Санкт-Петербургский государственный университет

Кафедра системного программирования
Группа 22.Б11-мм

Улучшение окружения кода miminet для дальнейшего масштабирования

Зайцев Дмитрий Сергеевич

Отчёт по производственной практике

Научный руководитель:
старший преподаватель СПбГУ Зеленчук И. В.

Санкт-Петербург
2024

Оглавление

Введение

Miminet [1] — веб-тренажёр, предназначенный для обучения основам компьютерных сетей. Он позволяет создавать виртуальные компьютерные сети, настраивать устройства и эмулировать процессы сетевого взаимодействия.

Проект активно развивается, на данный момент курс «Компьютерные сети» [2], который читается на математико-механическом факультете Санкт-Петербургского государственного университета, основан на примерах из этой обучающей платформы. Miminet также используется для проверки знаний в области компьютерных сетей. Например, при промежуточном тестировании студентов курса или при отборе стажёров на «YADRO ИМПУЛЬС 2023».

При этом разработка приложения не стоит на месте. Каждый семестр в Miminet приходят новые студенты, желающие получить опыт в работе над реальным проектом. Каждый из них добавляет новую функциональность, либо расширяет уже имеющуюся. Изменения в уже написанный код также вносятся, например, для исправления найденных ошибок, уязвимостей или просто для улучшения производительности и читаемости кода, повышения его масштабируемости и адаптации к новым требованиям. Например, в осеннем семестре разработкой занимались пять новых человек, и суммарно в проект было добавлено свыше пяти тысяч строк кода.

Однако процесс разработки усложняется отсутствием полноценного тестирования приложения. Каждое внесение изменений всегда сопряжено с риском повреждения уже работающего кода. При этом проверить новую функциональность на безопасность можно только вручную, самостоятельно используя каждую функцию в интерфейсе.

Из этого можно сделать вывод, что отсутствие полноценного автоматического тестирования создаёт лишние риски и неудобства. Ручная проверка каждой функции трудоёмка (от пяти минут до получаса на одно изменение, включая построение сетей, их настройку, проверку отдельных кнопок) и не гарантирует полного выявления проблем. Поэтому разработка метода автоматического end-to-end тестирования Miminet является важной задачей для обеспечения стабильности и надёжности приложения, а также для ускорения процесса разработки и снижения рисков, связанных с внесением изменений в код.

Задачи

Целью работы является создание метода автоматического end-to-end тестирования Miminet.

В ходе работы были поставлены следующие **задачи**:

1. Обзор технологий.
2. Сбор требований.
3. Создание первых тестов на ключевые функции Miminet.
4. Добавление возможности локального запуска тестов.
5. Интеграция с CI.

Обзор

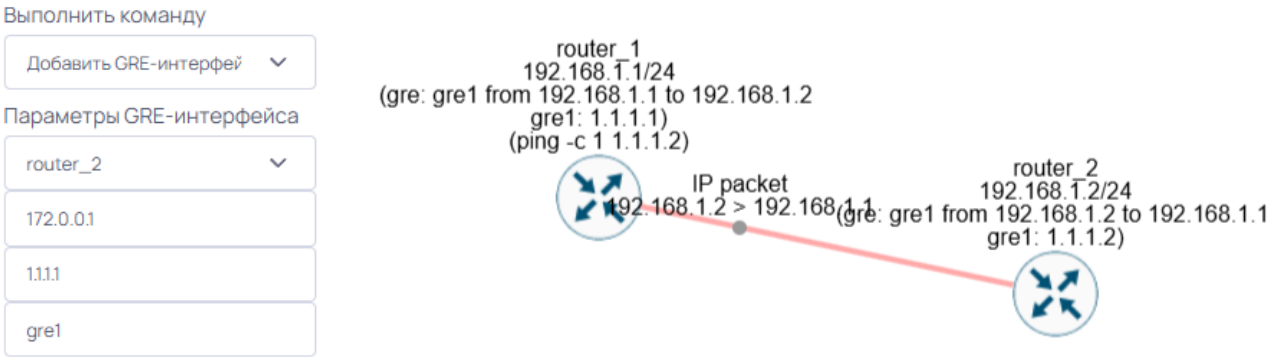
2.1 Добавление GRE

В связи с обновлением структуры курса «Компьютерные сети» и добавлением в него раздела «Туннелирование», появилась необходимость добавить в проект поддержку GRE-туннелей.

