СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc98088484)

[1. Анализ предметной области 4](#_Toc98088485)

[1.1. Описание предметной области 4](#_Toc98088486)

[1.2. Обзор аналогов 6](#_Toc98088487)

[1.3. Формирование требований к системе 9](#_Toc98088488)

[1.4. Выбор и обоснование средств разработки 10](#_Toc98088489)

[2. Проектирование и разработка программного обеспечения 12](#_Toc98088490)

[2.1. Проектирование архитектуры 12](#_Toc98088491)

[2.2. Разработка модуля обработки данных 15](#_Toc98088492)

[2.3. Разработка модуля работы с БД 17](#_Toc98088493)

[2.4. Разработка модуля анализа и отображения данных 18](#_Toc98088494)

[3. Тестирование программного обеспечения 21](#_Toc98088495)

[3.1. Разработка и проведение модульных тестов 21](#_Toc98088496)

[3.2. Разработка и проведение тестов производительности 23](#_Toc98088497)

[3.3. Оформление протокола логирования 24](#_Toc98088498)

[ЗАКЛЮЧЕНИЕ 25](#_Toc98088499)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26](#_Toc98088500)

[ПРИЛОЖЕНИЕ А. Модуль обработки данных 27](#_Toc98088501)

[ПРИЛОЖЕНИЕ Б. Модуль работы с БД 28](#_Toc98088502)

[ПРИЛОЖЕНИЕ В. Модуль визуализации данных 30](#_Toc98088503)

[ПРИЛОЖЕНИЕ Г. Код модульного тестирования 32](#_Toc98088504)

[ПРИЛОЖЕНИЕ Д. Код тестирования производительности 34](#_Toc98088505)

# ВВЕДЕНИЕ

В настоящее время, в эпоху стремительного развития компьютерной автоматизации, формат передачи данных является важнейшим фактором при построении каналов связи. Благодаря распространению беспроводных сетей, наибольшую популярность стал набирать стандарт IEEE 802.11S., описывающий mesh-сети. Данный стандарт предлагает создание децентрализованной системы передачи данных, где каждый участник сети является полноправным участником.

Целью данной курсовой работы является разработка программного обеспечения для отображения диагностической информации о состоянии беспроводной многоячейковой сети. Объектом исследования является процесс построения визуального отображения данных в виде графа. Предмет исследования – диагностика беспроводных многоячейковых сетей.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ предметной области, а именно описать сферу исследования, изучить аналоги, составить требования к системе в целом и выбрать инструменты для её реализации;
2. Спроектировать и разработать все основные модули, необходимые для работы информационной системы, начиная с разбора поступающих данных, продолжая работой с базой данных и заканчивая модулем построения визуализации;
3. Провести тестирование разработанного программного продукта, включающего в себя модульное тестирование, тестирование на производительность и построения протокола ведения журнала.

# Анализ предметной области

Данный раздел посвящен исследованию предметной области, обзору существующих систем анализа данных состояния беспроводных сетей, а также будет приведено обоснование для разработки собственного программного обеспечения с рассмотрением средств разработки. Анализ предметной области заключается в рассмотрении процесса развертывания mesh-сетей, их основные преимущества и недостатки. Обзор существующих аналогов позволит выявить основной функционал, требования к входным данным и прочие технические требования к системам анализа данных беспроводных устройств, находящихся в одной сети. На основе изучения вышеперечисленных данных будут составлены требования и соответственно выбраны инструменты для реализации собственной информационной системы.

## Описание предметной области

Mesh-сети – это сетевая топология, в которой беспроводные устройства объединяются между собой многочисленными каналами связей, зависящие от стратегических соображений развертывания сети. Идея создания подобной самоорганизующейся сети была предложена давно, но эффективная реализация стала доступна лишь в последнее время в связи с глобальным распространением беспроводных устройств.

Подобные сети часто применяются военными силовыми структурами различных стран для развертывания оперативной связи в тактических целях, к примеру, во время проведения антитеррористических операций, в зонах локальных военных конфликтов. Основным преимуществом данной топологии является её высокая отказоустойчивость, которая напрямую коррелирует с количеством устройств в сети. Каждый элемент в данной сети является не только абонентским устройством, но и играет роль узла и для других устройств. За счет этого появляется возможность легкого масштабирования mesh-сети. Mesh-сеть обладает следующими ключевыми возможностями:

1. Создание зон сплошного информационного покрытия большой территории;
2. Масштабирование сети в режиме самоорганизации;
3. Использование беспроводных транспортных каналов для реализации подключения «каждый к каждому»;
4. Устойчивость системы в целом при потере отдельных устройств.

Мesh-сети строятся как совокупность кластеров. Территория покрытия разделяется на кластерные зоны, количество которых теоретически не ограничено. В одном кластере размещается от 8 до 16 точек доступа. Одна из таких точек является узловой и подключается к магистральному информационному каналу с помощью кабеля или по радиоканалу. Узловые точки доступа, как и остальные точки доступа в кластере, соединяются между собой, а именно с ближайшими соседями, по транспортному радиоканалу. В зависимости от конкретного решения точки доступа могут выполнять функции ретранслятора, либо функции ретранслятора и абонентской точки доступа. Особенностью mesh-сети является использование специальных протоколов, позволяющих каждой точке доступа создавать таблицы абонентов сети с контролем состояния транспортного канала и поддержкой динамической маршрутизации трафика по оптимальному маршруту между соседними точками. При отказе какой-либо из них происходит автоматическое перенаправление трафика по другому маршруту, что гарантирует не просто доставку трафика адресату, а доставку за минимальное время. Процедура расширения сети в пределах кластера ограничивается установкой новых точек доступа, интеграция которых в существующую сеть происходит автоматически. Сеть способна к самовосстановлению и адаптации в условиях резких скачков трафика как внутри сети, так и на ее границах.

Недостаток подобных сетей заключается в том, что они используют промежуточные пункты для передачи данных. Это может вызвать задержку при пересылке информации и как следствие снизить качество трафика реального времени. Вследствие этого существуют ограничения на количество точек доступа в одном кластере.

