# shmir Documentation

***Release 2.0***

**Sylewster Brzęczkowski, Mateusz Flieger, Piotr Rogulski, Michał F**

September 03, 2015

Contents

Contents:

CHAPTER 1

# shmir package

## 1.1 Subpackages

### 1.1.1 shmir.data package

#### Submodules

#### shmir.data.models module

**class** shmir.data.models.**Backbone**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Backbone class with information about miRNA scaffolds

**active_strand**

**flanks3_a**

**flanks3_s**

**flanks5_a**

**flanks5_s**

**generate_regexp**()
Function creates regexps based on active_strand and saves it to the database.

**active_strand: if is equal to 3, function use miRNA_a;** if is equal to 1 or 5, function use miRNA_s if is equal to 0, function use both

**classmethod generate_regexp_all**()
Function takes all objects from the database and creates regular expressions for each.

**homogeneity**

**id**

**loop_a**

**loop_s**

**miRBase_link**

**miRNA_a**

**miRNA_end_3**

**miRNA_end_5**

**miRNA_length**

**miRNA_max**

**miRNA_min**

**miRNA_s**

**name**

**regexp**

**siRNA1 = None**

**siRNA2 = None**

**structure**

**template**()
Returns the template of DNA (sh-miR)

siRNA1 and siRNA2 are siRNA strands and they must be initialized before using this method

**Returns:** Sequence of sh-miR molecule on the base of chosen miRNA scaffold

**class** shmir.data.models.**Immuno**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Immuno motives class

**classmethod check_is_in_sequence**(*input_sequence*)
Checks if input sequence conteins sequences from immuno database

**Args:** input_sequence: RNA sequence of about 20nt length

**Returns:** Bool if the input_sequence contains immunostimulatory motifs

**id**

**link**

**receptor**

**sequence**

**class** shmir.data.models.**InputData**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Table storing input data to sh-miR algorithm

**id**

**immunostimulatory**

**maximum_CG**

**maximum_offtarget**

**minimum_CG**

**results**

**scaffold**

**transcript_name**

**class** shmir.data.models.**Result**(**kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

sh-miR results table

**as_json**()

**backbone**

**get_task_id**()

**id**

**input_data**

**input_id**

**pdf**

**score**

**sequence**

**shmir**

#### shmir.data.ncbi_api module

shmir.data.ncbi_api.**get_data**(*transcript*, *database='nucleotide'*, *email='amupoznan@gmail.com'*)
Function responsible to get data from given ncbi database.

**Args:** email(str): Email to authorization with ncbi. transcript(str): Transcript to search. database(str): Name of database from ncbi. ids(list of str): List of ids to search.

shmir.data.ncbi_api.**get_mRNA**(*transcript*, *database='nucleotide'*, *email='amupoznan@gmail.com'*)
Function to connect with NCBI database and get transcript by name

**Args:** transcript(str): name of transcript (from NCBI) database: name of database in which we look for (default "nucleotide") email(str): email to which NCBI needs to validate

**Returns:** mRNA(str)

**Raises:** NoResultError

#### Module contents

### 1.1.2 shmir.designer package

#### Submodules

#### shmir.designer.design module

#### shmir.designer.errors module

**exception** shmir.designer.errors.**BaseException**(*message=None*)
Bases: exceptions.Exception

Base exception class.

**exception** shmir.designer.errors.**IncorrectDataError**(*message=None*)
Bases: shmir.designer.errors.BaseException

Exception error class for incorrect data input

**exception** shmir.designer.errors.**NoResultError**(*message=None*)
Bases: shmir.designer.errors.BaseException

Exception error class for no results

**exception** shmir.designer.errors.**ValidationError**(*message=None*)
Bases: shmir.designer.errors.BaseException

Exception error class for incorrect input

### shmir.designer.offtarget module

### shmir.designer.score module

### shmir.designer.search module

### shmir.designer.ss module

### shmir.designer.utils module

shmir.designer.utils.**adjusted_frames**(*seq1*, *seq2*, *shift_left*, *shift_right*, *all_frames*)
Take output of parse_input function and insert into flanking sequences. take from database all miRNA results and check if ends of input is suitable for flanking sequences. If first value == and miRNA_end_5 second value == miRNA_end_3 then simply concatenate sequences flanks5_s + first_sequence + loop_s + second_sequence + flanks3_s. If any end is different function has to modify end of the insert: Right end: if miRNA_end_5 < first_end add to right site of second sequence additional nucleotides (as many as **|miRNA_end_5 - first_end|**) like (dots are nucleotides to add, big letter are flanking sequences, small are input):