GRE [3] (*Generic Routing Encapsulation*, *Общая Инкапсуляция Маршрутов*) — это протокол туннелирования, обеспечивающий инкапсуляцию пакетов сетевого уровня (например, IPv4, IPv6, IPX, AppleTalk и так далее) в IP-пакеты для передачи через IP-сети. Данный протокол решает основную проблему IPIP (IPIP туннель может инкапсулировать только IPv4 пакеты) и чаще используется на практике для построения туннелей.

Чтобы добавить поддержку GRE-туннелирования в Miminet, необходимо было создать форму для конфигурации роутера и добавить для его эмулятора необходимые настройки.

Результат выполненной работы представлен ниже:



(a) Панель настройки GRE.

(b) Пример сети.

Рис. 1: Реализация GRE.

2.2 Обновление архитектуры

В процессе реализации описанной выше функциональности был обнаружен небезопасный фрагмент с избыточным использованием повторяющегося (дублированного) кода. Эта часть проекта была предназначена для конфигурации сетевых устройств и проверки передаваемых им параметров на стороне фронтенда. Основными выявленными проблемами были: отсутствие целостной структуры, неравномерность проверок для входных данных и несогласованность сообщений об ошибках.

В ходе работы была проанализирована старая структура кода, были выявлены её слабые места, а также разработана полностью новая архитектура, позволяющая добавлять новую

функциональность (описывать методы конфигурации сетевых устройств, добавлять проверки на них) максимально просто, при этом не допуская прежних уязвимостей.

Ниже представлена диаграмма классов. На ней хорошо видно, как с помощью наследования получилось решить проблему дублирующегося кода и повысить связность.

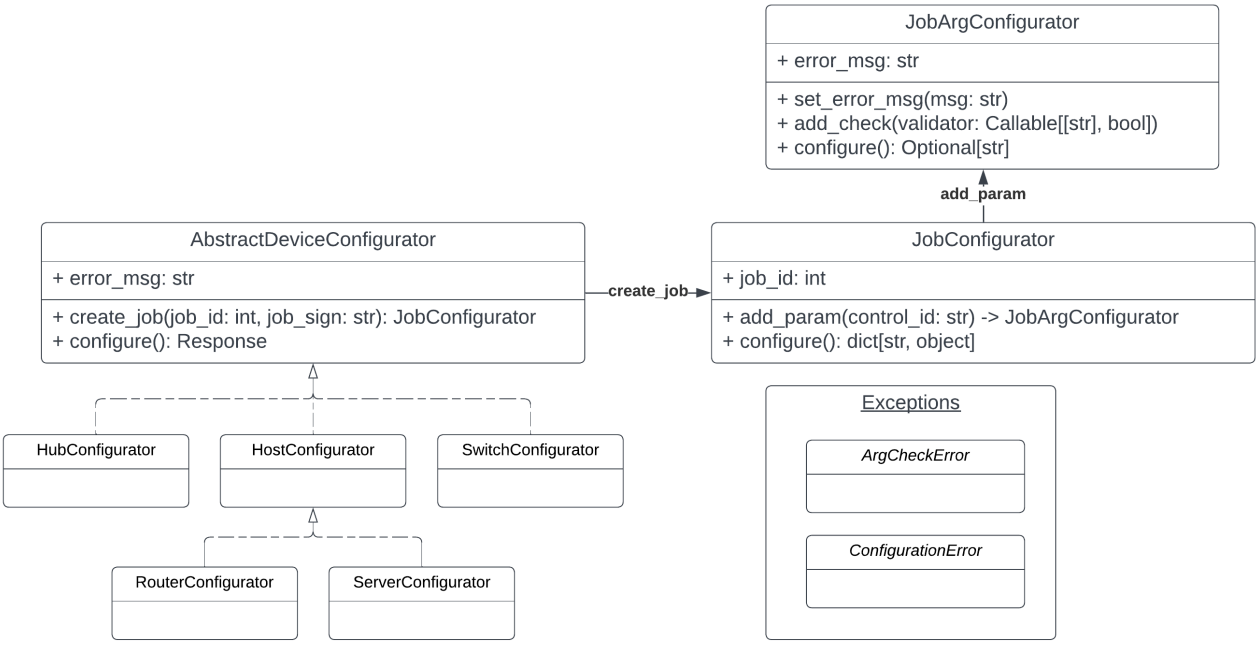


Рис. 2: Обновлённая архитектура.

