

**ПЛАТФОРМА ДЛЯ ФИНАНСИРОВАНИЯ ПРОЕКТОВ «CLOSERTOTHESKY»**

**Руководство программиста**

**Дата введения: 25 сентября 2018 г.**

**СОДЕРЖАНИЕ**

[1.](#_gjdgxs) ОБЩИЕ СВЕДЕНИЯ 3

[2.](#_30j0zll) НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ ПРОГРАММЫ 3

[3.](#_3znysh7) ХАРАКТЕРИСТИКИ ПРОГРАММЫ 4

[4.](#_2et92p0) ОБРАЩЕНИЕ К ПРОГРАММЕ 5

[5.](#_tyjcwt) ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ (Рис. 3) 6

[6.](#_3dy6vkm) СООБЩЕНИЯ 6

[Приложение 1. Пример файла сервера 9](#_4d34og8)

[Приложение 2. Пример файла клиента 12](#_2s8eyo1)

# Общие сведения

Программа является средством для помощи и существенного упрощения взаимодействия между разработчиками и пользователями платформы. Платформа на основе программы осуществляет возможность поддержки начинающих разработчиков программного обеспечения, оповещает поддержавших проект людей об этапах разработки, об окончании каждого из этих этапов, а также окончании полного цикла разработки.

Цель – программное обеспечение (далее – ПО) необходимо для помощи начинающим разработчикам программного обеспечения, которое даст им возможность продемонстрировать все свои умения и идеи. Планируется разработка полноценной системы для данного ПО.

Программа написана на языке С++, Python 3.

# Назначение и условия применения программы

Программный комплекс предназначен для реализации следующих задач:

* публикация проектов для финансирования;
* отслеживание и обработка поступающих на счет проекта средств;
* отслеживание активности пользователей платформы;
* обеспечение удобного и быстрого доступа к проектам;
* обеспечение безопасности персональных данных как пользователей, так и разработчиков.

Основными функциями, выполняемыми программным комплексом, являются:

* публикация проектов для финансирования;
* дополнение проектов разработчиками при надобности;
* отслеживание и обработка поступающих на счет проекта средств;
* отслеживание активности пользователей платформы;
* обеспечение удобного и быстрого доступа к проектам;
* обеспечение безопасности персональных данных как пользователей, так и разработчиков.

Для нормальной работы как серверной, так и клиентской частей необходимо – для ПК:

* компьютер с процессором Intel Pentium Dual Core и выше;
* оперативная память не менее 1 Gb;
* свободное место на жестком диске объемом не менее 5 Gb;
* наличие адаптера подключения к сети;
* установленная операционная система (далее – ОС) Windows 7/8/10.

Для сервера (Рис. 1).