## Обзор аналогов

До появления mesh-сетей огромная часть работы по управлению сложными взаимодействиями между сервисами ложилась на плечи разработчиков приложений. В этих условиях им нужен целый комплекс инструментов для управления жизненным циклом сети, начиная от контроля результатов развертывания кода и заканчивая управлением трафиком приложений в продакшн. Для успешной работы приложения все его службы должны нормально взаимодействовать друг с другом. Трассировка дает разработчику возможность отследить, как каждый сервис взаимодействует с остальными функциями, и помогает выявить узкие места, создающие ненужные задержки в реальной работе.

Для выявления основных требований к входным данным и функционалу информационной системы для визуализации данных диагностической беспроводной многоячейковой сети был произведен анализ действующих аналогов. Первый из них – любительский проект «Mesh Network», основанный на инструментах операционной системы Linux. Данный проект предоставляет самобытный протокол связи «B.A.T.M.A.N. Advanced», на основе которого есть возможность построения mesh-сети.

Визуализация топологии производится раз в минуту, обновляя изображение на веб-сервере. Протокол позволяет классифицировать устройства в сети по моделям, группировать по основным кластерам и вычислять время отклика между элементами, подключенных к одному каналу. К недостаткам визуализации относится невозможность динамического просмотра данных об элементах и слабая графическая составляющая визуализации.

На рисунке 1.1 приведено изображение визуализации топологии проекта «Mesh Network».

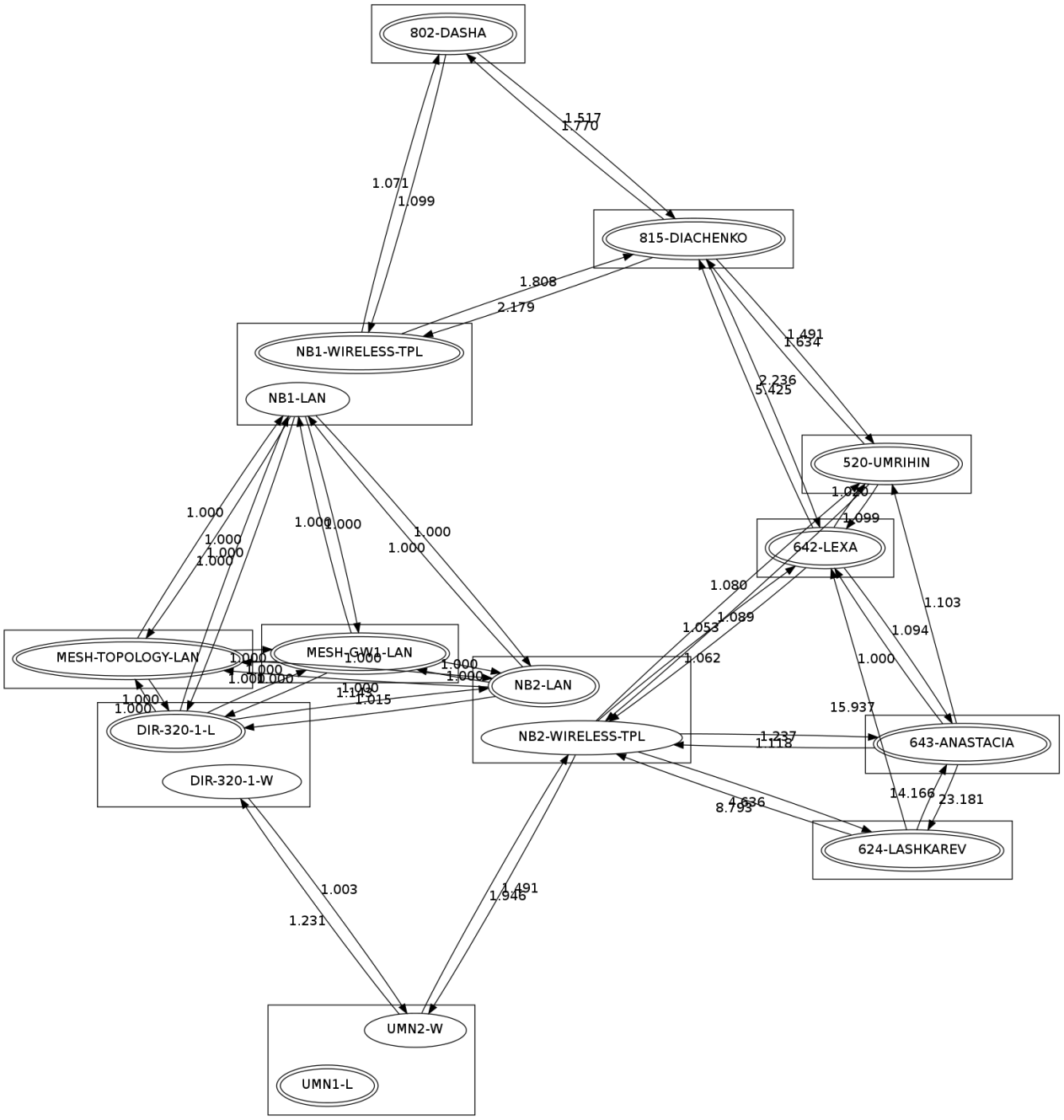


Рисунок 1.1 – Визуализация топологии проекта «Mesh Network».

Второй рассматриваемый аналог – коммерческий проект «OpenShift Service Mesh» от компании «Red Hat». Визуализация топологии системы беспроводных устройств строится на базе инструмента «Kiali». Данный инструмент позволяет не только строить визуализацию, но и поможет с созданием, проверкой и управлением конфигурациями mesh-сетки. Kiali визуализирует метрики, генерируемые Istio, и помогает понять, что происходит в mesh-сетке.

