

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет безопасности информационных технологий

Дисциплина:

«Технологии и методы программирования»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

Выполнил:

Михайлик Антон Денисович, студент группы N3351



(Подпись)

Проверил:

Ищенко Алексей Петрович

(Отметка о выполнении)

(Подпись)

Санкт-Петербург

2024 г.

Содержание

Содержание	2
1 Техническое задание	3
2 Описание сервиса для тестирования	3
3 Методы тестирования	3
3.1 Тестирование API	3
3.2 Проверка на уязвимости	4
3.3 Проверка на нагрузку системы	7
4 Заключение	15
5 Приложение	16

1 Техническое задание

Ознакомиться с основными методами тестирования (черный ящик, белый ящик, тестирование на основе требований). Разработать тестовые сценарии для заданного приложения. Провести функциональное тестирование приложения. Зафиксировать результаты тестирования и выявленные дефекты. Подготовить отчет о проведенной лабораторной работе.

2 Описание сервиса для тестирования

В качестве сервиса для тестирования было выбрано приложение, находящееся по доменному имени `cryptoroll.su`. Это приложение включает в себя игру по предугадыванию цены криптовалюты (BTC, ETH, TON) через некоторое время (30м, 4ч, 12ч). При правильном угадывании юзер получает в награду игровую валюту.

Сервер включает в себя `frontend` и `api`. В данной лабораторной работе тестируется и `api` и `frontend` различными способами.

Сервер с приложением находится на виртуальной машине, установленной поверх гипервизора `hyperV` на домашнем сервере, что позволяет гибко менять настройки и мониторить систему.

3 Методы тестирования

В качестве методов тестирования были выбраны следующие:

- тестирование правильности работы API с помощью `pytest`
- тестирование на уязвимости с помощью `burpsuite`
- тестирование на нагрузку, с помощью встроенной утилиты в OS Linux и различных других способов, позволяющие получить общую картину о нагрузке

3.1 Тестирование API

Тестирование эндпоинтов выполнялось с помощью написания скрипта на `python`, используя библиотеку `pytest`.

Были выполнены стандартные запросы на каждый из эндпоинтов, чтобы проверить корректность и правильность ответа (логин, регистрация, механика игры и т.п.). Также были намеренно вставлены данные, которые должны вызывать ошибки (неправильно прохождения опроса в разделе "награды" в профиле пользователя, ввод отрицательных чисел и т.п.)

В приложении много эндпоинтов, поэтому код с тестами находится в приложении к лабораторной работе, а на рисунке 1 приведён скриншот, показывающий удачное прохождение всех тестов.

```
(.venv) inception@inception:~/ITMO/TiMP/6lab$ pytest -v
===== test session starts =====
platform linux -- Python 3.12.3, pytest-8.3.3, pluggy-1.5.0 -- /home/inception/ITMO/TiMP/6lab/.venv/bin/python3
cachedir: .pytest_cache
rootdir: /home/inception/ITMO/TiMP/6lab
collected 14 items

test_api.py::test_signUp_user PASSED [ 7%]
test_api.py::test_post_login PASSED [ 14%]
test_api.py::test_get_user_info PASSED [ 21%]
test_api.py::test_put_change_user_info PASSED [ 28%]
test_api.py::test_get_users_tasks PASSED [ 35%]
test_api.py::test_post_change_status_of_tasks PASSED [ 42%]
test_api.py::test_get_live_price PASSED [ 50%]
test_api.py::test_post_make_prediction PASSED [ 57%]
test_api.py::test_get_match_history PASSED [ 64%]
test_api.py::test_get_reward_status PASSED [ 71%]
test_api.py::test_collect_daily_reward PASSED [ 78%]
test_api.py::test_get_user_referral_link PASSED [ 85%]
test_api.py::test_visit_referral_link PASSED [ 92%]
test_api.py::test_post_check_quiz_result PASSED [100%]

===== 14 passed in 4.78s =====
(.venv) inception@inception:~/ITMO/TiMP/6lab$
```

Рис. 1: Удачное прохождение всех тестов

3.2 Проверка на уязвимости

Для проверки на уязвимости будет использоваться BurpSuitePro. Это приложение поможет перехватывать пакеты и изменять их на ходу.

В ходе исследования работы веб приложения cryptoroll.su, была выявлена одна неточность (баг) в его работе. По сценарию игры пользователь вводит сумму, которую он хочет поставить и дальше выбирает один из трёх вариантов времени, через которое он думает цена будет больше или меньше нынешней. Всего существует три варианта: 30 минут, 4 часа, 12 часов. Но программист, который разрабатывал приложение, для своего удобства, добавил скрытый для пользователя промежуток времени, который равняется 15 секундам. Следующие действия покажут как этого можно добиться.

Открываем сайт и BurpSuitePro и убеждаемся, что мы корректно видим все запросы.

