain, anal_problems

class apps.questionnaire.ibd.forms.**IDBQuestionnaireForm6**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

IBDQuestionnaireForm6

Fieldsets:

- None: appetite, patient_weight, patient_length, stomach_ache

class apps.questionnaire.ibd.forms.**IDBQuestionnaireForm7**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

IBDQuestionnaireForm7

Fieldsets:

- None: fatigue, fever
- fever: fever_specify

class apps.questionnaire.ibd.forms.**IDBQuestionnaireForm8**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

IBDQuestionnaireForm8

Fieldsets:

- None: joint_pain
- joint_pain: joint_pain_complaints

class apps.questionnaire.ibd.forms.**IDBQuestionnaireForm9**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

IBDQuestionnaireForm9

Fieldsets:

- None: eye_inflammation
- eye_inflammation: eye_inflammation_complaints
- None: skin_disorder
- skin_disorder: skin_disorder_complaints

class apps.questionnaire.ibd.forms.**IDBQuestionnaireForm10**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

IBDQuestionnaireForm10

Fieldsets:

- None: does_smoke
- does_smoke: smoke_freq
- None: question_remarks

qohc

This package contains all the forms and models for the quality of healthcare (QOHC) questionnaires.

Modules:

models.py

This module contains all the quality of healthcare (QOHC) questionnaire models

Class definitions:

class apps.questionnaire.qohc.models.**QOHCQuestionnaire**(**args*, ***kwargs*)
Bases: *apps.questionnaire.models.QuestionnaireBase*

Quality of health care questionnaire

Parameters

- **id** (*AutoField*) –
- **request_step_id** (ForeignKey to *RequestStep*) –
- **not_fill_in** (*CharField*) – choices=[yes, no]
- **hc_satisfaction_score** (*IntegerField*) – choices=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
- **serious_score** (*CharField*) – choices=[not_at_all, a_bit, largely, yes_very]
- **friendly_score** (*CharField*) – choices=[not_at_all, a_bit, largely, yes_very]
- **information_score** (*CharField*) – choices=[not_at_all, a_bit, largely, yes_very]
- **rc_satisfation_score** (*IntegerField*) – choices=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

graphic_score_display

Returns: The graphic score display to use for the graphic

display_template

Returns: The template path & name to be used for showing the details page for the filled in data for this questionnaire.

forms.py

This module contains all the forms for the quality of healthcare (QOHC) questionnaires.

See the forms.py in the questionnaire app for documentation on how to manage the forms.

Class definitions:

class apps.questionnaire.qohc.forms.**QOHCQuestionnaireForm**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: not_fill_in

class apps.questionnaire.qohc.forms.**QOHCQuestionnaireForm1**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

QOHCQuestionnaireForm

Fieldsets:

- None: hc_satisfaction_score, serious_score, friendly_score, information_score, rc_satisfation_score

qol

This package contains all the forms and models for the quality of life (QOL) questionnaires.

Modules:

models.py

This module contains all the quality of life (QOL) questionnaire models

Class definitions:

class apps.questionnaire.qol.models.**QOLPracticalProblem**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Stores a list of QOL practical problems

Parameters

- **id** (*AutoField*) –
- **name** (*CharField*) –

class apps.questionnaire.qol.models.**QOLSocialProblem**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Stores a list of QOL social problems

Parameters

- **id** (*AutoField*) –
- **name** (*CharField*) –

class apps.questionnaire.qol.models.**QOLEmotionalProblem**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Stores a list of QOL emotional problems

Parameters

- **id** (*AutoField*) –
- **name** (*CharField*) –

class apps.questionnaire.qol.models.**QOLSpiritualProblem**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Stores a list of QOL spiritual problems

Parameters

- **id** (*AutoField*) –
- **name** (*CharField*) –

class apps.questionnaire.qol.models.**QOLFysicalProblem**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Stores a list of QOL fysicial problems