Другая важная вещь – умение маркировать сервисы приложения с помощью меток. Это важно, поскольку они добавляет контекстную информацию к метрикам и телеметрии, которые собираются Istio и затем используются в Kiali. Еще одна полезная вещь в Kiali – это то, что она может провести проверку вашей mesh-сетки, моментально демонстрируя «узкие» места рассматриваемой системы. Пример визуализации топологии беспроводных устройств в виде изображения приведено на рисунке 1.2.

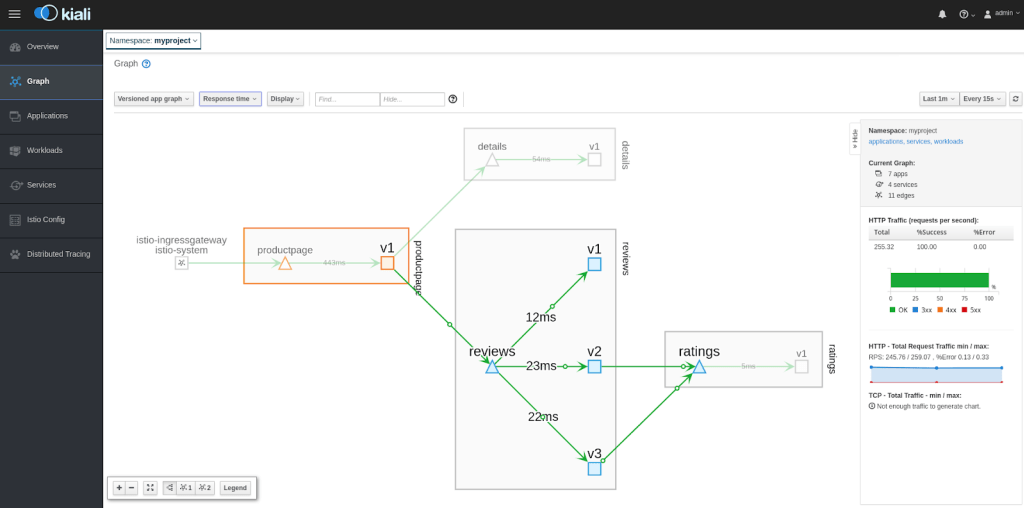


Рисунок 1.2 – Визуализация топологии в Kiali

## Формирование требований к системе

На основе проведенного анализа предметной области и изучения аналогичных действующих проектов были сформулированы требования к разрабатываемому программному обеспечению. Основная цель информационной системы – обработка диагностических данных о состоянии узлов многоячейковой беспроводной сети с последующей удобной и наглядной визуализации. Информация по каждой сети должна быть разобрана на составляющие и сохранена в базы данных.

В качестве исходных данных разрабатываемое программное обеспечение получает карту расположения узлов на объекте эксплуатации и информацию об их текущем сетевом окружении, включающие в себя множество идентификаторов соседних узлов, оценки качества связи, уровни сигнала, время с последнего полученного пакета и прочие параметры.

Разрабатываемая информационная система должна состоять из двух логических частей: конфигурация входных данных и визуализация выбранного объекта. Конфигурация входных данных заключается в указании названия файла и ввода JSON в соответствующее поле для последующей обработки. Визуализация же графа, исходя из требований наглядности, должна предоставлять полную информацию по каждому из устройств при минимальном занимаемом месте для отображения. Для наглядной визуализации необходимо использовать различные виды отображения данных, начиная от цветов и форм узлов, до всплывающих окон и подробным описанием устройств.

В силу специфики разрабатываемого программного обеспечения, оно должно быть кроссплатформенным и поддерживаться операционными системами Windows и Linux.

## Выбор и обоснование средств разработки

Для реализации поставленной цели в данной курсовой работе и выполнения всех требований, были выбраны следующие инструменты для реализации программного обеспечения для визуализации данных многоячейковой беспроводной сети:

1. Язык программирования Python для реализации серверной части приложения;
2. HTML и CSS для проектирования пользовательского интерфейса;
3. Нативный язык программирования JS для реализации канала связи между клиентской и серверной части приложения;
4. Специализированная библиотека Eel для установки слушателей запросов от клиентской части;
5. JSON в качестве формата входных данных;
6. Библиотека networkx для инициализации графа;
7. СУБД SQLite для хранения данных.

Язык программирования Python является активно развивающимся скриптовым языком, который используется для решения различных задач, начиная от реализации серверной логики веб-сервера и заканчивая обработкой больших данных. Python является кроссплатформенным и позволяет запускать собственные приложения в операционных системах как Windows, так и Linux.

Для реализации визуализации данных используется библиотека networkx, предназначение которой заключается в создании, обработке данных, выявлении закономерностей и возможность манипуляции созданными объектами в графическом интерфейсе. Данная библиотека позволяет оперировать следующими характеристиками графов: степени вершин, высота графов, диаметр, радиус узла, длины и промежуточные результаты ребер.

Для реализации графического интерфейса были выбраны инструменты HTML и CSS. Первый из них предназначен для формирования структуры содержания документа, их порядок и вложенность элементов. Второй же является инструментом для декорации, позволяя задавать различные стили каждому из элементов.

Для обработки входящих данных и отправки необходимых запросов серверу в клиентской части решено использовать язык программирования JS. Данный язык программирования имеет простой синтаксис и понятную архитектуру построения приложений. Так как основная среда выполнения данного языка – браузеры, то его запуск поддерживается различными операционными системами и устройствами.

Для наладки подключения к JS со стороны сервера будет использована библиотека Eel, позволяющая создавать события. Данная библиотека основана на веб-сокетах, и именно поэтому имеет схожий принцип построения канала связи между сервером и клиентом. Для корректной работы данной библиотеки запуск приложения должен инициализироваться самой библиотекой.

Между компонентами системы данные будут передаваться в формате JSON, что позволит повысить производительность вычислительных систем. Данный формат поддерживает различные типы данных, начиная объектами, продолжая массивами и заканчивая примитивными типами данных, таких как число, строка и так далее. Имеет иерархическую структуру построения архитектуры, где каждый из элементов может содержать множество других.