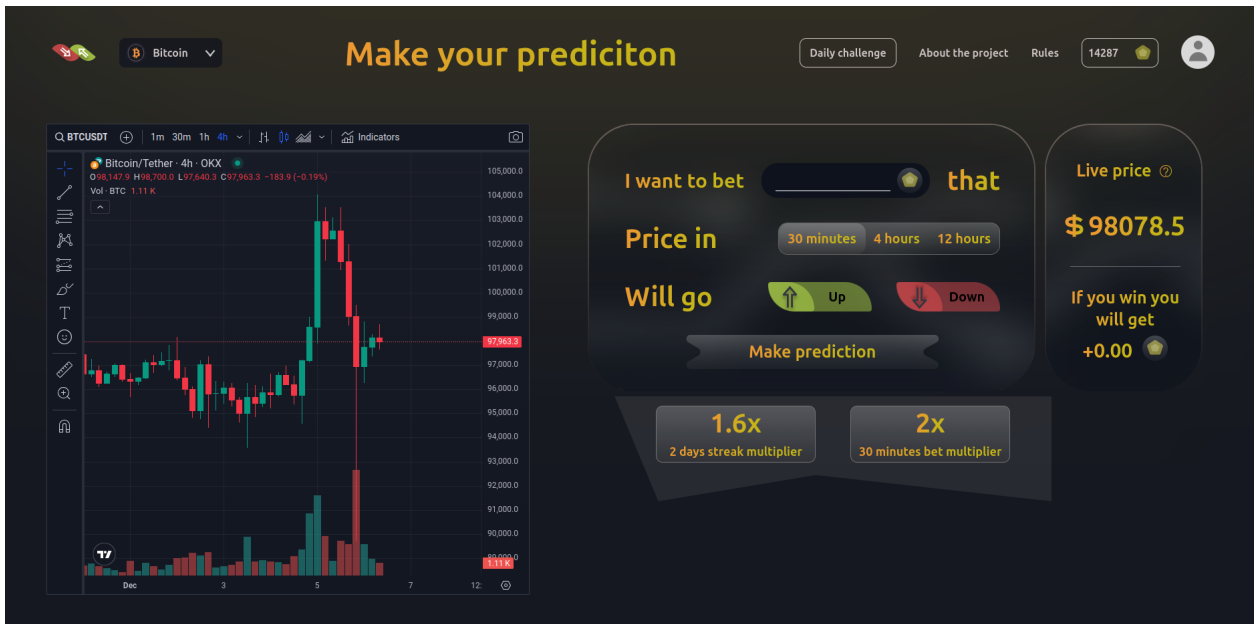


Рис. 2: Открываем cryptoroll.ru

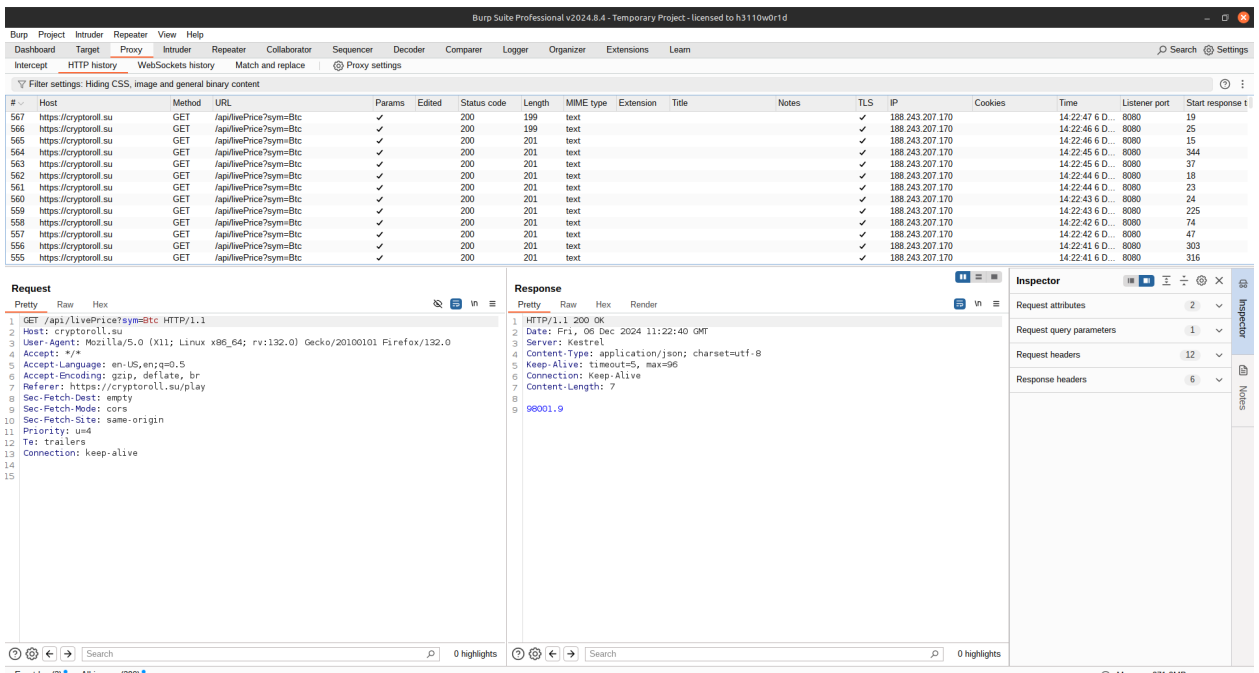


Рис. 3: Открываем BurpSuitePro



Рис. 6: Видим, что время, через которое должен выдаться результат меньше 15 секунд