Parameters

- **id** (*AutoField*) –
- **name** (*CharField*) –

class apps.questionnaire.qol.models.**QOLChronCUQuestionnaire**(**args*, ***kwargs*)
Bases: *apps.questionnaire.models.QuestionnaireBase*

QOL for Chron en Colitis Ulcerosa

Parameters

- **id** (*AutoField*) –

- **request_step_id** (ForeignKey to *RequestStep*) –
- **hasproblems** (*CharField*) – choices=[yes, no]
- **other_problems** (*TextField*) –
- **need_contact** (*CharField*) – choices=[yes, no, maybe]

display_template

Returns: The template path & name to be used for showing the details page for the filled in data for this questionnaire.

graphic_score_display

Returns: The score to display in the graphic

class apps.questionnaire.qol.models.**QOLQuestionnaire**(**args*, ***kwargs*)
Bases: *apps.questionnaire.qol.models.QOLChronCUQuestionnaire*

QOL generic version (include fysical problems)

Parameters

- **id** (*AutoField*) –
- **request_step_id** (ForeignKey to *RequestStep*) –
- **hasproblems** (*CharField*) – choices=[yes, no]
- **other_problems** (*TextField*) –
- **need_contact** (*CharField*) – choices=[yes, no, maybe]
- **qolchroncuquestionnaire_ptr_id** (OneToOneField to *QOLChronCUQuestionnaire*) –

display_template

Returns: The template path & name to be used for showing the details page for the filled in data for this questionnaire.

forms.py

This module contains all the forms for the quality of life (QOL) questionnaires.

See the forms.py in the questionnaire app for documentation on how to manage the forms.

Class definitions:

class apps.questionnaire.qol.forms.**QOLChronCUQuestionnaireForm**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None:

class apps.questionnaire.qol.forms.**QOLChronCUQuestionnaireForm1**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

QOLChronCUQuestionnaireForm

Fieldsets:

- None: hasproblems
- hasproblems: practical_problems

class apps.questionnaire.qol.forms.**QOLChronCUQuestionnaireForm2**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: social_problems, emotional_problems

class apps.questionnaire.qol.forms.**QOLChronCUQuestionnaireForm3**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: spiritual_problems

class apps.questionnaire.qol.forms.**QOLChronCUQuestionnaireForm4**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: other_problems, need_contact

class apps.questionnaire.qol.forms.**QOLQuestionnaireForm**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: hasproblems
- hasproblems: practical_problems

class apps.questionnaire.qol.forms.**QOLQuestionnaireForm1**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: social_problems, emotional_problems

class apps.questionnaire.qol.forms.**QOLQuestionnaireForm2**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: spiritual_problems, fysical_problems

class apps.questionnaire.qol.forms.**QOLQuestionnaireForm3**(**args*, ***kwargs*)
Bases: *apps.questionnaire.forms.BaseQuestionnaireForm*

Fieldsets:

- None: other_problems, need_contact

Modules:

tests.py

Contains questionnaire tests. By default only tests the wizard questionnaire processes only for one diagnose. But all other forms are tested seperately.

Class definitions:

class apps.questionnaire.tests.**MockWizard**
Mock Wizard object with a method for getting the cleaned_data

get_cleaned_data_for_form_class(*form_class*)
Get the cleaned data for a form_class

Args: The form_class, unused

Returns: self.cleaned_data

class apps.questionnaire.tests.**QuestionnaireTest**(*methodName='runTest'*)
Bases: *core.unittest.baseunittest.BaseUnitTest*

Provides tests for testing Questionnaires

questionnaire_post_form(*res*, *url*, *status_code=200*, *save_and_exit=False*, *previous_step=False*)
Post a questionnaire form

Args:

- res: the response instance
- url: the url to post to
- status_code: the expected status_code (default=200)
- save_and_exit: perform a save and exit the questionnaire
- previouse_step: if True go one step back

Returns: The response instance after posting

run_questionnaire(*urgent=False*)
Runs a complete set of questionnaires for an urgent or normal control

Args:

- urgent: run the urgent control

get_questionnaire_fields_from_object(*obj*)
Gets the questionnaire fields from a given questionnaire instance

Args:

- obj: the Questionnaire instance

Returns:

- All fields from the obj

run_questionnaire_for_patient(*email*, *urgent=False*)
Runs a full control for a patient

Args:

- email: the email to use for login
- urgent: if true run it for an urgent control

check_form(*form_class*)
Check some default form methods for a given form_class

Args:

- form_class: the form_class to check

check_other_forms()
Run over all other (untested) questionnaire forms by initializing, cleaning and checking the 'condition' function

check_controle_process()
To speed up testing only do the whole control and urgent control for one patient. All other forms that are not hit by this test are tested in the 'check_other_forms' function

Currently testes the following diagnoses: colitis_ulcerosa (= same as chron), intestinal_transplantation, rheumatoid_arthritis

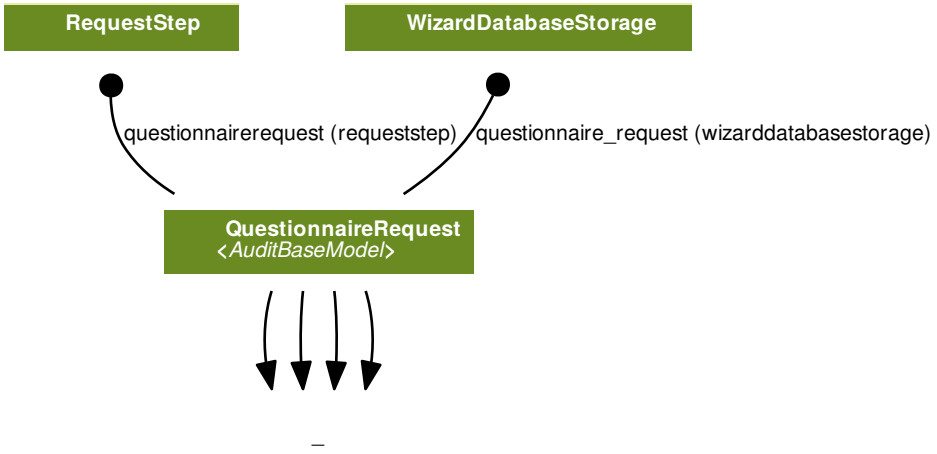
check_controle_navigation(*urgent*)
Partially fill in control and check the navigation functionality + save partially filled in & return

test_controles()
Questionnaire tests runner

models.py

This module contains the questionnaire model definitions which both includes the generic questionnaire parts as well as the disease specific questionnaire model definitions.

The main entry point for a control is the *QuestionnaireRequest* which can have multiple related *RequestStep* instances. Every questionnaire is a subclass of *QuestionnaireBase* which is coupled to a RequestStep. The *WizardDatabaseStorage* model is used to temporarily store the data. The relationships are shown in the following model diagram:



–

Questionnaires models and associated forms should be included into their own apps. For each added questionnaire model the PACKAGE_LOCATION dict needs to be updated. This dict is used by both the get_model_class method in this module as well as the get_forms_for method in the forms.py module.

Models can be defined as the following the example:

```
class IBDQuestionnaire(QuestionnaireBase):
    #Make sure to subclass QuestionnaireBase

    #Add this property to return a template that can be used
    #to display all the filled in values, see other templates
    #for the basic layout.
    @property
    def display_template(self):
```

```
        return 'questionnaire/details/IBDQuestionnaire.html'

    #Returns the graphic score to display in the graphic
    # Currently only works for a couple predefined questionnaire
    # categories: Ziekteactiviteit, kwaliteitvanleven, kwaliteitvanzorg
    @property
    def graphic_score_display(self):
        return str(self.BMI).replace(',', '.')

    #The axis maximum score for the graph
    @property
    def graphic_score_max(self):
        return 40

    #The axis minimum score for the graph
    @property
    def graphic_score_min(self):
        return 10

    #The axis score name to set for the graphic
    @property
    def graphic_score_name(self):
        return _('BMI')

    #The category of this questionnaire and also the name to display
    display_name = _('Ziekteactiviteit')

    #The lower case name of this questionnaire
    lower_case_name = 'ibd_questionnaire'
```

Note: After adding new questionnaires make sure to update the PACKAGE_LOCATION dict in models.py

Class and function definitions:

apps.questionnaire.models.**get_model_class**(*model_class_name*)

Get the model_class by the model_name, uses the PACKAGE_LOCATION list to get the package location.