Также удалось упростить процесс добавления новых параметров конфигурации. До обновления архитектуры для выполнения тех же действий приходилось вручную разбираться в структуре всего файла (около двух тысяч строк кода), а затем копировать подходящий участок и вставлять его, внося некоторые изменения под свой конкретный случай. Результатом таких преобразований становились: плохая читаемость кода, сложности в его обновлении и поддержке, а также некорректная обработка неправильно введённых пользователем данных.

Сейчас для добавления новой функциональности достаточно ознакомиться с примером ниже:

Алгоритм 1 Добавление новых параметров конфигурации устройства.

```
# ping -c 1 (1 param)
host_ping_job = host.create_job(1, "ping -c 1 [0]")
host_ping_job.add_param("config_host_ping_c_1_ip")\
    .add_check(IPv4_check)\
    .set_error_msg(build_error(ErrorType.ip, "ping"))
```

На нём можно понять, как переиспользуются методы проверки корректности введённых данных, а также как происходит генерация сообщений об ошибках. Кроме того, общее количество строк в файле уменьшилось почти в два раза, что значительно облегчило процесс добавления новых элементов и проверок.

2.3 Подробное описание проблемы

Чтобы обосновать необходимость в системе для автоматического тестирования, приведу несколько примеров из своей собственной практики.

Через неделю после обновления архитектуры пользователи начали находить ошибки в работе Miminet. Например, из-за пропущенного оператора *break* кнопка «Добавить сабинтерфейс с VLAN [4]» перестала показывать панель конфигурации. Точно так же нашли ещё одну проблему: проверка корректности введённых TCP [5] и UDP [6] портов давала сбой при числах, больше тридцати двух. Причина была в том, что при переносе настроек в новую версию проекта один из параметров был перепутан, и вместо проверки порта выполнялась проверка маски подсети.

Хотя устранение неполадок заняло не более получаса, всё то время, в течение которого сервис функционировал некорректно, причинило пользователям значительные неудобства. Кроме того, выявление этих проблем могло затянуться ещё сильнее, если бы пользователи сами не уведомили о них.

Подобных ситуаций можно было бы легко избежать, если бы в Miminet была полноценная система для end-to-end тестирования.

2.4 Обзор технологий

Архитектура Miminet состоит из фронтенда (веб-интерфейс и серверная часть для работы с базой данных) и бэкенда (эмулятор сетей Mininet). Фронтенд взаимодействует с бэкендом через брокер сообщений RabbitMQ [7]. Каждая отдельная часть проекта запускается в своём docker-контейнере, что позволяет избежать проблем с зависимостями, а также обеспечивает необходимый уровень изоляции и защиты компонентов приложения.

Для реализации системы автоматического тестирования необходимо было найти подходящую технологию для имитации работы пользователя с интерфейсом приложения. В результате анализа был выбран **Selenium** [8] [9], инструмент для автоматизации действий веб-браузера. Он поддерживает большинство современных браузеров: Chrome, Firefox, Safari, Edge и имеет реализацию в таких языках, как Java, Python (на котором написана большая часть Miminet), C#, JavaScript, Ruby. Кроме того, Selenium получает регулярные обновления и поддерживается уже двадцать лет, а также имеет исчерпывающую документацию и активное сообщество разработчиков. Среди аналогов (таких как **Cypress** [10]) инструмент выделяется своей простотой, стабильностью, поддержкой в популярных языках, а также возможностью параллельного запуска тестов и лёгкой интеграцией в CI/CD.

Для реализации возможности локального запуска тестов были использованы **Docker** [11] (система контейнеризации, позволяющая упаковывать приложения и их зависимости в изолированные контейнеры) и **Docker Compose** [12] (инструмент для определения и управления многоконтейнерными приложениями Docker). Также был применён **Selenium Grid** [13] – расширение Selenium, позволяющее распределять запросы по нескольким машинам (или контейнерам) для параллельного запуска тестов и одновременного тестирования приложения на разных версиях веб-браузеров.

Требования к тестирующей системе

Перед разработкой тестирующей системы был определён список требований. Система для тестирования Miminet должна:

- уметь выполнять автоматические тесты, имитирующие реальные сценарии использования проекта,
- обеспечивать возможность локального запуска тестов без ручной настройки среды и управления зависимостями,
- работать только в специальном режиме приложения для тестирования,
- быть легко адаптируемой к изменениям проекта (простое добавление и обновление тестов),
- интегрироваться с CI/CD, автоматически запускаясь при изменении кода и предоставляя наглядные результаты его проверки.