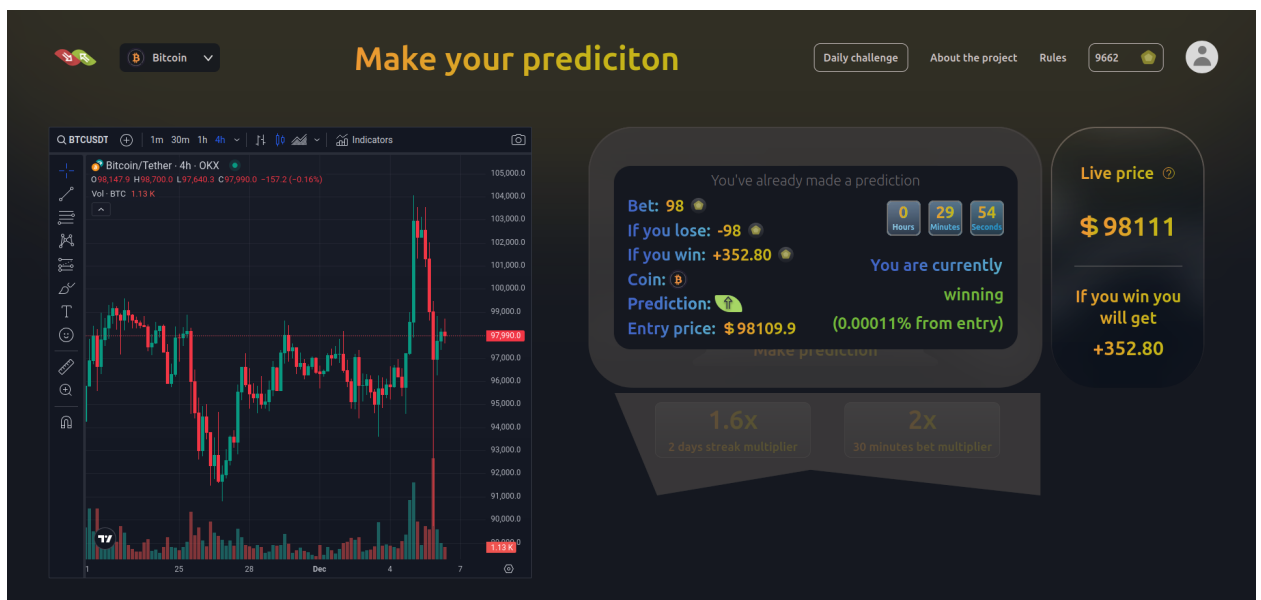


Рис. 7: Для сравнения пример нормально работающего приложения

3.3 Проверка на нагрузку системы

Проверка нагрузки через троттлинг

Через f12 в любом браузере, перейдя во вкладку Network, мы можем в самом низу увидеть сколько времени нужно было, чтобы полностью загрузить сайт. (все дальнейшие тесты будут производиться нажимая сочетание клавиш **ctrl+f5**, чтобы стирать весь кеш и имитировать первый заход пользователя)

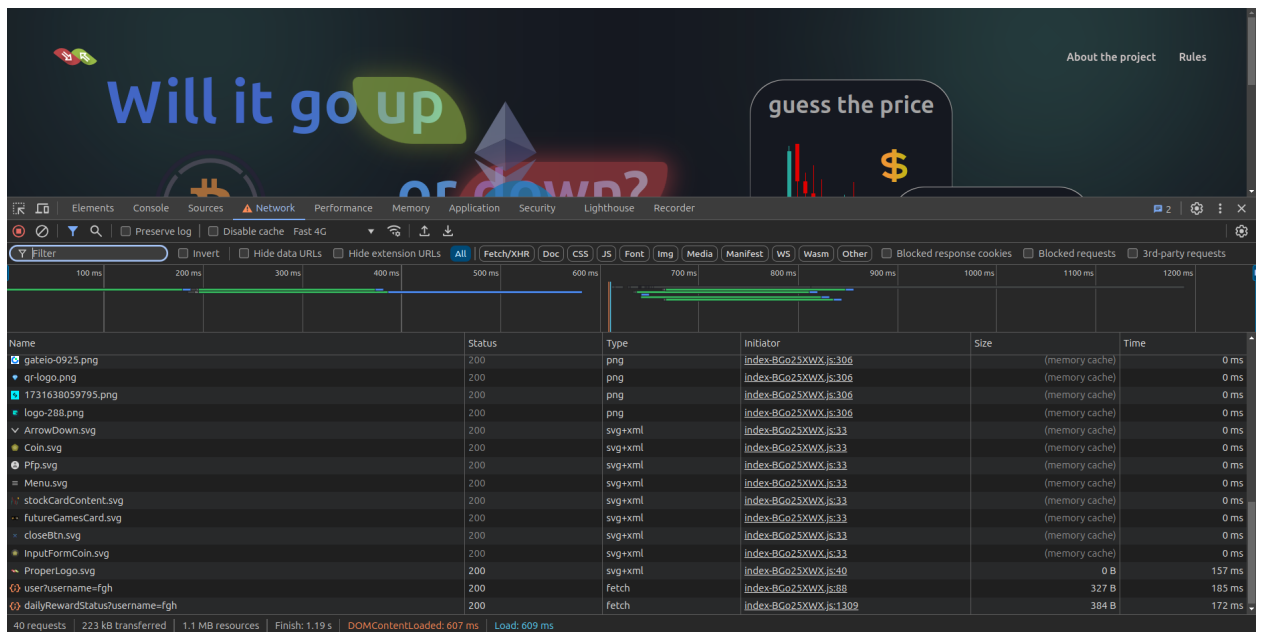


Рис. 8: load без троттлинга 609 мс

Выглядит всё очень даже хорошо, но теперь добавим троттлинг "slow 4g и посмотрим на результат.