Args:

- model_class_name: the model class name (case sensitive)

Returns: The model class for model_class_name

class apps.questionnaire.models.**QuestionnaireRequest**(**args*, ***kwargs*)

Bases: *core.models.AuditBaseModel*

The questionnaire request is the base for a periodic control. It couples the patient to the request steps which specify the questionnaire models that need to be filled in by the patient.

It also keeps tracks of the questionnaire status like the deadline, finished_date and read_on date.

Parameters

- **id** (*AutoField*) –
- **patient_id** (ForeignKey to *Patient*) –
- **urgent** (*BooleanField*) –
- **practitioner_id** (ForeignKey to *HealthProfessional*) –

- **deadline_nr** (*IntegerField*) –
- **deadline** (*DateField*) –
- **created_on** (DateField) –
- **finished_on** (DateField) –
- **read_on** (DateField) –
- **patient_diagnose** (*CharField*) – choices=[rheumatoid_arthritis, chron, colitis_ulcerosa, intestinal_transplantation]
- **saved_finish_later** (*BooleanField*) –
- **last_filled_in_step** (*CharField*) –
- **last_filled_in_form_step** (*CharField*) –
- **handled_on** (DateField) –
- **handled_by_id** (ForeignKey to *HealthProfessional*) –
- **appointment_needed** (*BooleanField*) –
- **appointment_added_on** (DateField) –
- **appointment_added_by_id** (ForeignKey to *Secretary*) –

filled_in

Returns: True if the finished_on is set else False

handled

Returns: True if the handled_on is set else False

appointment_arranged

Returns: Ja if the appointment is arranged else Nee

appointment_on_short_term

Returns: True if the appointment is on short term else False

blood_taken

Returns: True if blood taken question is answered postive or False

blood_taken_date

Returns: The last blood taken date or None

patient_needs_appointment

Returns: True if the patient needs to have an appointment else False

appointment_period_date

Returns: The appointment period date or None

appointment_period

Returns: The appointment period or None

class apps.questionnaire.models.**RequestStep**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

The requeststep model is coupled to the *QuestionnaireRequest* and is used to store the questionnaire steps of the control.

Every step is coupled to one and only one Questionnaire model class. Which this model class the forms are initalized and displayed to the user with help of the wizard.

The step_nr field is used for sorting.

Parameters

- **id** (*AutoField*) –
- **questionnairerequest_id** (ForeignKey to *QuestionnaireRequest*) –
- **step_nr** (*IntegerField*) –
- **model** (*CharField*) – choices=[StartQuestionnaire, IBDQuestionnaire, RADAIQuestionnaire, QOLChronCUQuestionnaire, QOLQuestionnaire, RheumatismSF36, QOHCQuestionnaire, FinishQuestionnaire, StartUrgentQuestionnaire, UrgentProblemQuestionnaire]

model_class

Returns: The model_class of associated with the model (name) attribute of this RequestStep

questionnaire

Returns: The filled in questionnaire for this RequestStep or None

class apps.questionnaire.models.**WizardDatabaseStorage**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Wizard database storage, saves both clean and unclean data in the data field in json format. The wizard uses an instance of this model to temporarily store all filled in questionnaire data until all questionnaires are filled in correctly.