Для хранения данных была выбрана СУБД SQlite, позволяющая в кратчайшие сроки развернуть базу данных. Её главная особенность – не использование клиент-серверной технологии, то есть движок СУБД представляет из себя локальную библиотеку. Имеет открытый исходный код.

# Проектирование и разработка программного обеспечения

Данный раздел посвящен проектированию и разработке программного обеспечения, позволяющего отобразить диагностическую информацию о состоянии беспроводной многоячейковой сети. Реализация приложения состоит из нескольких этапов: проектирование архитектуры приложения, реализация модуля обработки входных данных и последующая их интеграция в базы данных.

## Проектирование архитектуры

Разрабатываемая информационная система для отображения диагностической информации о состоянии беспроводной многоячейковой сети реализовывается на основе клиент-серверной технологии. Термин «клиент-серверная архитектура» – сборное понятие, состоящее из двух взаимодополняющих компонентов: сервера и, собственно, клиента.

Клиент – программа, которая напрямую взаимодействует с пользователем посредством интерфейса, которая выполняет отправку запросов к серверу, а также обрабатывает поступающие данные для их последующей визуализации на интерфейсе.

Сервер же выполняет работы сервисного обслуживания клиентских запросов, предоставляет пользователям доступ к определенным системным ресурсам, сохраняет данные или БД. Особенности такой модели заключаются в том, что пользователь отправляет определенный запрос на сервер, где тот системно обрабатывается и конечный результат отсылается клиенту. В возможности сервера входит одновременное обслуживание сразу нескольких клиентов.

Параметры, которые могут реализоваться на стороне сервера:

1. Хранение, защита и доступ к данным.
2. Работа с поступающими клиентскими запросами.
3. Процесс отправки ответа клиенту.

Параметры, которые могут реализоваться на стороне клиента:

1. Площадка по предоставлению пользовательского графического интерфейса.
2. Формулировка запроса к серверу и его последующая отправка.
3. Получение итогов запроса и отправка дополнительной группы команд (запросы на добавление, обновление информации, удаление группы данных).

Архитектура системы клиент-сервер формулирует принципы виртуального общения между локальными компьютерами, а все правила и принципы взаимодействия находятся внутри протокола. На рисунке 2.1 продемонстрирована схема клиент-серверной архитектуры разрабатываемого приложения.

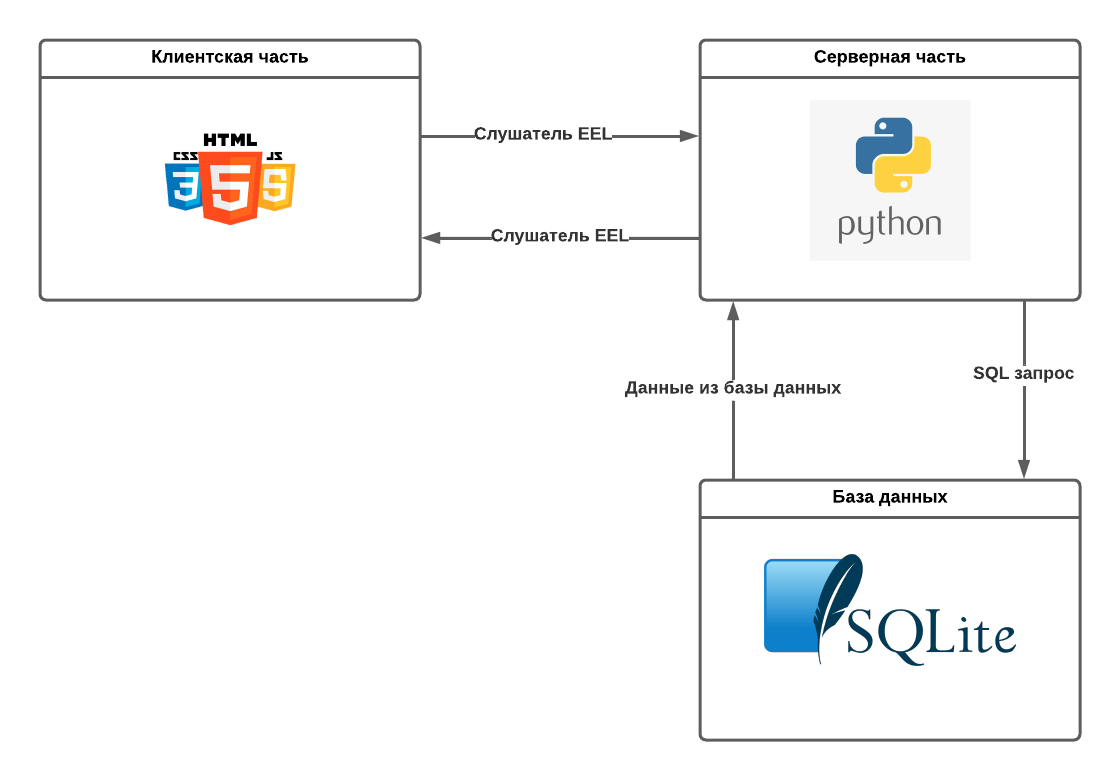


Рисунок 2.1 – Клиент-серверная архитектура

Разрабатываемая информационная система имеет серверную часть, написанную на языке программирования Python, клиентская часть реализовывается посредством HTML и CSS, а для обработки потока данных используется язык программирования JS. Для создания канала связи между сервером и клиентом применяется библиотека «Eel».

Процесс движения данных имеет представление «коридорного» типа, где все основные функции вызываются последовательно. Каждая такая функция имеет заголовок «@eel.expose» и выполняется асинхронно. Асинхронные запросы позволяют запускать функцию, не мешая выполнению другим участкам кода. При этом асинхронные запросы позволяют приостановить выполнение функции, пока не будет получен какой-либо ответ. К примеру, при запросе к базе данных, асинхронный запрос позволяет дать время СУБД сформировать все необходимые данные.

Начало работы приложения начинается с запуска страны конфигурации, событие которого запускает функцию инициализации, делающего запрос в базу данных и возвращает полученные значения странице. Далее, при событии создания новой mesh-сети, вызывается функция добавления поступающей информации в базу данных, в процессе которого происходит разбор JSON на составляющие элементы.

