Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

**Отчет по лабораторной работе**

**по курсу «Тестирование и отладка ПО»**

Студенты: Марабян К.В.

Селивёрстов И.А.

Маркарьян К.Х.

Группа: ИУ7-75

Преподаватель: Рогозин Н.О.

Москва, 2019

**Цель лабораторной работы**

Написать спецификацию программного продукта и протестировать его с помощью модульных, интеграционных, регрессионных, функциональных и системных тестов.

**Задача**

Целью работы является реализация веб-приложения для моделирования работы сайта библиотеки. Необходимо обеспечить хранение данных о книгах, авторах и пользователях. Подключение новых пользователей происходит с помощью регистрации – создания новой записи в базе данных. После аутентификации пользователь может воспользоваться всем функционалом сайта.

Данное WEB-приложение написано на языке Python с использованием фреймворка Django, который основан на концепции MVC(Model-View-Controller). Также была использована СУБД SQLite.

**1 Модульное тестирование**

Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны. С помощью модульных тестов мы протестировали основные компоненты нашего приложения.

Модульное тестирование производилось на методом registration()

Соответствующие классы эквивалентности registration():

Добавление пользователя:

- добавить нового пользователя с не существующими в базе данных email и login

- добавить пользователя с существующим логином

- добавить пользователя с существующим email

- добавить пользователя не заполнив все параметры

Пример:

|  |
| --- |
| def test\_lable\_name(self):  print("Method: test\_lable\_name.")  author = Author.objects.get(id = 1)  field\_lable = author.\_meta.get\_field('name').verbose\_name  self.assertEqual(field\_lable,'Имя автора') |

**2 Интеграционное тестирование**

Целью интеграционного тестирования является проверка соответствия проектируемых единиц функциональным, приёмным и требованиям надежности.

Пример:

|  |
| --- |
| class LibraryViewTest(TestCase):  def test\_view\_url\_exists(self):  print("Method: test\_url\_exists.")  resp = self.client.get('/library/')  self.assertEqual(resp.status\_code, 200)  def test\_view\_uses\_correct\_template(self):  print("Method: test\_view\_uses\_correct\_template.")  resp = self.client.get(reverse('main'))  self.assertEqual(resp.status\_code, 200)  self.assertTemplateUsed(resp, 'library/main.html') |

**3 Системное тестирование**

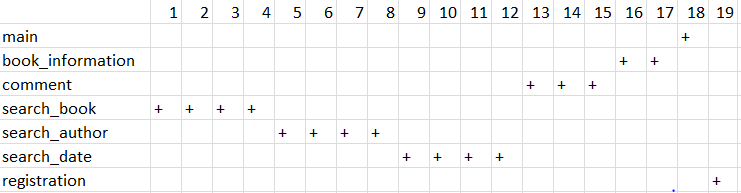
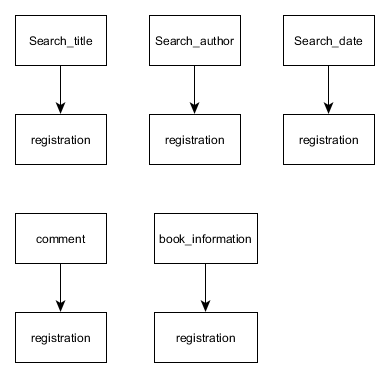
Системное тестирование программного обеспечения — это тестирование программного обеспечения (ПО), выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям. Системное тестирование относится к методам тестирования чёрного ящика, и, тем самым, не требует знаний о внутреннем устройстве системы.

Пример:

|  |
| --- |
| class LibraryTest(TestCase):  def setUp(self):  test\_user1=User.objects.create\_user(username='koi',email='koryun9999@mail.ru',password='11091998k',first\_name='Koryun',last\_name='Marabyan')  test\_user1.save()    def test\_search\_book\_redirect\_if\_not\_logged\_in(self):  resp = self.client.get('/library/search\_book/')  self.assertRedirects(resp, '/accounts/login/?next=/library/search\_book/')  def test\_book\_information\_logged\_in\_correct\_get\_request(self):  login = self.client.login(username='koi', password='11091998k')  resp = self.client.get('/library/book\_information/1/')  self.assertEqual(str(resp.context['user']), 'koi')  self.assertEqual(resp.status\_code, 200)  self.assertTemplateUsed(resp, 'library/book\_information.html') |

**4 Регрессионное тестирование**

Регрессионное тестирование — это выборочное тестирование, позволяющее убедиться, что изменения не вызвали нежелательных побочных эффектов, или что измененная система по-прежнему соответствует требованиям.