Parameters

- **id** (*AutoField*) –
- **questionnaire_request_id** (ForeignKey to *QuestionnaireRequest*) –
- **data** (*TextField*) –
- **created_on** (DateField) –

class apps.questionnaire.models.**QuestionnaireBase**(**args*, ***kwargs*)
Bases: *core.models.AuditBaseModel*

Abstract class which is the baseclass for every Questionnaire models. Couples to one and only one *RequestStep*

Parameters **request_step_id** (ForeignKey to *RequestStep*) –

patient

Returns: The patient for the questionnaire request associated with the requeststep for this the questionnaire

finished_on

Returns: The finished date of the questionnaire request or None

get_finished_on_timestamp

Returns: The finished timestamp of the questionnaire request or None

forms.py

This module contains the baseclass for all questionnaire forms and the functions for getting all forms based on a questionnaire model class.

To add a new form, a new apps needs to be added including a models.py with the new questionnaire model and a forms.py file with all the new forms.

Note: Make sure to update the PACKAGE_LOCATION dict in models.py

Forms can be defined as the following the example:

```
class IDBQuestionnaireForm3A(BaseQuestionnaireForm):
    #Forms need to subclass BaseQuestionnaireForm.

    #The form_template which is used rendering the template
    form_template = 'questionnaire/DefaultQuestionnaireForm.html'

    #The number of the form. This number is used to sort the
    #forms when they are retrieved for a model
    form_nr = 3

    #Condition method for conditionally showing the form (True=show)
    #Forms that are not shown are also not validated.
    #The provided wizard argument can be used to get the cleaned
    #data of a form like shown in the example below.
    def condition(self, wizard):
        cleaned_data = wizard.get_cleaned_data_for_form_class(
            IDBQuestionnaireForm)
        if cleaned_data:
            if 'has_stoma' in cleaned_data:
                if cleaned_data['has_stoma'] == 'yes':
                    return False
        return True

    class Meta:
        model = IBDQuestionnaire

        fieldsets = (
            (None, {'fields': (
                'stool_urgency', 'stool_planning', 'stool_continence',)}),
        )

        #By using the create_exclude_list method you only have to specify
        #the fieldsets. The excluded list is computed automatically based
        #on the fieldsets en model definition. For multiple forms for one
        #model this saves a lot of time & possibilities for errors.
        exclude = create_exclude_list(model, fieldsets)
```

The get_forms_for method automatically will pick up the new forms when the package is added to the PACKAGE_LOCATION dict in models.py.

Class definitions:

apps.questionnaire.forms.**form_sorting_function**(*form*)

Simple sorting helper function

Args:

- form: a form instance

Returns: The form_nr of the form.

apps.questionnaire.forms.**get_forms_for**(*questionnaire_model_class*)
Helper function for getting all forms for a questionnaire model class.

Args:

- questionnaire_model_class: the class of the questionnaire model.

Returns: A list of forms for a questionnaire model class, sorted on form_nr.

apps.questionnaire.forms.**create_exclude_list**(*model*, *fieldsets*)
Helper method for automatic generating the exclude list of fields based on a model and a fieldset definition. Allows to only set the fieldsets attribute and use this function for the exclude list.

Args:

- model: the model to inspect for modelfields
- fieldsets: the fieldsets that should be included

Returns: A tuple with all model fields excluding the fields in fieldsets

class apps.questionnaire.forms.**BaseQuestionnaireForm**(**args*, ***kwargs*)
Bases: *core.forms.BaseModelForm*

Base form for all questionnaires uses a form_nr attribute for sorting and a condition method for conditionally showing/using the form.

condition(*wizard*)
Override this function to allow conditional showing/using forms. If the form is not shown it is not validated.

Args:

- wizard: the wizard instance that is used to show the forms which can be used to get clean and unclean data of other steps/forms to determine if the form should be shown.

Returns: True if the form should be used else False (default=True)

storage.py

This module contains a storage model definition used by *apps.questionnaire.wizards.QuestionnaireWizard* to temporarily store all filled in information from the questionnaires.

