Дано следующее задание:

‘’

*Реализовать по методологии TDD (Test-Driven Development) консольное приложение, предоставляющее возможность отправлять письма на указанный почтовый ящик сервиса rambler.ru. Обеспечить подгрузку содержания и темы писем из файла spam\_content.txt. В файле должно содержаться не менее 50 записей по образцу:*

*ТЕМА\_ПИСЬМА – СОДЕРЖАНИЕ (кириллица и латиница, формат UTF-8, всё в одной строке).*

*Обеспечить возможность покрытия 100% кода юнит-тестами. Предусмотреть все возможные исключительные ситуации (слишком большое время отклика, обрыв соединения, некорректный формат письма и т.д.). Предоставить возможность протоколирования (в свободной форме) каждого события в программе (отправка письма, получение ответа от сервера и т.д.) в файле log.txt. По результатам всех тестов сформировать отчёт в файле report.txt следующего формата:*

*(+) CHECK TIMEOUT – PASS*

*(-) CHECK FOR RESPONSE 200 – FAIL*

*…*

*‘’*

**Разбор задания**

В первую очередь, разберём данную постановку на требования:

1. Cчитать с текстового файла как минимум 50 строк определённого формата.
2. Сформировать согласно протоколу SMTP сообщение, которое возможно отправить на почтовый ящик
3. Отправить письма на указанный адрес электронной почты

**ЗАМЕЧАНИЕ:** вышеперечисленные требования относятся к т.н. “бизнес-логике” приложения. требования из разряда контроля работы протокола, проверки работы считывания из текстового файла (исключая формат), как правило, юнит-тестами не закрываются. Для контроля этих функций существуют специально определённые исключения.

В методологии TDD юнит-тесты УЖЕ являются исчёрпывающей и актуальной документацией на продукт, т. к. любое отклонение от этих требований является дефектом. Автоматизированное тестирование не исключает процедур, предусмотренных в ручном. Так и поступим: на каждое требование напишем чек-лист, в котором будут отражаться как позитивные, так и негативные проверки, сформируем тест-кейсы и запрограммируем тест-кейсы. И только после начинаем писать код, реализующий эти требования.

100% покрытие кода юнит-тестами в таких случаях не является обязательным условием, т.к. нам достаточно закрыть функциональные требования, чтобы убедиться, что требования действительно реализованы. Что касается проектироевания – для комфортного написания и сопровождения тестовой инфраструктуры нужно изолировать используемые алгоритмы, обеспечить разбиение приложения на слабо связанные модули.

**Примерный перечень модулей**

* приёмник из файла spam\_content;
* модуль для работы с протоколом smtp;
* логгер для записи в лог-файл;
* модуль, хранящий информацию об очереди сообщений;
* модуль, формирующий отчёты;
* тестовый модуль: приёмника, очереди, отправителя на ящик.

**Чек-лист**

Приёмник

* считанных сообщений 50;
* латиница воспринимается корректно;
* кириллица воспринимается корректно;
* смесь латиницы и кириллицы воспринимается корректно;
* записанных сообщений 50;
* в очереди сообщений нет пустых элементов;
* неверный формат файла;
* неверный формат сообщений;
* очередь сообщений пуста;
* в очереди присутствуют пустые элементы;
* латиница воспринимается некорректно;
* кириллица воспринимается некорректно;
* смесь латиницы и кириллицы воспринимается некорректно.

Очередь сообщений

* в очереди 50 элементов;
* латиница отображается коррректно;
* кириллица отображается корректно;
* смесь латиницы и кириллицы отображается корректно;
* элементы в очереди верного формата;
* пустые элементы в очереди;
* смесь латиницы и кириллицы отображается некорректно;
* кириллица отображается некорректно;
* латиница отображается некорректно;
* неверный формат;
* очередь пуста.

Отправитель

* отправитель принял 50 элементов очереди сообщений;
* отправлено 50 сообщений;
* неверный адрес;
* неверный хост;
* etc.