AAAGGGGCTTTTagtcttaga TTTCCCCGAA....agaatct

if miRNA_end_5 > first_end cut nucleotides from rigth site of flanks3_s and/or from right site of second sequence

before cut: AAAGGGGCTTTTagtcttaga TTTCCCCGAAAATTcctcagaatct (-2, +2)

After AAAGGGGCTTTTagtcttaga TTTCCCCGAAAAtcagaatct

Nucleotides are always added to the right side of sequences. We cut off nucleotides only from flanking sequences or loop.

**Args:** seq1(str) first sequence. seq2(str) second sequence. shift_left(int) - shift on sequence from left side. shift_right(int) - shift on sequence from right side. all_frames(pri-miRNA objects) frames from which we create sh-miRs.

**Returns:** list of tuples (changed frame, first sequence, second sequence).

shmir.designer.utils.**reverse_complement**(*sequence*)
Generates reverse complement sequence to given

**Args:** sequence(str).

**Returns:** revese complement sequence(str) to given.

#### shmir.designer.validators module

#### Module contents

### 1.1.3 shmir.tests package

#### Submodules

#### shmir.tests.test_db module

**class** shmir.tests.test_db.**TestBackboneModel**(*methodName='runTest'*)
Bases: shmir.testing.TestModelBase

**test_adjusted_frames**()

**test_reverse_complement**()

**test_template**()

**class** shmir.tests.test_db.**TestModelBaseCase**(*methodName='runTest'*)
Bases: shmir.testing.TestModelBase

**class TestModel**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

**id**

TestModelBaseCase.**test_get_and_put_to_db**()

#### shmir.tests.test_ncbi_api module

**class** shmir.tests.test_ncbi_api.**TestGetDataFromNcbi**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**BAD_MNRA = 'NC_000071.6'**

**DATA_BASE = 'nucleotide'**

**GOOD_MRNA = 'NM_001128164.1'**

**test_get_data**(**args*, ***keywargs*)

**test_get_mRNA**(**args*, ***keywargs*)

**test_incorrect_get_mRNA**(**args*, ***keywargs*)

**test_no_results_get_mRNA**(**args*, ***keywargs*)

#### shmir.tests.test_validators module

**class** shmir.tests.test_validators.**TestTranscriptValidators**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**test_calculate_gc_content**()

**test_validate_gc_content**()

**test_validate_gc_content_not_in_range**()

### Module contents

## 1.2 Submodules

## 1.3 shmir.async module

shmir.async.**get_async_result**(*task*, *task_id*, *timeout=1.0*, *only_status=False*)
Gets AsyncResult of task, excepting TimeoutError and handling failures

**Args:** task: the task from which want extract result from task_id: id of task timeout(float): timeout of task only_status(bool): boolean if function should return only status (default: False)

**Returns:** Task response

## 1.4 shmir.contextmanagers module

shmir.contextmanagers.**blast_path**(**args*, ***kwds*)
Context manager which changes path for blast executions

shmir.contextmanagers.**generic_path**(*path*)
Function which changes current path to given path

**Args:** path(str): path where interpreter should be

shmir.contextmanagers.**mfold_path**(**args*, ***kwds*)
Context manager which changes path for mfold tasks

**Args:** task_id: id of mfold task

## 1.5 shmir.decorators module

shmir.decorators.**catch_errors**(**errors*)
Decorator to catch specific errors given

**Args:** *errors: All error which should be catched

**Returns:** dict with error status and its message or function result

shmir.decorators.**send_email**(*file_handler=None*)
Decorator to send email after task