Class definitions:

class apps.questionnaire.storage.**DatabaseStorage**(**args*, ***kwargs*)
Bases: formtools.wizard.storage.base.BaseStorage

Wizard database storage class stores the temporary data of a wizard in a database instead of the session or a cookie

init_data()
Override the method to append the unclean_data key

get_unclean_data(*step*)
Get the unclean data for a step

Args:

- step: the step key

Returns: The values in dict format

set_unclean_data(*step*, *unclean_form_data*)
Set the unclean data for a step

Args:

- step: The step key to set the unclean_data for
- unclean_form_data: the unclean data from a form

load_data()
Load the data from the database

Returns: A dict with the loaded data

update_response(*response*)
Stores the data, automatically called before the response is sent to the client

Args:

- response: the response instance

views.py

This module contains the views for displaying filled in questionnaires and the start and finish questionnaire views.

The questionnaire fillin process is handled by the *apps.questionnaire.wizards.QuestionnaireWizard* view.

Class and function definitions:

apps.questionnaire.views.**insert_new_questionnaire_request_for_patient**(*patient*)
Adds a new questionnaire request for a patient including the requeststeps

Args:

- patient: the patient to add the questionnaire_request for

Returns: The created questionnaire_request

class apps.questionnaire.views.**PatientView**(**kwargs*)
Bases: django.views.generic.base.View

Basic patient view which checks if the user has logged in and automatically sets the patient on the view

set_patient(**kwargs*)
Sets the patient on the view

class apps.questionnaire.views.**PatientTemplateView**(**kwargs*)
Bases: *apps.questionnaire.views.PatientView*, django.views.generic.base.TemplateView

Templateview baseclass view

class apps.questionnaire.views.**PatientQuestionnaireTemplateView**(**kwargs*)
Bases: *apps.questionnaire.views.PatientView*, django.views.generic.base.TemplateView

Base view for all questionnaire views that use templateview. Sets the questionnaire_request on the view

class apps.questionnaire.views.**BaseOverviewTemplateView**(**kwargs*)
Bases: *apps.questionnaire.views.PatientTemplateView*

Base view for overview views that show the details pages for filled in questionnaires Automatically adds the context.

get_questionnaires_for(*selected_model_class*, *selected_questionnaire_request*)
Retrieve the questionnaires coupled to the selected_questionnaire_request for the selected_model_class

Args:

- selected_model_class: the model_class to find the questionnaire for
- selected_questionnaire_request: the questionnaire_request which should be coupled to the questionnaires.

Returns: [The selected questionnaire, list of questionnaires]

get_context()
Creates the context based on the questionnaire request and model_display_name adding various information.

Returns: The context dict

class apps.questionnaire.views.**DiseaseActivityOverview**(***kwargs*)
Bases: *apps.questionnaire.views.BaseOverviewTemplateView*

Shows the disease activity overview page

class apps.questionnaire.views.**QualityOfLifeOverview**(***kwargs*)
Bases: *apps.questionnaire.views.BaseOverviewTemplateView*

Shows the quality of life overview page

class apps.questionnaire.views.**HealthcareQualityOverview**(***kwargs*)
Bases: *apps.questionnaire.views.BaseOverviewTemplateView*

Shows the healthcare quality overview page

class apps.questionnaire.views.**QuestionnaireStartControle**(***kwargs*)
Bases: *apps.questionnaire.views.PatientView*

Start page for normal (non urgent) controls. The view checks if there is a partially filled in control and let's the user finish it or redirects the user to the first step of the questionnaires fillin procedure.

class apps.questionnaire.views.**QuestionnaireStartUrgent**(***kwargs*)
Bases: *apps.questionnaire.views.PatientView*

Start page for urgen controls. The view checks if there is a partially filled in control and let's the user finish it or redirects the user to the first step of the questionnaires fillin procedure.

class apps.questionnaire.views.**QuestionnaireFinishedView**(***kwargs*)
Bases: *apps.questionnaire.views.PatientQuestionnaireTemplateView*

Shows a succesfully finished all questionnaires view. Is both used for urgent and non urgent controls.

class apps.questionnaire.views.**QuestionnaireRequestRemove**(***kwargs*)
Bases: *apps.questionnaire.views.PatientView*

View which allows to remove a questionnaire request. Used when canceling the control.

wizards.py

This module contains the subclassed WizardView which allows filling in a list of forms.