**5 Многопоточное тестирование**

Django не поддерживает несколько потоков одновременно, поэтому для поддержки многопоточности был скачен nginx. В конфигурационный файл nginx.confдобавлен следующий код:

server {  
server\_name 127.0.0.1;  
listen 8080;  
location / {  
proxy\_pass http://127.0.0.1:8000/;  
}

Для тестирования производительности сайта использовалась утилита Apache Benchmark.

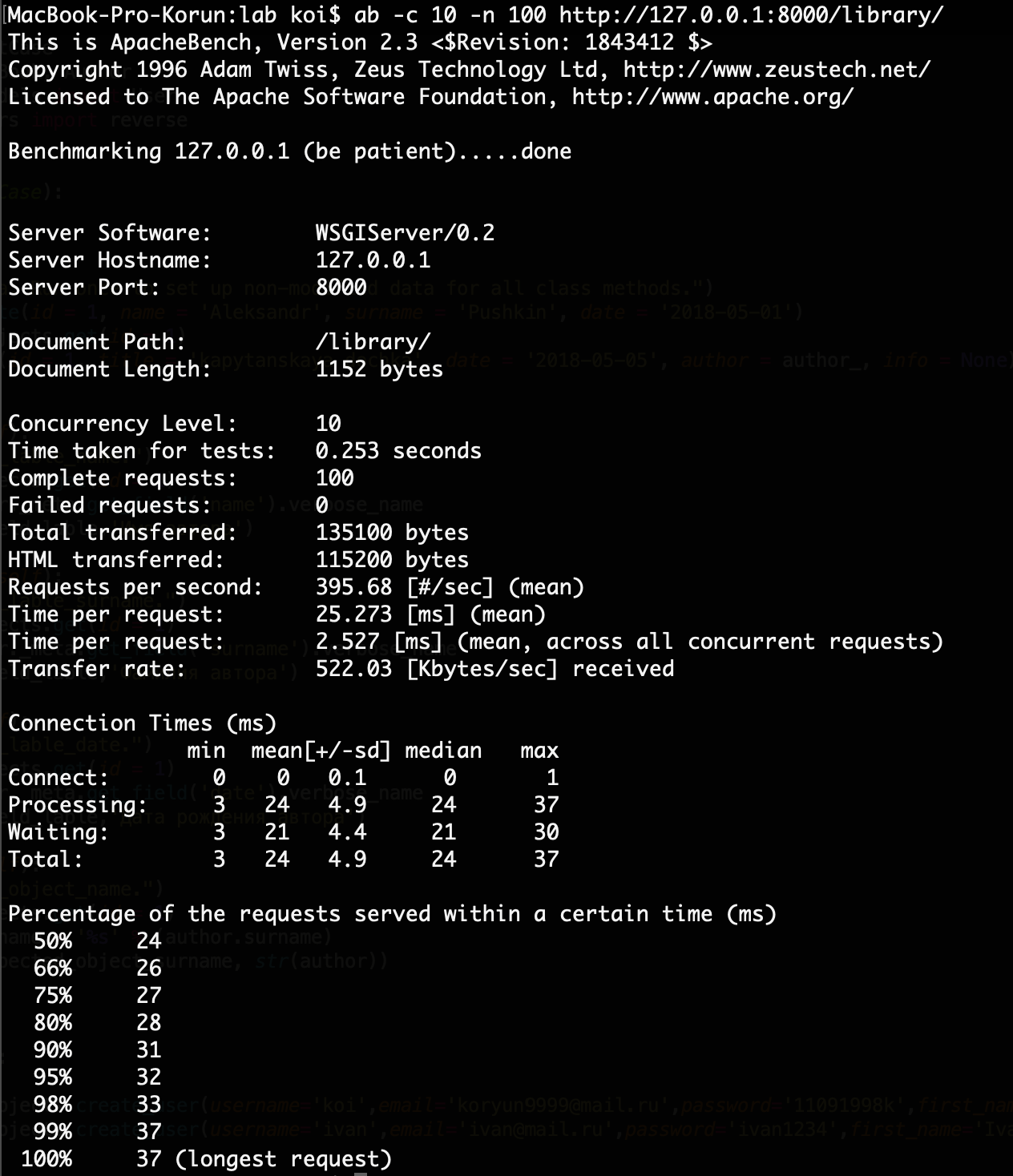
Команда:

ab –c 10 –n 100 http://127.0.0.1:8080/library/

c = 10 – количество конкурентных запросов

n = 100 – общее количество запросов

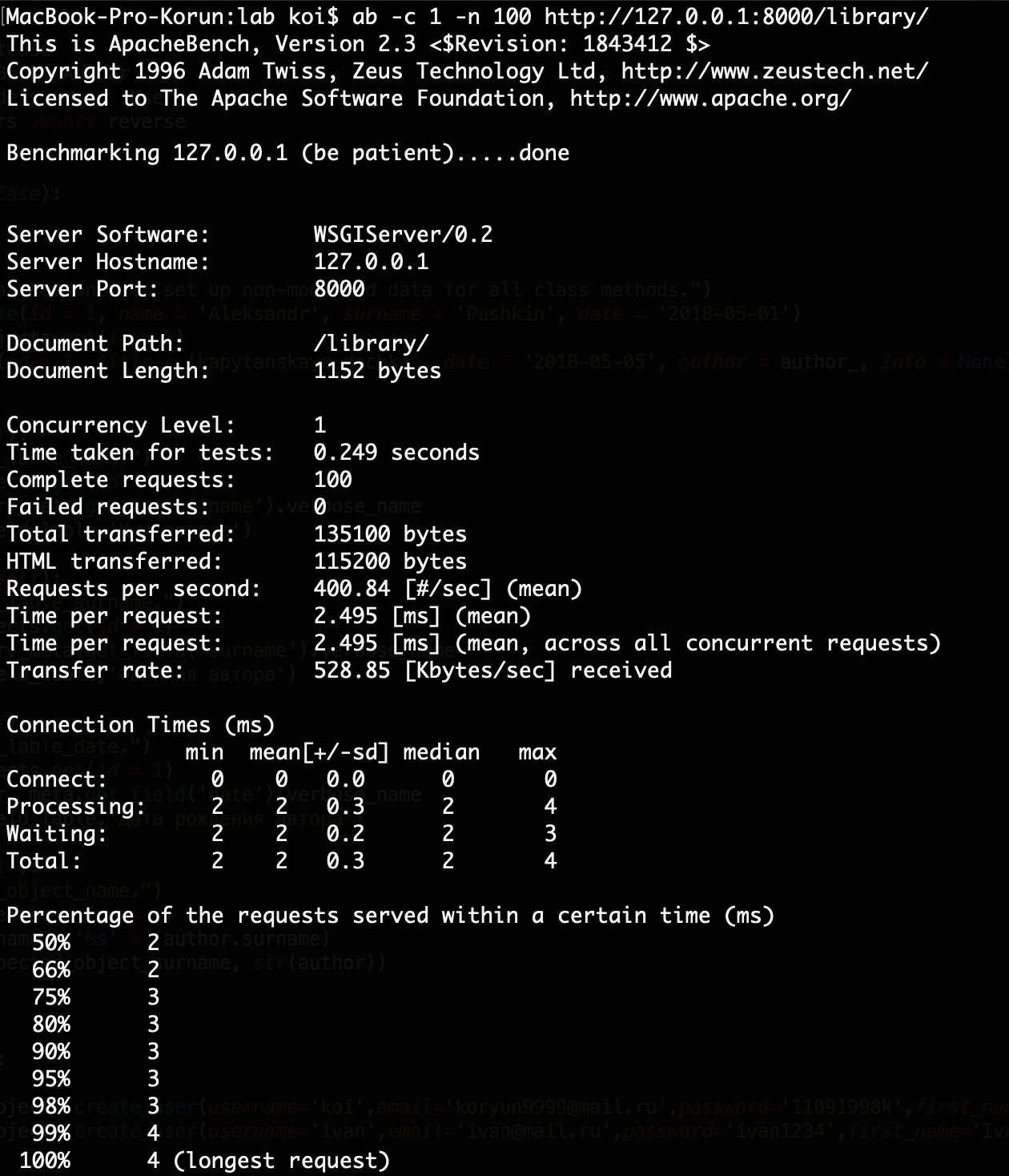
Результат:



Команда:

ab –c 1 –n 100 http://127.0.0.1:8080/library/

Результат:



**6 Автоматизированное тестирование**

Был написан скрипт для запуска тестов в соответствии с получаемым списком параметров.