При запуске страницы визуализации срабатывает функция построения графа, выбранного на странице конфигурации. Данная функция выполняет запрос к базе данных и формирует новый JSON файл, вычленяет необходимые данные и отправляет клиенту. При получении данных клиентом, JS создает динамический граф, способный наглядно отобразить все важные данные как в целом по mesh-сети, так и по отдельным устройствам в сети. Схема работы приложения приведена на рисунке 2.2.

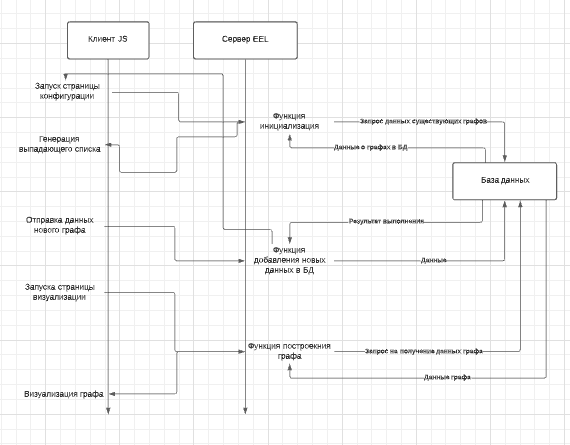


Рисунок 2.2 – Схема работы приложения

## Разработка модуля обработки данных

Входные данные имеют формат JSON. Для работы с данным форматом язык программирования Python предоставляет библиотеку «json», способная как сериализовать, так и десериализовать информацию. Обработка данных полностью производится на стороне сервера, итоговая информация же отправляется в клиентскую часть приложения.

Рассмотрим структуру и значения входящих данных:

1. «Info» – содержит общую информация о сети
   1. «AdapterCount» – количество адаптеров, соединений
   2. «DeviceCount» – количество устройств в сети
2. «Last\_changed» – код последнего изменения
3. «Nodes» – узлы сети
   1. «Alarm» – состояние тревоги
   2. «0Hops» – замыкающее устройство
   3. «ModbusID» – уникальный код протокола соеденения
   4. «Nodes» – количество связанных узлов
   5. «Quality» – качество связи
   6. «RadioID» – уникальный код устройства
   7. «RSSI» – уровень принимаемого сигнала
   8. «Seconds» – время в секундах с момента получения последнего сигнала
   9. «Uptime» – время безотказной работы
   10. «Devices» – список связанных устройств
       1. «ModbusID» – уникальный код протокола соеденения
       2. «Quality» – качество связи
       3. «RadioID» – уникальный код устройства
       4. «RSSI» – уровень принимаемого сигнала
       5. «Seconds» – время в секундах с момента получения последнего сигнала

Программный код модуля обработки данных приведен в приложении А. Данный модуль использует библиотеку networkx, предназначенную для создания, манипуляции и изучения сложных структур, имеющие вид графа. Входные данные для данного модуля состоят из списка устройств с подробным описанием характеристик, и содержит информацию о связях с другими устройствами. Для наглядного отображения данных используется граф, где каждая вершина – уникальное устройство, а грани – взаимоотношение устройств.

Программный код рассматриваемого модуля содержит два цикла, в первом происходит создание каждой вершины, и в той же итерации определяются связи с другими устройствами. На данном этапе происходит настройка параметров визуализации графа. К примеру, в случае, если устройство является замыкающим, то форма вершины будет прямоугольной, в ином случае форма вершины – круг. Помимо формы вершины в данном модуле также задается и цвет, зависящий от нескольких условий: если устройство передает тревогу, то ее цвет становится красным, стандартное же устройство представлено в оранжевом цвете.

Второй цикл необходим для перебора каждой связи между устройствами. Каждое из ребер имеет собственный цвет, напрямую зависящий от уровня принимаемого сигнала. Если значение сигнала меньше 100, цвет ребра – черный, если значение сигнала не достигает 200, то ребро окрашивается в желтый цвет, в ином случае цвет ребра будет зеленым.

## Разработка модуля работы с БД

Для описания информационного обеспечения была сформирована модель структуры данных (ER, IDEF 1X), построенной на основе модели DFD to-be, в которой указаны хранилища данных, необходимые для мониторинга противопожарной системы и дымоудаления организации. Для работы информационной системы необходима лишь база данных с тремя таблицами.

Первая из них предназначена для хранения данных о существующих mesh-сетях, а именно название сети, путь к файлу, количество адаптеров и устройств. Вторая и третья таблица используется для хранения данных об узлах и ребрах определенной mesh-сети. Каждая подобная сеть имеет собственное уникальное значение, на которое ссылается узел. Таким же образом каждый узел содержит уникальный код, на который уже ссылается каждое ребро. Схема отношений таблиц предоставлена на рисунке 2.3.

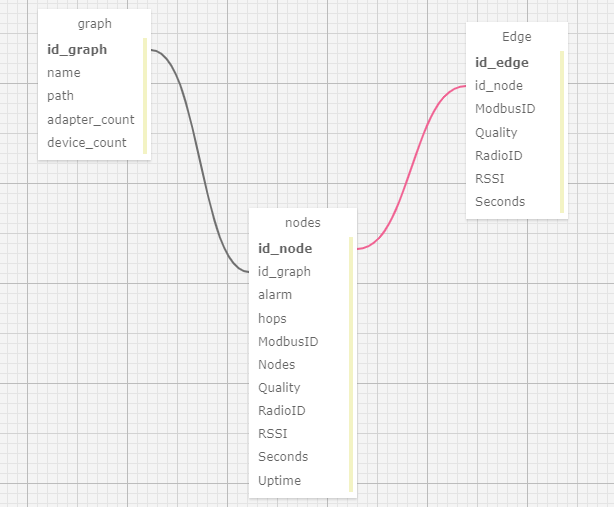


Рисунок 2.3 – Схема базы данных

Для работы с SQLite язык программирования Python предоставляет встроенную утилиту, позволяющую напрямую подключаться к СУБД и выполнять SQL-запросы. Программный код модуля приведен в приложении Б. Данный модуль выполняет подключение к существующей базе данных, выборку данных и внесение новых записей в таблицы.