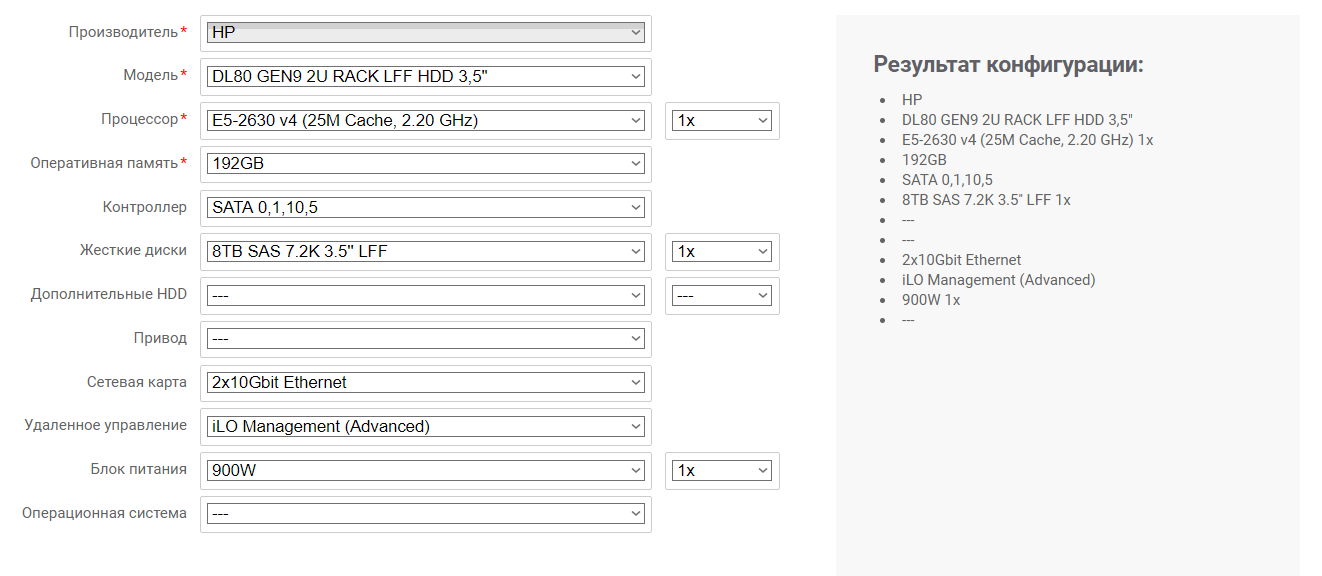


Рис. 1. Характеристики сервера

# Характеристики программы

**3.1. Режим работы программы**

Система эксплуатируется в круглосуточном режиме с учетом технологических профилактических перерывов и перерывов на проведение регламентных работ в течение не более 2-х часов в сутки с 24.00 до 4.00 по московскому времени.

Развитие Системы производится без нарушения ее работоспособности. Регламентные работы, требующие остановки системы, могут проводиться не более 2 раз в месяц и не дольше 4 часов.

* 1. **Скорость программы**

Скорость программы зависит от количества проектов, одновременно отображающихся на экране.

# Обращение к программе

Для запуска системы CLOSERTOTHESKY запустить программу на компьютере. Тем самым будет произведен запуск системы CLOSERTOTHESKY.

После запуска системы CLOSERTOTHESKY на экране появится окно запуска программы (Рис. 2).