The wizard uses a *apps.questionnaire.storage.DatabaseStorage* instance to store the filled in information.

In short the wizard uses the following internal procedure for displaying forms and saving data:

1. The wizard view is called with a questionnaire_request.id. The corresponding questionnaire_request is used to get the associated requeststeps. From these steps the models are used to get the full list of forms for the wizard.
2. By using the step queryparameter (or by default the first step) the corresponding form is selected to be rendered. From the storage the cleaned data or unclean data is used to initialize the form with data.
3. After an HTML post the form is checked for validation. If it validates the clean_data is saved as cleaned-data, else it is stored as unclean data.
4. When the last form has been posted all clean data is retrieved from the storage instance and used to initalize & save instances of the models which are coupled to the forms.

Class definitions:

class apps.questionnaire.wizards.**QuestionnaireWizard**(***kwargs*)
Bases: formtools.wizard.views.WizardView

Questionnaire Wizard view class Used for all questionnaires

init_form_list(*form_list*)
Re-executes the below code after the form_list is updated with the forms based on the models coupled to the questionnaires for the questionnaire_request

Args: -form_list: the list of forms

Returns: The computed form list which is dict of forms with a zero based counter

strip_initial_data(*step*, *posted_data*)
Helper function for stripping posted data to be usable as initial_data

Args:

- step: the step key
- posted_data: the posted_data in dict form

Returns: The initial data based on the post_data

get_form_initial(*step*)
Get the initial data to initialize the form with

Args:

- step: is the step key

Returns: The initial data for a form based on the step key

get_prefix(**args*, ***kwargs*)
The questionnaire_request_id is used as prefix for storing data in the storage instance

---

Note: This is not really necessary since the storage instance is also coupled to the questionnaire_request

---

init_forms(*request*, **args*, ***kwargs*)
Initialize the form_list by getting all forms for the models for each request_step coupled to the questionnaire_requests

Args:

- request: the request instance

get_form_list()
Override of the baseclass method to insert custom condition testing on the forms.

Returns: A list of forms exluding the forms for which the condition returns False

post (*request*, **args*, ***kwargs*)
Processes the post_data when a form is posted, is also called when the next or back button or save_and_exit is clicked. Allows saving partially filled (invalid) forms.

Args:

- request: the request instance

Returns: The response of the post method of the superclass

get (*request*, **args*, ***kwargs*)
The get function either loads the first form or the form corresponding to the step querystring parameter

Args:

- request: the request instance

Returns: The response of the render_goto_step function

get_form_kwargs (*step=None*)
Adds the patient as default kwargs

Args:

- step: the step key (default = None)

Returns: A form kwargs dict.

process_step (*form*)
Processes a valid form step by returning the data and remove unclean_data if present. This function is called in the post method.

Args:

- form: the form to process

Returns: The form step data for the given form

get_template_names ()

Returns: The default wizard template name

get_cleaned_data_for_form_class (*form_class*)
Helper function for getting the cleaned_data for a form class

Args:

- form_class: the form_class to get the cleaned_data for

Returns: The cleaned data for the form step or None

get_context_data (*form*, ***kwargs*)
Adds extra context data for rendering the template

Args:

- form: the form to include and get the context for

Returns: A dict with the context to use for rendering the template

done (*form_list*, ***kwargs*)
Save all gathered data and redirect to the finished page Called when the all forms are valid and the last form is posted.

Args:

- form_list: the list with all processed forms

Returns: Redirect to 'finished url'

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

C

Index

A

B

C

D

E

F

G

H

I

J

L

Q

R

S

T

U

V

W

X

Y