## Разработка модуля анализа и отображения данных

Анализ и визуализация данных производится в клиентской части приложения. Графическая часть реализована посредством HTML и CSS, а управление данными производится посредством скриптового языка программирования JS. Программный код данного модуля представлен в приложении В.

Для реализации визуализации диагностических данных устройств mesh-сети были созданы два HTML-файла. В первом производится внесение данных новой сети, а также выбор графа из выпадающего списка для визуализации. Демонстрация данного файла приведено на рисунке 2.4.

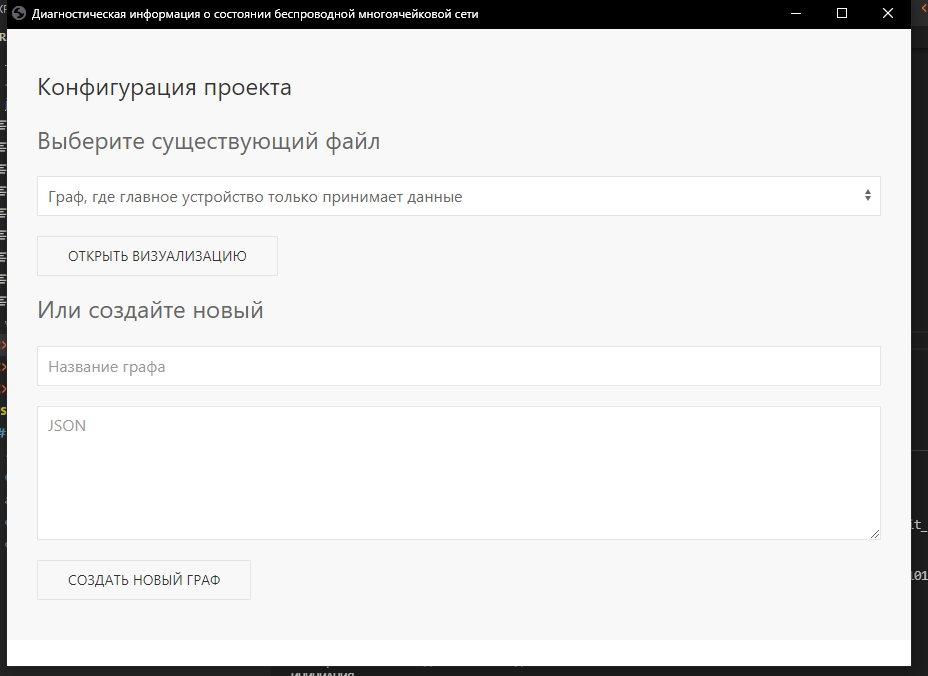


Рисунок 2.4 – Изображение страницы конфигурации

Выбрав необходимую сеть, пользователю необходимо нажать кнопку «открыть визуализацию». Информационная система перенаправит пользователя на новую страницу и отобразит граф, в котором каждая вершина – отдельное устройство, а ребра – каналы связи. Каждое устройство подписывается собственным уникальным кодом и имеет форму и цвет в зависимости от параметров. Каждый канал связи имеет направление, цвет и текстовую отметку об уровне сигнала. Для более удобного восприятия данных и во избежание наложения данных представленный граф находится в постоянном движении и имеет динамическую интерактивность, позволяющая передвигать вершины графа. Изображение визуализации графа представлено на рисунке 2.5.

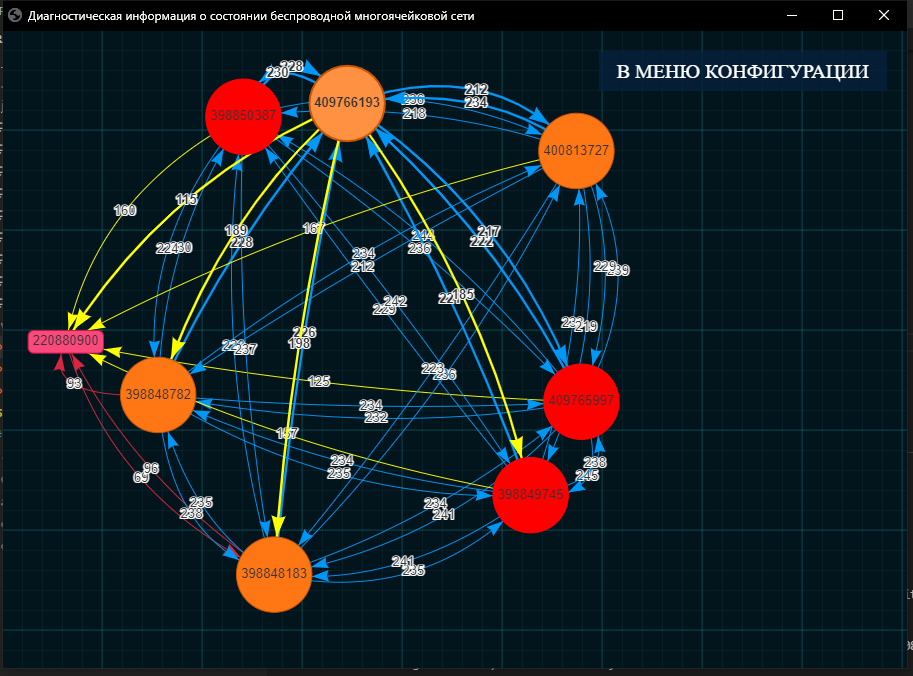


Рисунок 2.5 – Визуализация mesh-сети

# Тестирование программного обеспечения

Тестирование программного обеспечения является важным этапом реализации информационной системы, позволяющее выявить слабые места и предотвратить возникновение ошибок на ранних этапах. Для тестирования разрабатываемой информационной системы были реализованы UNIT-тесты, тесты на производительность основных функций. Помимо этого было реализована система протоколирования данных.