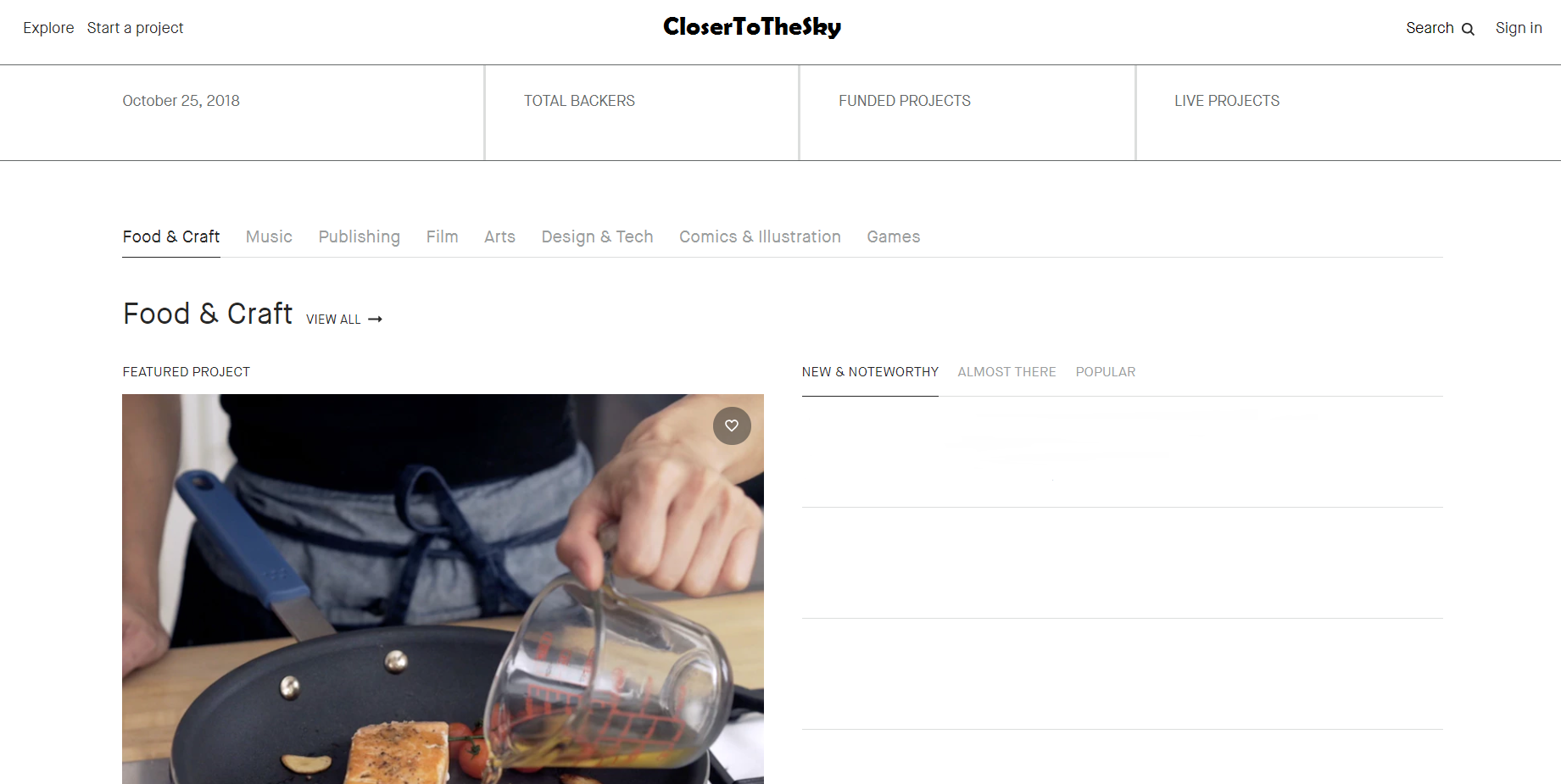


Рис. 2. Окно загрузки программы CLOSERTOTHESKY

Завершение работы программы производится закрытием окон. Закрытие окна одного из режимов не повлияет на работу подсистемы в другом режиме.

# Входные и выходные данные (Рис. 3)



Рис. 3. Входные и выходные данные

# сообщения

Ошибка соединения с сервером:

* проверить соединение с интернетом;
* проверить работу сервера;
* обратиться к системному администратору;
* перезапустить сервер.

Ошибки загрузки программы:

* проверить совместимость программы с системой;
* проверить, правильно ли установлено программное обеспечение;
* проверить, есть ли доступ в интернет;
* переустановить программу, соблюдая все правила.

Ошибка авторизации:

* проверить, правильно ли введены данные;
* проверить, есть ли доступ в интернет;
* обратиться в тех. поддержку.

Ошибка регистрации проекта:

* проверить наличие соединения с интернетом;
* проверить не нарушает ли ваш проект правил Платформы;
* обратиться к администратору.

Ошибка краудфандинговой системы:

* проверить, есть ли доступ в интернет;
* проверить, правильны ли введенные пользователем данные;
* проверить доступность платежей в системе.

**7. ПРИМЕРЫ ОШИБОК**

* + - Ошибка соединения с сервером (Рис. 4);

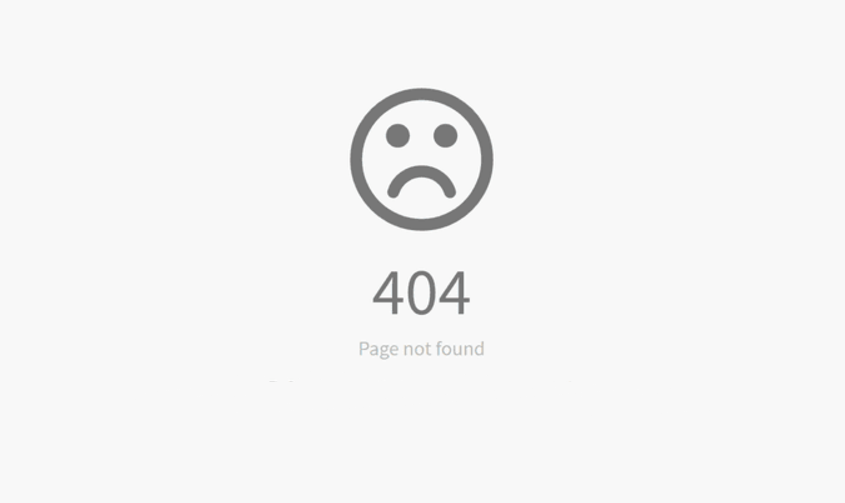


Рис. 4. Ошибка соединения с сервером

* + - ошибка авторизации (Рис. 5).