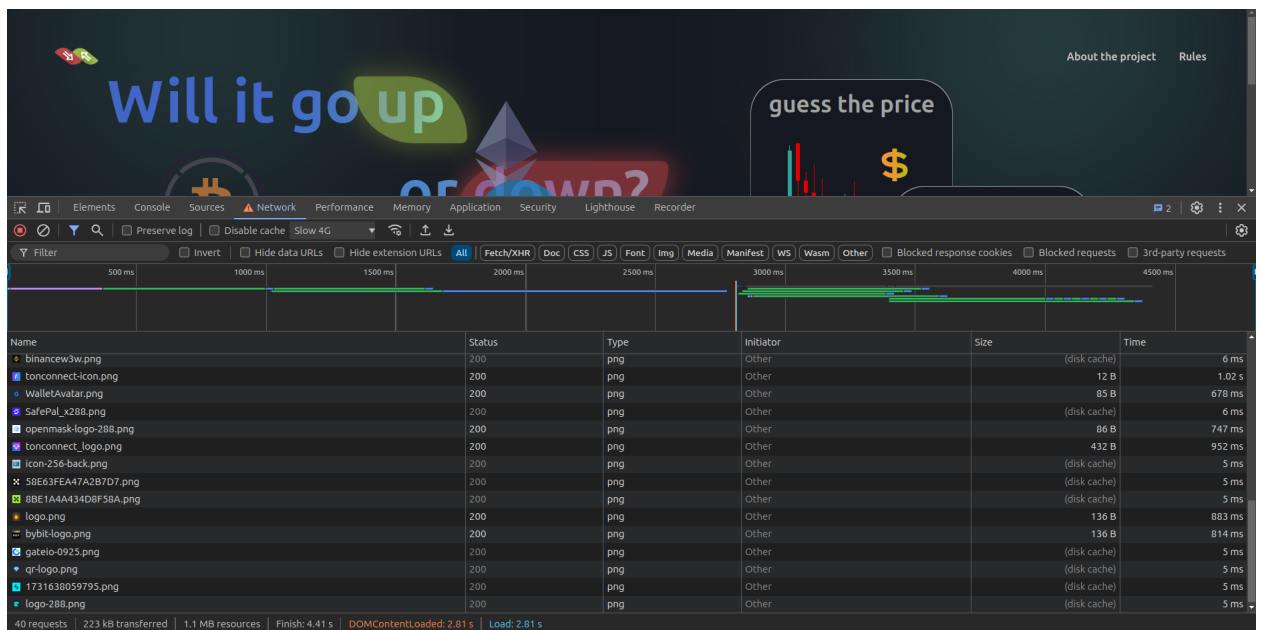


Рис. 9: load slow 4g 2.81 с

Видно, что даже небольшое замедление интернета у пользователя, и время загрузки увеличивается аж до 2.81 секунд (приличное время загрузки в бизнесе считается до трёх секунд).

Теперь увеличим троттлинг до 3g.

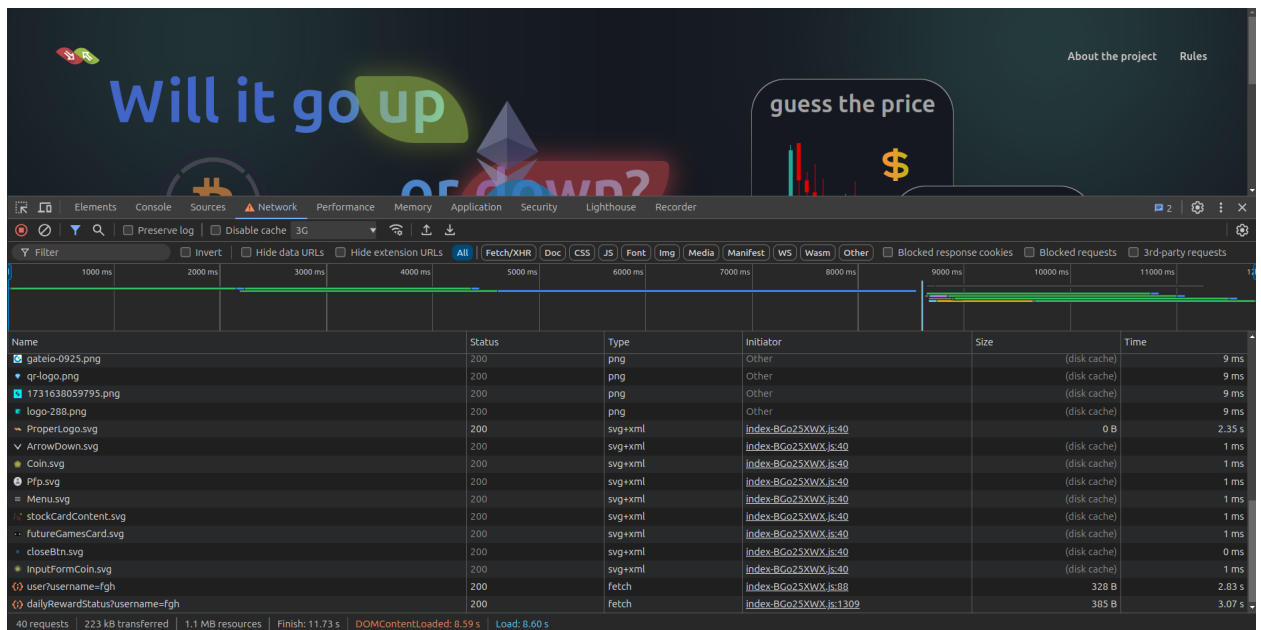


Рис. 10: load 3g 8.60 с

Видим, что теперь загрузка увеличилась до 8.60 секунд, что очень много, а это скорость без нагрузки в виде многих пользователей. Теперь выясним как мы можем сразу понять в чём проблема.

Тестирование с помощью сайта для проверки нагрузки

Воспользуемся сервисом pagespeed.web.dev.

На сайте мы можем посмотреть различную статистику, включая первую загрузку какой-либо отрисовки, производительность на телефоне и компьютере и т.п. Но самое главное, что мы можем понять, что мы можем сделать для улучшения скорости загрузки сайта.