## Разработка и проведение модульных тестов

Модульное тестирование кода является неотъемлемой частью жизненного цикла разработки программного обеспечения. Модульные тесты также формируют основу для проведения регрессионного тестирования, то есть они гарантируют, что система будет вести себя согласно сценарию, когда добавятся новые функциональные возможности или изменятся существующие.

Ввиду характера разработанных функций, каждая из них возвращает динамический ответ, не коррелирующий напрямую от входных данных. Потому для реализации модульных тестов все основные функции были выведены в отдельный файл и возвращают определенное значение при успешном выполнении. Модульному тестированию подверглись три основные функции – подключение к базе данных, чтение данных с файла и генерация файлов на основе поступающих данных. Код модульного тестирования приведен в приложении Г.

Для реализации модульного тестирования использовалась встроенная библиотека «unittest», позволяющая задавать параметры функциям и вводить ожидаемый ответ. В случае совпадения результата тестируемой функции с ожидаемым ответом, модульный тест будет считаться успешно пройденным, в обратном же случае программа выдаст ошибку и укажет на разницу в ожидаемых ответах.

Для проведения модульного тестирования был создан класс «TestSystem», который наследуется от «unittest.TestCase». В нем прописаны все три тестовые функции. Для запуска модульного тестирования в корневом каталоге проекта необходимо запустить командную строку и ввести следующий скрипт «python -m unittest unitTest.py». В случае успешного прохождения всех модульных тестов, программа выведет об этом уведомление. Иллюстрация данного ответа представлена на рисунке 3.1.

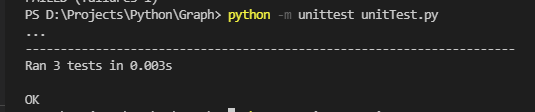


Рисунок 3.1 – Успешное прохождение всех модульных тестов

Для демонстрации неудачного прохождения модульного теста были внесены ошибочные данные в ожидаемый результат для функции открытия файла. В данном случае модуль тестирования выведет название функции и укажет на несоответствие результатов. На рисунке 3.2 продемонстрирован ответ приложения в данном случае.

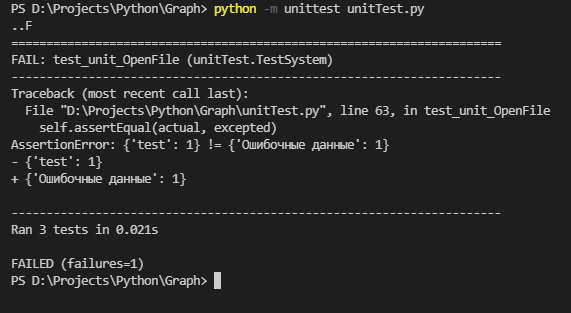


Рисунок 3.2 – Вывод данных об ошибке

## Разработка и проведение тестов производительности

Тестирование производительности, в отличие от модульного тестирования, связано не с проверкой на работоспособность, а на время выполнение каждой функции. Также может служить для проверки и подтверждения других атрибутов качества системы, таких как масштабируемость, надёжность и потребление ресурсов. Для разработанного программного обеспечения по визуализации диагностических данных многоячейковой сети будет проведено нагрузочное тестирование производительности на количество времени, требуемого для выполнения функции.

Для проведения подобного тестирования язык программирования Python предоставляет библиотеку «timeit», способную измерить количество времени, затрачиваемое на выполнение. Так как используемые функции в разработанной информационной системе являются вложенными, был создан отдельный файл, в котором функции принимают на вход псевдо данные. Всего было проверено 5 функций, а именно: производительность функции подключения к базе данных, чтения файла, генерации графа, выборки данных из БД, создание массива данных. Полный код модуля тестирования производительности приведен в приложении Д.

Первый параметром принимает код функции, вторым – количество итераций выполнений кода. Это позволяет выявить усредненное время выполнения кода. В нашем случае каждая функция выполняется по 10000 раз. Также стоит отметить, что результаты тестов напрямую зависят от аппаратного обеспечения устройства, на котором производятся вычисления. К примеру, запустив тест в первый раз, персональный компьютер затратит около 14 секунд на построения графа, однако далее, так как место в оперативной памяти уже занято данной функцией, дальнейшие вычисления будут занимать всего 2-3 секунды. Для запуска теста производительности необходимо открыть командную строку и запустить скрипт «python time.py».

Результат выполнения данной команды показан на рисунке 3.3.

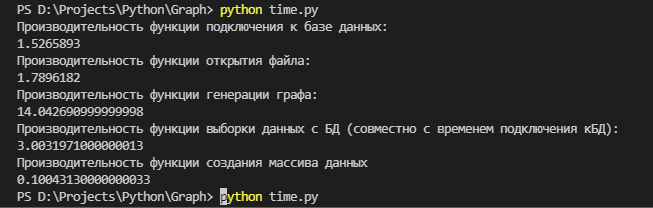


Рисунок 3.3 – Проведение тестирования производительности

## Оформление протокола логирования

Логирование – процесс автоматической записи всех процессов на всех этапах жизнедеятельности информационной системы. Зачастую журналирование ведется в текстовом файле и содержит время записи и описание процесса. Для ведения журнала действий разработанной информационной системы для визуализации диагностических данных mesh-сетей был создан текстовый файл «log.txt». Внутри каждой функции запрограммирована команда на добавление данных о результатах проделанной работы с добавлением даты и времени. В случае неуспешного выполнения функции в текстовый файл также передается информация с текстом ошибки. На рисунке 3.4 приведен пример логирования.