**Args:** file_handler: function to handle files

## 1.6 shmir.mfold module

## 1.7 shmir.result_handlers module

shmir.result_handlers.**zip_file_mfold**(*path*)
Function which creates zip_msg to send via email

**Args:** path: path of zip file

**Returns:** tuple of zip message

shmir.result_handlers.**zip_files_from_sirna**(*struct*)
Function which generates email msg from siRNA

**Args:** struct: sh-miR struct

**Returns:** zip message

shmir.result_handlers.**zip_files_from_transcript**(*struct*)
Function which generates email msg from transcript

**Args:** struct: sh-miR struct

**Returns:** zip message

## 1.8 shmir.settings module

shmir.settings.**get_bool**(*section*, *option*, *default=None*)
Function which gets settings

**Args:** section(str): section of settings option(str): option of settings

**Returns:** config(bool) or default value

shmir.settings.**get_config**(*section*, *option*, *default=None*)
Function which gets settings

**Args:** section(str): section of settings option(str): option of settings

**Returns:** config or default value

shmir.settings.**get_db_config**(*option*, *default=None*)

shmir.settings.**get_int**(*section*, *option*, *default=None*)
Function which gets settings

**Args:** section(str): section of settings option(str): option of settings

**Returns:** config(int) or default value

## 1.9 shmir.testing module

**class** shmir.testing.**TestModelBase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

**get_all**(*model*)

**put_to_db**(*obj*)

**setUp**()

**tearDown**()

shmir.testing.**create_backbone**()
Function to create default backbone object

## 1.10 shmir.utils module

## 1.11 shmir.views module

shmir.views.**mfold_task_creator**(**args*, ***kwargs*)
Handler to create folded structure via mfold for specific sequence.

**Args:** data: sequence

**Returns:** Task id of this task

shmir.views.**mfold_task_result**(*task_id*)
Handler for getting mfold result from just a task id

**Args:** task_id: Id of task generated via RESTful API

**Returns:** Json object of status or sends zipped file

shmir.views.**mfold_task_status**(*task_id*)
Mfold status getter

**Args:** task_id: Id of task generated via RESTful API

**Returns:** Json object with status of specific id

shmir.views.**scaffolds**(**args*, ***kwargs*)
Handler to list all possible backbones

**Returns:** Json object with names of backbones list

shmir.views.**sirna_task_creator**(**args*, ***kwargs*)
Handler to initialize task which creates sh-miR(s) from siRNA

**Args:**

**data: one siRNA strand (active) or two siRNA strands separated by space.** First strand is active, both are in 5-3 orientation.

**Returns:** Task id

shmir.views.**sirna_task_result**(*task_id*)
Handler to give result of sh-miR(s) created from siRNA

**Args:** task_id: Id of task generated via RESTful API

**Returns:** Json object with sh-miRs created from siRNA: [(score, sh-miR, bacbone name, pdf(id to download pdf via mfold)...]

shmir.views.**sirna_task_status**(*task_id*)
Handler to check status of task which creates sh-miR from siRNA

**Args:** task_id: Id of task generated via RESTful API

**Returns:** Json object with status of given task_id

shmir.views.**transcript_task_creator**(**args*, ***kwargs*)
Handler to create sh-miR from transcript

**Args:** transcript_name: name of transcript

**Returns:** Id of task

shmir.views.**transcript_task_result**(*task_id*)
Handler to get results of task which creates sh-miR(s) from transcript

**Args:** task_id: Id of task generated via RESTful API

**Returns:** Json object with list of sh-miRs in structure: [sh-miR, score, pdf(id to download pdf via mfold), sequence and backbone name...]

shmir.views.**transcript_task_status**(*task_id*)
Handler to check status of task which creates sh-miR from transcript

**Args:** task_id: Id of task generated via RESTful API

**Returns:** Json object with status of given task_id

## 1.12 Module contents

CHAPTER 2

# Indices and tables

- *genindex*
- *modindex*
- *search*

## S

## N



## P



## R



## S



## T

## V



## Z