Реализация

4.1 Архитектура

Для реализации возможности локального запуска тестов в проект были добавлены два новых docker-контейнера: Selenium-hub и Chrome. Первый контейнер [14] – это центральный узел в Selenium Grid, который управляет всеми узлами (nodes) и распределяет задачи между ними. С его помощью можно параллельно запускать тесты или выполнять проверку Miminet сразу на разных версиях веб-браузеров. Второй контейнер [14] – узел в Selenium Grid, который запускает веб-браузер (Chrome) и выполняет в нём тесты, имитируя действия пользователя. Новые контейнеры объединены в сеть miminet_network, в которой также находится фронтенд приложения, к которому отправляются все запросы.

На данный момент все тесты запускаются только на одной версии браузера и параллельность не используется, однако архитектура допускает простое масштабирование в случае необходимости.

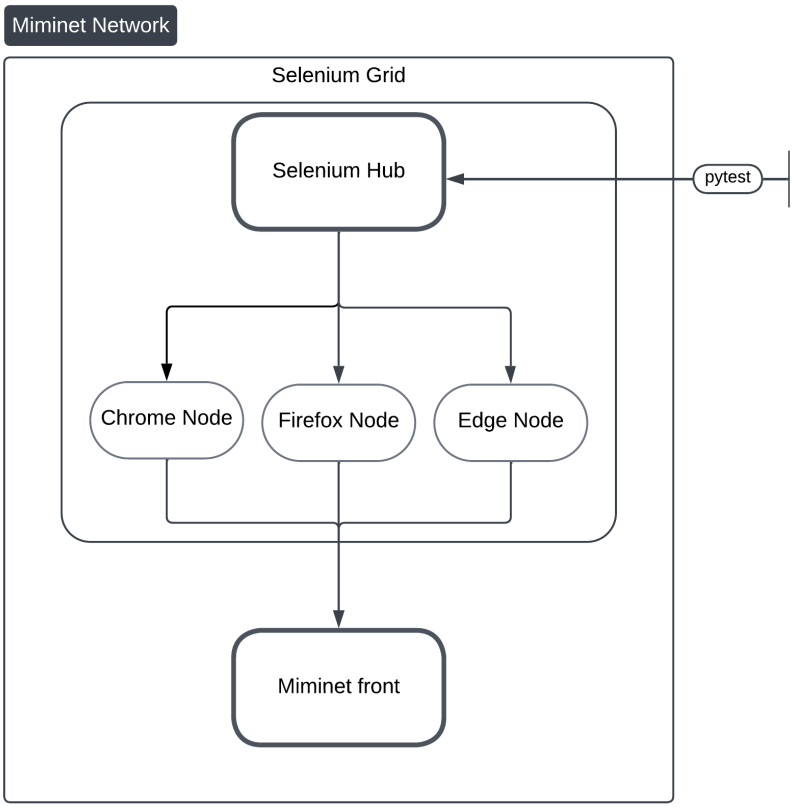


Рис. 1: Взаимодействие между тестирующей системой и приложением.

Специально для тестирования, в Miminet был добавлен «режим разработки» и «рабочий режим». Их главное отличие в том, что в «режиме разработки» тестирующая система может

свободно пользоваться Miminet, проходя авторизацию через специально внесённые в базу данных параметры. При этом в «рабочем режиме», в котором приложение запускается по умолчанию, база данных не содержит таких параметров для входа. Сделано это было с целью обеспечения безопасности рабочей среды и предотвращения несанкционированного доступа к системе.

Как было заявлено в требованиях, процесс создания новых тестов и изменения старых должен быть достаточно удобным, чтобы новые разработчики не тратили на это лишнее время. С этой целью было принято несколько архитектурных решений, благодаря которым появилась возможность писать тесты на конфигурацию сетей на более высоком уровне абстракций. Схема того, как это было реализовано, приведена ниже:

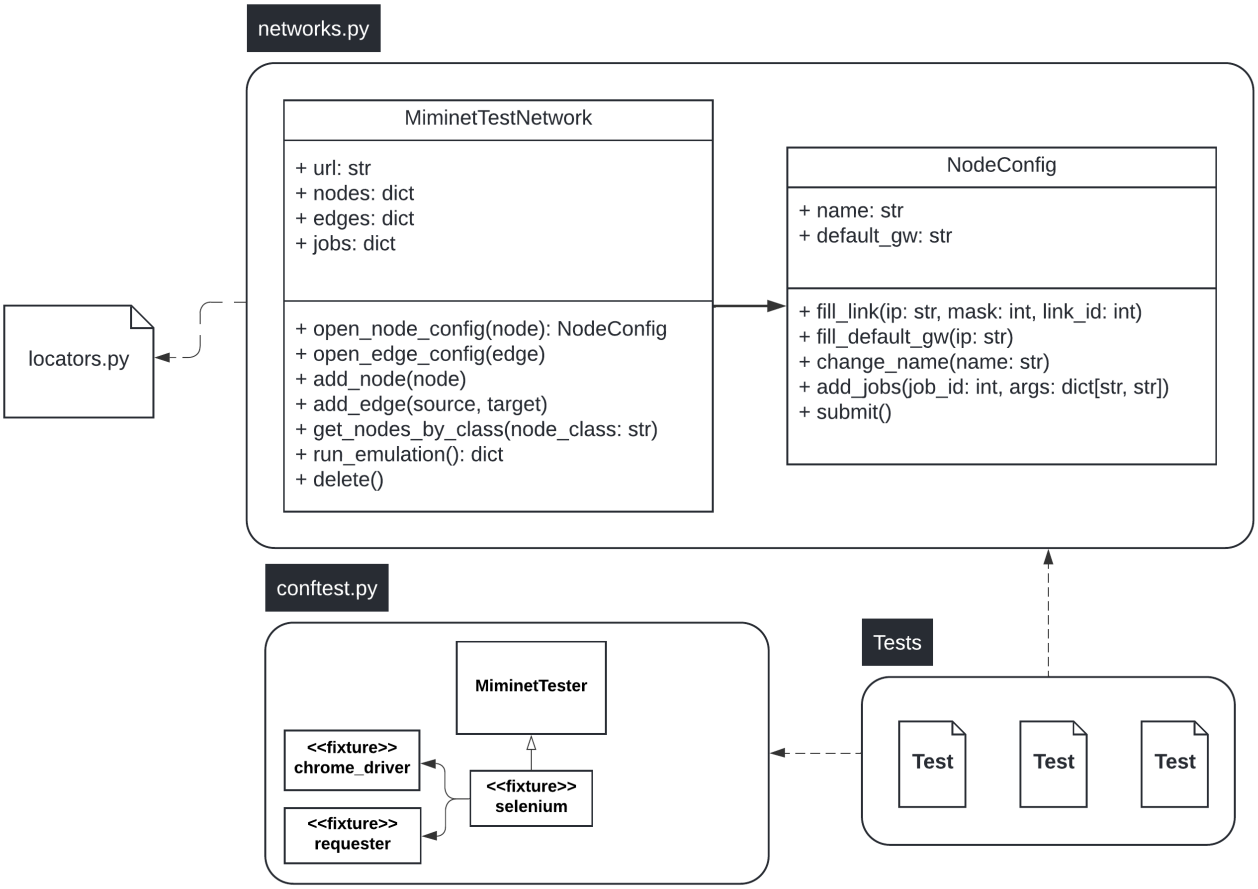


Рис. 2: Архитектура тестирующей системы.

В файле networks.py описаны абстракции, предназначенные для простой конфигурации сетей. Благодаря их использованию, код тестов не зависит от изменений в Miminet. Кроме того, время, необходимое на написание одного теста, сокращается до пяти-десяти минут за счёт эффективного переиспользования кода.

Фикстуры, определённые в conftest.py, используются в каждом тесте, они позволяют взаимодействовать с необходимыми функциями selenium, а также использовать специализированные методы для работы с Miminet (определённые в MiminetTester).

Файл locators.py представляет собой удобно структурированный набор констант, к которому обращаются элементы тестирующей системы при необходимости найти что-либо на странице. В случае изменений в структуре Miminet, константы можно будет легко заменить на другие, не меняя сам код тестов.

4.2 Тесты

На данный момент реализованы следующие тесты:

1. Проверка доступности ключевых страниц сайта.
2. Создание пустой сети.
3. Проверка навигации в меню выбора сетей.
4. Размещение устройств в сети.
5. Изменение имён устройств.
6. Создание связей между устройствами.
7. Конфигурация простой сети, использующей задачу «ping».
8. Конфигурация сетей, использующих протоколы туннелирования «IPIP»/«GRE».
9. Конфигурация сети с «NAT».
10. Конфигурация сетей, использующих протоколы «TCP»/«UDP».