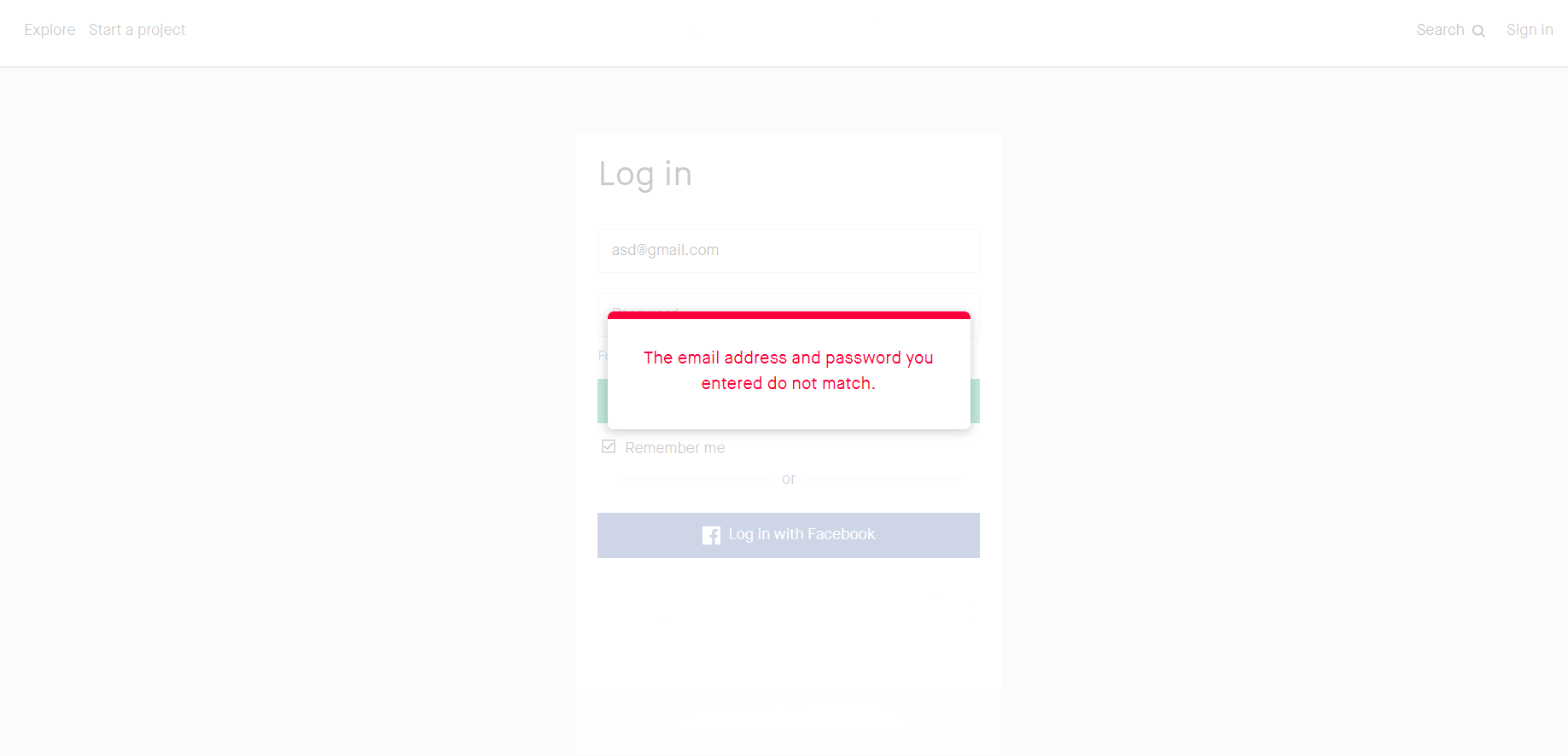


Рис. 5. Ошибка авторизации**Приложение 1. Пример файла сервера**

#include "server.hpp"

#include <boost/asio/write.hpp>

#include "yield.hpp"

namespace awesome {

using boost::asio::ip::tcp;

server::server(

const std::string& listen\_address, const std::string& listen\_port,

const std::string& target\_address, const std::string& target\_port)

: io\_service\_(std::make\_shared<boost::asio::io\_service>()),

resolver\_(std::make\_shared<tcp::resolver>(\*io\_service\_)),

acceptor\_(std::make\_shared<tcp::acceptor>(\*io\_service\_,

\*resolver\_->resolve({listen\_address, listen\_port}))),

up\_endpoint\_(\*resolver\_->resolve({target\_address, target\_port})),

allocator\_(std::make\_shared<allocator>())

{

}

server::server(const server& other)