Каждый из элементов этого списка является юнит-тестом.

**Пример работы библиотеки unittest (python 3.х)**

Библиотека семейства unittest (бывший pyunit) (аналог в С++ - Сunit, Java – Junit, TestNG, C# - Nunit)

**import requests # библиотека для работы с http**

**class APIMockup**(object):

**def** \_\_init\_\_(self, host):

self.host = host

self.session = requests.session()

**def register\_user**(self, data):

resp = self.session.post(self.host+'/user', data=data)

**return** resp

**def approve\_user**(self, data):

resp = self.session.post(self.host + '/user/approve', data=data)

**return** resp

**def login\_user**(self, data):

resp = self.session.post(self.host+'/user/login', data=data)

**return** resp

**def get\_user\_by\_id**(self, id, data):

resp = self.session.get(self.host+'/user/'+str(id), data=data)

**return** resp

**def get\_user\_by\_token**(self, token):

resp = self.session.get(self.host+'/user', cookies=token)

**return** resp

Код, приведённый выше – часть макета (mock-object) тестируемого кода. Продемонстрируем работу библиотеки:

**import unittest**

**from** api **import** APIMockup

**from** data.user.register\_data **import** \*

**from** data.user.login\_data **import** \*

**class APITester**(unittest.TestCase):

**def setUp**(self):

self.api\_object = APIMockup('http://178.172.161.36')

self.response = **None**

''' Register Tests '''

**def test\_register\_positive**(self):

self.response = self.api\_object.register\_user(ok\_registration).json()

self.assertTrue(self.response['user']['userkey'] **is not None**)

**def test\_register\_duplicate**(self):

self.response = self.api\_object.register\_user(double\_registration).json()

self.assertEqual(self.response['error']['code'], 10050)

**def test\_register\_wrong\_number\_format**(self):

self.response = self.api\_object.register\_user(wrong\_number\_format\_regitration).json()

self.assertEqual(self.response['error']['code'], 0)

''' Authorize Tests '''

**def test\_approve\_user\_positive**(self):

response\_register = self.api\_object.register\_user(ok\_registration).json()

positive\_phone = response\_register['user']['phone']

positive\_userkey = response\_register['user']['userkey']

data = '{"phone": "'+str(positive\_phone)+'", "userkey": "'+positive\_userkey+'"}'

self.response = self.api\_object.approve\_user(data).json()

token = self.response['token']

self.assertEqual(token, self.response['token'])

**def test\_approve\_not\_exists\_user**(self):

data = '{"phone": "+0000000000000", "userkey": "123456"}'

self.response = self.api\_object.approve\_user(data).json()

self.assertEqual(self.response['error']['code'], 10053)

''' Login Test '''

**def test\_login\_positive**(self):

response\_register = self.api\_object.register\_user(ok\_registration).json()

positive\_phone = response\_register['user']['phone']

positive\_userkey = response\_register['user']['userkey']

data = '{"phone": "' + str(positive\_phone) + '", "userkey": "' + positive\_userkey + '"}'

response\_approve = self.api\_object.approve\_user(data).json()

self.response = self.api\_object.login\_user(data).json()

self.assertEqual(self.response['user']['phone'], positive\_phone)

**def test\_login\_number\_not\_exists**(self):

self.response = self.api\_object.login\_user(not\_exists\_user\_login).json()

self.assertTrue(self.response['error']['code'], 10053)

**def test\_login\_wrong\_number\_format**(self):

self.response = self.api\_object.login\_user(not\_exists\_user\_login).json()

self.assertTrue(self.response['error']['code'], 10053)

**def test\_login\_without\_userkey**(self):

self.response = self.api\_object.login\_user(empty\_userkey\_login).json()

self.assertEqual(self.response['error']['code'], 10052)

**def test\_login\_without\_number**(self):

self.response = self.api\_object.login\_user(empty\_number\_login).json()

self.assertEqual(self.response['error']['code'], 0)