При создании и настройке сетей тестирующая система имитирует действия пользователя. При этом результат сверяется с заданным заранее шаблоном. Таким образом, гарантируется корректность работы приложения (возможность создавать аналогичные сети). Ниже приведён пример такой сети, а также описан процесс её тестирования и конфигурации.

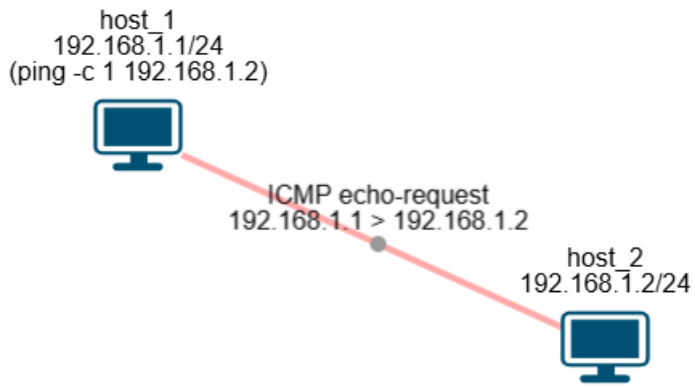


Рис. 3: Сеть, построенная тестирующей системой.

Алгоритм 2 Конфигурация и тестирование простой сети.

```
@pytest.fixture(scope="class")
    def network(self, selenium: MiminetTester):
        # Create a new network
        network = MiminetTestNetwork(selenium)

        # Add network devices
        network.add_node(NodeType.Host)
        network.add_node(NodeType.Host)

        # Add edges between them
        network.add_edge(source_id=0, target_id=1)

        # Configure first host
        config0 = network.open_node_config(node_id=0)
        config0.fill_link("192.168.1.1", 24)
        config0.add_jobs(1, {host_job_ip_selector: "192.168.1.2"})
        config0.submit()

        # Configure second host
        config1 = network.open_node_config(1)
        config1.fill_link("192.168.1.2", 24)
        config1.submit()

        # Return configured network
        yield network

        network.delete()

    def test_ping(self, selenium: MiminetTester, network: MiminetTestNetwork)
        # Compare network's data with templates
        assert compare_nodes(network.nodes, self.JSON_NODES)
        assert compare_edges(network.edges, self.JSON_EDGES)
        assert compare_jobs(network.jobs, self.JOBS)
```

Результаты

За семестр в рамках учебной практики были решены следующие задачи:

- выполнен обзор технологий,
- собраны требования для тестирующей системы,
- созданы первые тесты на основные функции Miminet,
- реализована возможность локального запуска тестов,
- выполнена интеграция с CI.

Литература

1. *Зеленчук Илья*. Веб-эмулятор компьютерных сетей Miminet. — URL: https://miminet.ru/.
2. *Зеленчук Илья*. Курс «Основы компьютерных сетей». — URL: https://stepik.org/course/208904/info.
3. *D. Farinacci T. Li S. Hanks D. Meyer P. Traina*. Generic Routing Encapsulation specification. — 2000. — URL: https://datatracker.ietf.org/doc/html/rfc2784.
4. *IEEE*. Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks. — 2018. — URL: https://standards.ieee.org/ieee/802.1Q/6844/.
5. *Information Sciences Institute*. Transmission Control Protocol Specification. — 1981. — URL: https://datatracker.ietf.org/doc/html/rfc793.
6. *Postel J.* User Datagram Protocol specification. — 1980. — URL: https://www.rfc-editor.org/rfc/rfc768.
7. *VMware*. RabbitMQ main page. — URL: https://www.rabbitmq.com/.
8. *Selenium*. Selenium Documentation. — URL: https://www.selenium.dev/documentation/.
9. *Selenium*. Selenium github page. — URL: https://github.com/SeleniumHQ/selenium.
10. *Cypress*. Cypress Documentation. — URL: https://docs.cypress.io/app/get-started/why-cypress.
11. *Inc. Docker*. Docker main page. — URL: https://www.docker.com/.
12. *Inc. Docker*. Docker Compose Documentation. — URL: https://docs.docker.com/compose/.
13. *Selenium*. Selenium Grid Documentation. — URL: https://www.selenium.dev/documentation/grid/.
14. *Selenium*. Hub and Node in Selenium Grid. — URL: https://www.selenium.dev/documentation/grid/getting_started/#hub-and-node.