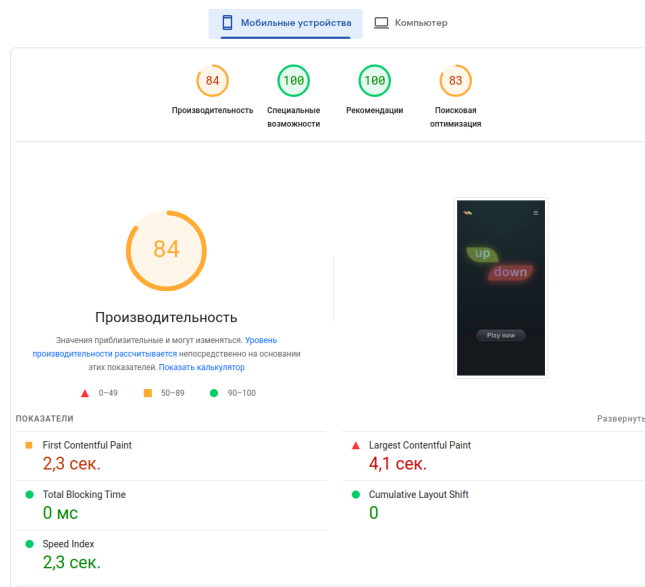


Рис. 11: Производительность на телефоне

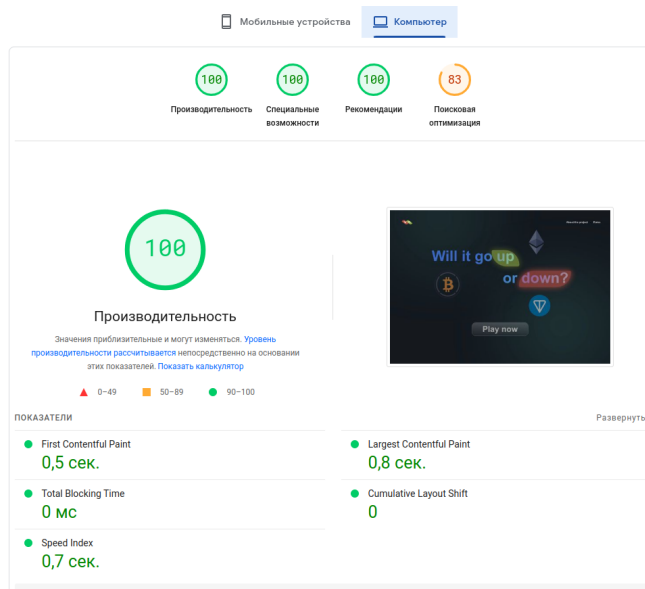


Рис. 12: Производительность на компьютере

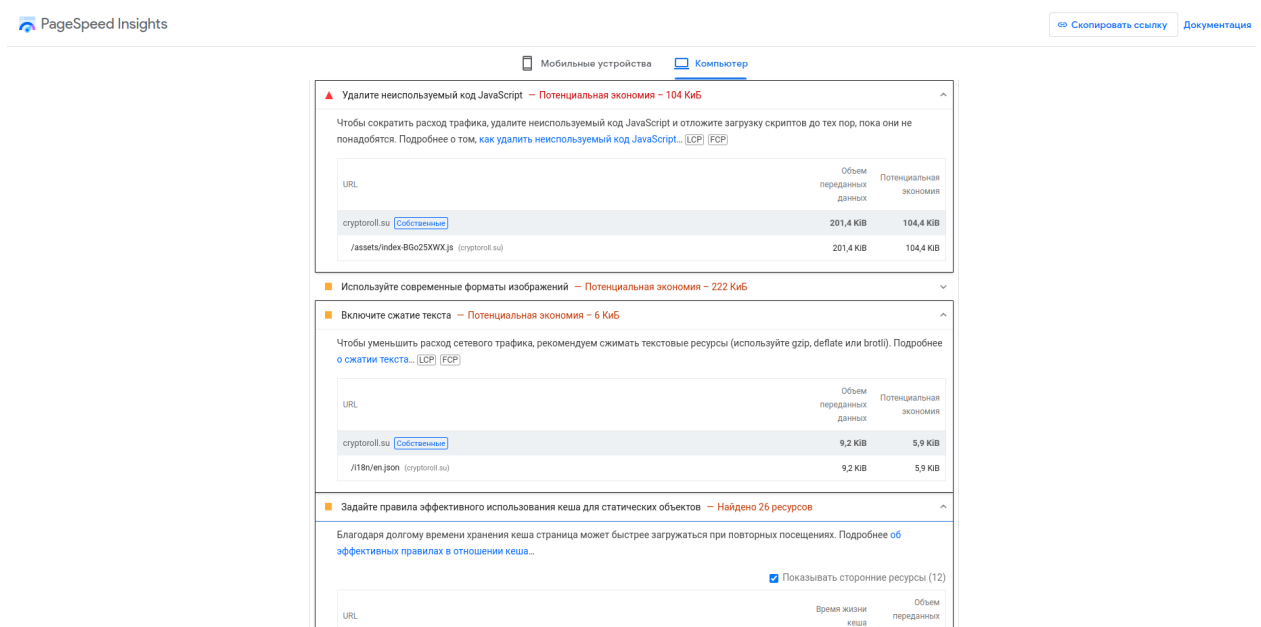


Рис. 13: Некоторые советы по улучшению скорости загрузки

Тестирование нагрузки для нескольких пользователей

Теперь протестируем работу приложения под нагрузкой. Воспользуемся встроенной Linux утилитой Apache benchmark. Она позволяет отправлять несколько запросов на ресурс многопоточно, что позволяет оценить насколько хорошо сервер справляется с нагрузкой.

Протестируем лендинговую страницу cryptoroll.ru, подав на вход в утилиту 600 запросов с многопоточностью в 200 запросов.

```
(.venv) inception@inception:~/ITMO/TiMP/6lab$ ab -n 600 -c 200 https://cryptoroll.su/
This is ApacheBench, Version 2.3 <$Revision: 1903618 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking cryptoroll.su (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Finished 600 requests

Server Software:      nginx/1.23.4
Server Hostname:      cryptoroll.su
Server Port:          443
SSL/TLS Protocol:     TLSv1.3,TLS_AES_256_GCM_SHA384,4096,256
Server Temp Key:       X25519 253 bits
TLS Server Name:       cryptoroll.su

Document Path:        /
Document Length:       548 bytes

Concurrency Level:     200
Time taken for tests:   4.489 seconds
Complete requests:      600
Failed requests:         0
Total transferred:      482400 bytes
HTML transferred:       328800 bytes
Requests per second:    133.66 [#/sec] (mean)
Time per request:       1496.364 [ms] (mean)
Time per request:       7.482 [ms] (mean, across all concurrent requests)
Transfer rate:          104.94 [Kbytes/sec] received

Connection Times (ms)
  min   mean[+/-sd] median   max
Connect:  53   722 331.5   749   1914
Processing:  58   338 309.9   259   2615
Waiting:    38   333 308.1   256   2614
Total:     111 1060 488.5  1031  4383

Percentage of the requests served within a certain time (ms)
 50%    1031
 66%    1146
 75%    1218
 80%    1301
 90%    1672
 95%    1992
 98%    2560
 99%    2699
100%    4383 (longest request)
(.venv) inception@inception:~/ITMO/TiMP/6lab$
```

Рис. 14: Работа утилиты Apache Benchmark

Здесь есть несколько ключевых моментов, на которые надо обратить внимание:

1. Failed requests - количество запросов, которые не были обработаны.
2. Time per request (mean) - среднее количество, которое потребовалось на загрузку страницы.
3. Total (max) - максимальное время, которое потребовалось для загрузки страницы.