: coroutine(other),

io\_service\_(other.io\_service\_),

resolver\_(other.resolver\_),

acceptor\_(other.acceptor\_),

up\_endpoint\_(other.up\_endpoint\_),

socket1\_(other.socket1\_),

socket2\_(other.socket2\_),

buffer\_(other.buffer\_),

allocator\_(other.allocator\_)

{

std::cout << "COPY" << std::endl;

}

void server::operator()(boost::system::error\_code ec, std::size\_t length)

{

reenter (this)

{

fork server(\*this)();

if (is\_parent())

{

io\_service\_->run();

return;

}

do

{

socket1\_ = std::make\_shared<tcp::socket>(\*io\_service\_);

yield acceptor\_->async\_accept(\*socket1\_, std::move(\*this));

if (!ec) fork server(\*this)();

} while (is\_parent());

std::cout << "connection::start()" << std::endl;

allocator\_ = std::make\_shared<allocator>();

socket2\_ = std::make\_shared<tcp::socket>(\*io\_service\_);

yield socket2\_->async\_connect(up\_endpoint\_, std::move(\*this));

if (!ec)

{

fork server(\*this)();

buffer\_ = std::make\_shared<std::array<unsigned char, 1024>>();

if (is\_child())

{

allocator\_ = std::make\_shared<allocator>();

std::swap(socket1\_, socket2\_);

}

while (!ec)

{

yield socket1\_->async\_read\_some(

boost::asio::buffer(\*buffer\_), std::move(\*this));

if (!ec)

{

yield boost::asio::async\_write(\*socket2\_,

boost::asio::buffer(\*buffer\_, length), std::move(\*this));

}

}

}

if (socket1\_->is\_open() || socket2\_->is\_open())

{

std::cout << "connection::stop()" << std::endl;

socket1\_->close(ec);

socket2\_->close(ec);

}

}

}

}

#include "unyield.hpp"

**Приложение 2. Пример файла клиента**

import argparse

import datetime

import random

import time

from influxdb import InfluxDBClient

from influxdb.client import InfluxDBClientError

USER = 'root'

PASSWORD = 'root'

DBNAME = 'tutorial'

def main(host='localhost', port=8086, nb\_day=15):

"""Instantiate a connection to the backend."""

nb\_day = 15 # number of day to generate time series

timeinterval\_min = 5 # create an event every x minutes

total\_minutes = 1440 \* nb\_day

total\_records = int(total\_minutes / timeinterval\_min)

now = datetime.datetime.today()

metric = "server\_data.cpu\_idle"

series = []

for i in range(0, total\_records):

past\_date = now - datetime.timedelta(minutes=i \* timeinterval\_min)

value = random.randint(0, 200)

hostName = "server-%d" % random.randint(1, 5)

# pointValues = [int(past\_date.strftime('%s')), value, hostName]

pointValues = {

"time": int(past\_date.strftime('%s')),

"measurement": metric,

"fields": {

"value": value,

},

"tags": {

"hostName": hostName,

},

}

series.append(pointValues)

print(series)

client = InfluxDBClient(host, port, USER, PASSWORD, DBNAME)

print("Create database: " + DBNAME)

try:

client.create\_database(DBNAME)

except InfluxDBClientError:

# Drop and create

client.drop\_database(DBNAME)

client.create\_database(DBNAME)

print("Create a retention policy")

retention\_policy = 'server\_data'

client.create\_retention\_policy(retention\_policy, '3d', 3, default=True)

print("Write points #: {0}".format(total\_records))

client.write\_points(series, retention\_policy=retention\_policy)

time.sleep(2)

query = "SELECT MEAN(value) FROM {} WHERE \

time > now() - 10d GROUP BY time(500m)".format(metric)

result = client.query(query, database=DBNAME)

print(result)

print("Result: {0}".format(result))

print("Drop database: {}".format(DBNAME))

client.drop\_database(DBNAME)

def parse\_args():

"""Parse the args."""

parser = argparse.ArgumentParser(

description='example code to play with InfluxDB')

parser.add\_argument('--host', type=str, required=False,

default='localhost',

help='hostname influxdb http API')

parser.add\_argument('--port', type=int, required=False, default=8086,

help='port influxdb http API')

parser.add\_argument('--nb\_day', type=int, required=False, default=15,

help='number of days to generate time series data')

return parser.parse\_args()

if \_\_name\_\_ == '\_\_main\_\_':

args = parse\_args()

main(host=args.host, port=args.port, nb\_day=args.nb\_day)