В примере выше все запросы были удачно обработаны, среднее время на загрузку страницы было примерно 1.5 секунды и максимальное время загрузки примерно 4 секунды. Все параметры выглядят в норме, кроме максимального времени загрузки, 4 секунды означает, что мы можем потерять некоторых пользователей, которые зашли на страницу.

Попробуем ещё сильнее увеличить нагрузку, подав на вход 3000 запросов с многопоточность в 600 запросов.

```

Document Path:      /
Document Length:    548 bytes

Concurrency Level:   600
Time taken for tests: 40.257 seconds
Complete requests:   3000
Failed requests:      2
  (Connect: 0, Receive: 0, Length: 2, Exceptions: 0)
Total transferred:  2410392 bytes
HTML transferred:   1642904 bytes
Requests per second: 74.52 [#/sec] (mean)
Time per request:    8051.331 [ms] (mean)
Time per request:    13.419 [ms] (mean, across all concurrent requests)
Transfer rate:       58.47 [Kbytes/sec] received

Connection Times (ms)
  min  mean[+/-sd] median   max
Connect:    0  3030 1169.2   2644  13750
Processing: 52   672  905.1    442   23061
Waiting:    21   649  691.9    438   9253
Total:      111  3702 1415.5   3325  23061

Percentage of the requests served within a certain time (ms)
 50%    3325
 66%    3768
 75%    4151
 80%    4375
 90%    5137
 95%    6106
 98%    7514
 99%    8766
100%   23061 (longest request)
(.venv) inception@inception:~/ITMO/TiMP/6lab$

```

Рис. 15: Результат после увеличения нагрузки

Мы видим, что из 3000 запросов 2 запроса были "Failed", среднее время ожидания на загрузку было 8, а максимальное 23 секунды. Это уже выглядит как очень плохой результат.

При дальнейшей нагрузке количество неудачных запросов будет увеличиваться до тех пор, пока сервис не перестанет быть доступным.

"Failed requests" означает, что процессор не справляется с нагрузкой, это означает, что чтобы решить данную проблему надо выделить больше ядер для данной виртуальной машины.

Проверка доступности сервера из разных уголков мира

Есть сервисы, которые позволяют проверить сколько времени понадобится людям в различных странах, чтобы загрузить файлы сайта. Воспользуемся сервисом check-host.net/check-http

IP: 5.144.117.76 Страна: Russian Federation (Leningradskaya Oblast', Murino) Offshore Cloud Services

Инфо
Ping
HTTP
TCP-порт
UDP-порт
DNS

Кешбек 15% на **VPS Aéza**

Проверка веб-сайта **http://cryptoroll.su:80**

Постоянная ссылка на этот отчет | Поделиться на [Twitter](#)

Интерактивный терминал

Местонахождение ▾	Результат	Время	Код	IP адрес
Brazil, Sao Paulo	OK	1.142 c	200 (OK)	188.243.207.170
Bulgaria, Sofia	OK	0.379 c	200 (OK)	188.243.207.170
Croatia, Sisak	OK	0.099 c	200 (OK)	188.243.207.170
Czechia, C.Budejovice	OK	0.172 c	200 (OK)	188.243.207.170
Finland, Helsinki	OK	0.161 c	200 (OK)	188.243.207.170
France, Paris	OK	0.187 c	200 (OK)	188.243.207.170
Germany, Nuremberg	OK	0.131 c	200 (OK)	188.243.207.170
Hong Kong, Hong Kong	OK	1.030 c	200 (OK)	188.243.207.170
India, Chennai	OK	0.993 c	200 (OK)	188.243.207.170
India, Hyderabad	OK	1.099 c	200 (OK)	188.243.207.170
India, Mumbai	OK	0.595 c	200 (OK)	188.243.207.170
Indonesia, Jakarta	OK	0.894 c	200 (OK)	188.243.207.170
Iran, Esfahan	OK	0.564 c	200 (OK)	188.243.207.170
Iran, Karaj	OK	0.413 c	200 (OK)	188.243.207.170
Iran, Shiraz	OK	0.520 c	200 (OK)	188.243.207.170
Iran, Tehran	OK	0.344 c	200 (OK)	188.243.207.170
Israel, Netanya	OK	0.259 c	200 (OK)	188.243.207.170
Israel, Tel Aviv	OK	0.244 c	200 (OK)	188.243.207.170
Italy, Milan	OK	0.167 c	200 (OK)	188.243.207.170
Japan, Tokyo	OK	1.024 c	200 (OK)	188.243.207.170
Kazakhstan, Karaganda	OK	0.293 c	200 (OK)	188.243.207.170
Lithuania, Vilnius	OK	0.119 c	200 (OK)	188.243.207.170
Moldova, Chisinau	OK	0.166 c	200 (OK)	188.243.207.170
Netherlands, Amsterdam	OK	0.125 c	200 (OK)	188.243.207.170
Netherlands, Meppel	OK	0.129 c	200 (OK)	188.243.207.170
Poland, Poznan	OK	0.158 c	200 (OK)	188.243.207.170

Рис. 16





















 Poland, Warsaw	OK	0.235 c	200 (OK)	188.243.207.170
 Portugal, Viana	OK	0.364 c	200 (OK)	188.243.207.170
 Russia, Ekaterinburg	OK	0.175 c	200 (OK)	188.243.207.170
 Russia, Moscow	OK	0.065 c	200 (OK)	188.243.207.170
 Russia, Moscow	OK	0.105 c	200 (OK)	188.243.207.170
 Russia, Saint Petersburg	OK	0.018 c	200 (OK)	188.243.207.170
 Serbia, Belgrade	OK	0.206 c	200 (OK)	188.243.207.170
 Spain, Barcelona	OK	0.293 c	200 (OK)	188.243.207.170
 Sweden, Tallberg	OK	0.094 c	200 (OK)	188.243.207.170
 Switzerland, Zurich	OK	0.168 c	200 (OK)	188.243.207.170
 Turkey, Gebze	OK	0.174 c	200 (OK)	188.243.207.170
 Turkey, Istanbul	OK	0.368 c	200 (OK)	188.243.207.170
 UAE, Dubai	OK	0.623 c	200 (OK)	188.243.207.170
 UK, Coventry	OK	0.194 c	200 (OK)	188.243.207.170
 Ukraine, Khmelnytskyi	OK	0.268 c	200 (OK)	188.243.207.170
 Ukraine, Kyiv	OK	0.162 c	200 (OK)	188.243.207.170
 Ukraine, SpaceX Starlink	OK	0.347 c	200 (OK)	188.243.207.170
 USA, Atlanta	OK	0.516 c	200 (OK)	188.243.207.170
 USA, Chicago	OK	0.597 c	200 (OK)	188.243.207.170
 USA, Dallas	OK	0.641 c	200 (OK)	188.243.207.170

Рис. 17

Можно увидеть, что сайт хорошо доступен по всему миру. Самое долгое время загрузки потребовалось для Индии, Гонконга и Бразилии, время ответа для этих районов составило чуть больше 1 секунды.

4 Заключение

В результате лабораторной работы было проведено тестирование сервиса `cryptoroll.su`. Были получены теоретические и практические навыки способов тестирования.

Было произведено тестирования API, тестирование на уязвимость и тестирование нагрузки в различных формах. Было описано как решить и исправить выявленные недостатки.

5 Приложение

```
1 import string
2 import requests
3 import random
4
5 def randomString(length=10):
6     characters = string.ascii_letters + string.digits
7     return ''.join(random.choice(characters) for _ in range(length))
8
9 baseUrl = "https://cryptoroll.su/api"
10 jwt_token = None
11 login = randomString()
12 passwd = randomString()
13
14
15 def test_signUp_user():
16     global login
17     global passwd
18
19     body = {
20         "username": f"{login}",
21         "password": f"{passwd}"
22     }
23
24     response = requests.post(url=f"{baseUrl}/signup", json=body)
25     assert response.status_code == 201, f"Error! Status code = {response.status_code}"
26
27     dataOfResponse = response.json()
28     jwt_token = dataOfResponse.get('jwtToken')
29     assert jwt_token, "Error! JWT token was not provided"
30
31
32
33 def test_post_login():
34     global jwt_token
35     global login
36     global passwd
37
38     data={
39         "username": f"{login}",
40         "password": f"{passwd}"
41     }
42
43     response = requests.post(f"{baseUrl}/login", json=data)
44     assert response.status_code == 201, f"Error! Response code is {response.status_code}"
45     dataResponse = response.json()
46     user = dataResponse.get('user', {})
47     jwt_token = dataResponse.get('jwtToken')
48
49     assert user, "User is empty"
50     assert jwt_token, "User is empty"
51
52     name = user.get('name')
53     assert name == login
54
55 def test_get_user_info():
56     global jwt_token
```



```

57     global login
58
59     token = jwt_token
60
61     headers = {
62         'Authorization': f'Bearer {token}',
63     }
64
65     response = requests.get(f"{baseUrl}/user?username={login}", headers=
headers)
66     dataOfResponse = response.json()
67
68     assert response.status_code == 200, f"Error! Status code is {response.
status_code}"
69     assert dataOfResponse, "Error! The recieved data is empty"
70
71     username = dataOfResponse.get('username')
72     assert username == login
73
74 def test_put_change_user_info():
75     global jwt_token
76     global login
77
78     data = {
79         "name": "haha",
80         "password": f"{passwd}",
81         "walletAddress": "hehe"
82     }
83
84     token = jwt_token
85
86     headers = {
87         'Authorization': f'Bearer {token}',
88     }
89
90     response = requests.put(f"{baseUrl}/user?username={login}", json=data,
headers=headers)
91     assert response.status_code == 200, f"Error! Status code: {response.
status_code}"
92     dataOfResponse = response.json()
93
94     newToken = dataOfResponse.get("newToken")
95     assert newToken != token
96     jwt_token = newToken
97
98     data = {
99         "name": f"{login}",
100        "password": f"{passwd}",
101        "walletAddress": "hehe"
102    }
103
104    token = jwt_token
105
106    headers = {
107        'Authorization': f'Bearer {token}',
108    }
109
110    response = requests.put(f"{baseUrl}/user?username=haha", json=data,
headers=headers)
111    assert response.status_code == 200, f"Status code: {response.

```

```

112     status_code}"
113     dataOfResponse = response.json()
114
115     newToken = dataOfResponse.get("newToken")
116     assert newToken != token
117     jwt_token = newToken
118
119 def test_get_users_tasks():
120     global jwt_token
121     global login
122
123     token = jwt_token
124
125     headers = {
126         'Authorization': f'Bearer {token}',
127     }
128
129     response = requests.get(url=f"{baseUrl}/tasks?username={login}",
130 headers=headers)
131     assert response.status_code == 200, f"Error! Status code is {response.
132 status_code}"
133
134     dataOfResponse = response.json()
135     assert dataOfResponse, "No data recieved"
136
137 def test_post_change_status_of_tasks():
138     global jwt_token
139     global login
140
141     token = jwt_token
142
143     headers = {
144         'Authorization': f'Bearer {token}',
145     }
146     response = requests.get(url=f"{baseUrl}/tasks?username={login}",
147 headers=headers)
148     assert response.status_code == 200, f"Error! Status code is {response.
149 status_code}"
150
151     dataOfResponse = response.json()
152     tasksInfo = dataOfResponse.get('tasks', [])
153     statusTastOne = tasksInfo[0].get('status')
154     assert statusTastOne == "Uncompleted"
155
156     body = {
157         "taskId": 1,
158         "changedStatus": "Completed"
159     }
160
161     headers = {
162         'Authorization': f'Bearer {token}',
163     }
164
165     response = requests.post(url=f"{baseUrl}/tasks?username={login}", json
166 = body, headers=headers)
167     assert response.status_code == 200, f"Error! Status code is {response.
168 status_code}"

```

```

165     response = requests.get(url=f"{baseUrl}/tasks?username={login}",
166     headers=headers)
167     assert response.status_code == 200, f"Error! Status code is {response.
168     status_code}"
169     dataOfResponse = response.json()
170     tasksInfo = dataOfResponse.get('tasks', [])
171     statusTastOne = tasksInfo[0].get('status')
172     assert statusTastOne == "Completed"
173
174 def test_get_live_price():
175     global login
176
177     response = requests.get(f"{baseUrl}/livePrice?sym=Btc")
178     assert response.status_code == 200, f"Error! Status code is {response.
179     status_code}"
180
181     dataOfResponse = response.text
182     assert dataOfResponse, f"Error! The price is {dataOfResponse}"
183
184 def test_post_make_prediction():
185     global jwt_token
186     global login
187
188     token = jwt_token
189
190     body = {
191         "username": f"{login}",
192         "coin": "Btc",
193         "predictionAmount": 10,
194         "predictionTimeframe": "00:30:00",
195         "predictionValue": "Up"
196     }
197
198     headers = {
199         'Authorization': f'Bearer {token}',
200     }
201
202     response = requests.post(url=f"{baseUrl}/match/createMatch?username={
203     login}", json=body, headers=headers)
204
205     assert response.status_code == 204, f"Error! Status code is {response.
206     status_code}"
207
208     response = requests.post(url=f"{baseUrl}/match/createMatch?username={
209     login}", json=body, headers=headers)
210
211     assert response.status_code == 400, f"Error! Status code is {response.
212     status_code}"
213
214 def test_get_match_history():
215     global jwt_token
216     global login
217
218     token = jwt_token
219
220     headers = {
221         'Authorization': f'Bearer {token}',
222     }

```

```

218     response = requests.get(url=f"{baseUrl}/match/history?username={login
219     }&offset=0&limit=10", headers=headers)
220
221 def test_get_reward_status():
222     global jwt_token
223     global login
224
225     token = jwt_token
226
227     headers = {
228         'Authorization': f'Bearer {token}',
229     }
230
231     response = requests.get(url=f"{baseUrl}/rewards/dailyRewardStatus?
232     username={login}", headers=headers)
233     assert response.status_code == 200, f"Error! Status code is {response.
234     status_code}"
235     assert response.json()
236
237 def test_collect_daily_reward():
238     global jwt_token
239     global login
240
241     token = jwt_token
242
243     headers = {
244         'Authorization': f'Bearer {token}',
245     }
246
247     response = requests.get(url=f"{baseUrl}/rewards/dailyRewardStatus?
248     username={login}", headers=headers)
249     assert response.status_code == 200, f"Error! Status code is {response.
250     status_code}"
251     assert response.text
252
253 def test_get_user_referral_link():
254     global jwt_token
255     global login
256
257     token = jwt_token
258
259     headers = {
260         'Authorization': f'Bearer {token}',
261     }
262
263     response = requests.get(url=f"{baseUrl}/referralLinks/?username={login
264     }", headers=headers)
265     assert response.status_code == 200, f"Error! Status code is {response.
266     status_code}"
267     assert response.__str__, f"Error! The response is {response}"
268
269 def test_visit_referral_link():
270     global jwt_token
271     global login
272
273     token = jwt_token
274
275     headers = {

```

```

270         'Authorization': f'Bearer {token}',
271     }
272
273     salt = requests.get(url=f"{baseUrl}/referralLinks/?username={login}",
274 headers=headers).__str__
275
276     response = requests.post(url=f"{baseUrl}/referralLinks/visit?
277 visitorName=hehe&referralSalt={salt}", headers=headers)
278
279     assert response.status_code == 403, f"Error! The status code is {
280 response.status_code}"
281
282 def test_post_check_quiz_result():
283     global jwt_token
284     global login
285
286     token = jwt_token
287
288     headers = {
289         'Authorization': f'Bearer {token}',
290     }
291
292     body = [
293         {
294             "questionId": 1,
295             "questionAnswer": 0
296         },
297         {
298             "questionId": 2,
299             "questionAnswer": 3
300         },
301         {
302             "questionId": 3,
303             "questionAnswer": 2
304         },
305         {
306             "questionId": 4,
307             "questionAnswer": 2
308         },
309         {
310             "questionId": 5,
311             "questionAnswer": 1
312         }
313     ]
314
315     [
316         {
317             "questionId": 1,
318             "questionAnswer": 0
319         },
320         {
321             "questionId": 2,
322             "questionAnswer": 3
323         },
324         {
325             "questionId": 3,
326             "questionAnswer": 2
327         },
328         {
329             "questionId": 4,

```

```

327         "questionAnswer": 2
328     },
329     {
330         "questionId": 5,
331         "questionAnswer": 1
332     }
333 ]
334
335 response = requests.post(url=f"{baseUrl}/quiz?username={login}",
336 headers=headers, json=body)
337 assert response.status_code == 200, f"Error! The status code is {
338 response.status_code}"
339
340 dataOfResponse = response.json()
341 isQuizCompleted = dataOfResponse.get('isQuizCompleted')
342 assert isQuizCompleted == False

```

Листинг 1: Код для тестирования API