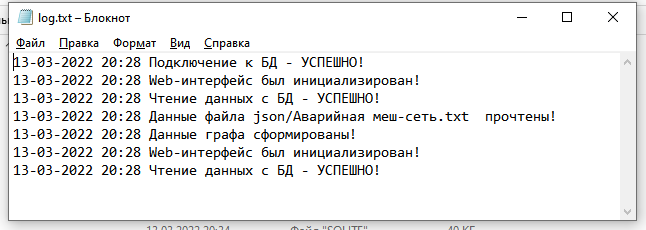


Рисунок 3.4 – Ведение логов информационной системы

# ЗАКЛЮЧЕНИЕ

Результатом выполнения данной курсовой работы стала информационная система, позволяющая визуализировать диагностические данные беспроводной многоячейковой сети, что означает успешное достижение поставленной цели. Для этого были решены несколько задач.

Первой решенной задачей стал анализ предметной области, где было подробно описана технология построения mesh-сетей, их основные преимущества и недостатки. Для закрепления и расширения полученных данных был произведен анализ действующих аналогов, изучены их принципы работы. На основе данной информации были составлены требования к разрабатываемой информационной системе, а также описан инструментарий для её реализации.

Второй решенной задачей стало проектирование и разработка отдельных модулей. Разработанные модули основаны на клиент-серверной архитектуре, где работы обработки данных и с базой данных производятся на стороне сервера, а визуализация на клиенте. В процессе решения данной задачи был реализован модуль обработки данных, в котором на составные части разбираются поступающие данные и экспортируются для дальнейшего построения графа. Вторым модулем стал программный код для работы с базой данных, в котором были реализованы функции подключения, выборки и добавления данных. Третий разработанный модуль отвечает собственно за саму визуализацию на основе обработанных данных.

Третьей решенная задача – разработка систем тестирования. Тестирование производилось тремя методами: модульное тестирование, при котором проверялся результат каждой функции на ожидаемый результат, тестирование производительности, где было замерено время выполнения работы функции и составлен протокол журнала действий.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вишневский В. В. Энциклопедия Wi-Max. Путь к 4G / В. В. Вишневский, С. Л. Портной, И. В.Шахнович. М.: Техносфера, 2010.
2. Легков К. Е., Донченко А. А. Беспроводные mesh сети специального назначения // T-Comm. 2009. №3. URL: <https://cyberleninka.ru/article/n/besprovodnye-mesh-seti-spetsialnogo-> naznacheniya (дата обращения: 28.02.2022).
3. Мальцев, Ю.Н. Введение в дискретную математику. Элементы комбинаторики, теории графов и теории кодирования / Ю.Н. Мальцев, Е.П. Петров. - М.: [не указано], 2010. - 869 c.
4. Попков Г. В. Mesh-сети: перспективы развития, возможные применения // Проблемы информатики. 2012. №3. URL: https://cyberleninka.ru/article/n/mesh-seti-perspektivy-razvitiya-vozmozhnye-primeneniya (дата обращения: 28.02.2022).
5. Портнов Э. Л. Принципы построения первичных сетей и оптические кабельные линии связи. М.: Горячая линия — телеком, 2009.
6. Соколов Н. А. Задачи планирования сетей электросвязи. СПб.: Техника связи, 2012.

# ПРИЛОЖЕНИЕ А

Модуль обработки данных

###                            МОДУЛЬ ОБРАБОТКИ ДАННЫХ            ###

def OpenFile(path):

    with open(path) as json\_file:

        json\_data = json.load(json\_file)

    return json\_data

#Генерация графа

def GenerateGraph(data):

    G = None

    G = nx.DiGraph()

    for nodes in data['nodes']:

        if nodes['hops'] == '1':

            node\_shape = 'circle'

            node\_color = '#ff7714'

        else:

            node\_shape = 'box'

            node\_color = '#fb497c'

        if nodes['alarm'] == '1':

            node\_color = 'red'

        title\_data = "ID: " + nodes['radioId'] + "\nВремя безотказной работы " + nodes['uptime'] + "c.\nУровень принимаего сигнала "+nodes['rssi']

        G.add\_node(nodes['radioId'], shape = node\_shape, color = node\_color, title = title\_data)

        for devices in nodes['devices']:

            if int(devices['quality']) < 100:

                colorEdge = "#d7263d"

            elif 100 < int(devices['quality']) < 200:

                colorEdge = "yellow"

            else:

                colorEdge = "#0197F6"

            G.add\_edge(nodes['radioId'], devices['radioId'], color = colorEdge, label = devices['quality'])

    return G

net = None

def createNet(data):

    net = Network(directed=True)

    net.set\_edge\_smooth('dynamic')

    net.from\_nx(GenerateGraph(OpenFile(data)))

    return net

# ПРИЛОЖЕНИЕ Б

Модуль работы с БД

###                            МОДУЛЬ ПОДКЛЮЧЕНИЯ К БАЗЕ ДАННЫх

#Создаем подключение

def createConnect(path):

    connection = None

    try:

        connection = sqlite3.connect(path)

        print("Тестирование #1: Подключение к БД - УСПЕШНО!")

    except Error as e:

        print("Тестирование #1: Подключение к БД - ОШИБКА! Информация - " + e)

    return connection

#Функция для чтение данных

def execute\_read\_query(connection, query):

    cursor = connection.cursor()

    result = None

    try:

        cursor.execute(query)

        result = cursor.fetchall()

        return result

    except Error as e:

        return e

#Функция для добавления данных

def execute\_query(connection, query):

    cursor = connection.cursor()

    try:

        cursor.execute(query)

        connection.commit()

        print("Запись успешно добавлена!")

        return cursor.lastrowid

    except Error as e:

        print(f"The error '{e}' occurred")

#Подключаемся

connection = createConnect("db.sqlite")

#Получаем данные всех доступных графов

select\_graph\_info = "SELECT \* FROM graph"

#Создаем новую меш-сеть

@eel.expose

def createNewGraph(name, json\_data):

    print(name)

    f = open(f"json/{name}.txt", "w")

    f.write(json\_data)

    insert\_graph\_info = f"INSERT INTO graph('name', 'path', 'adapter\_count', 'device\_count') VALUES('{name}', 'json/{name}.txt', 7, 1)"

    id = execute\_query(connection, insert\_graph\_info)

    #for nodes in json\_data['nodes']:

    #   insert\_node\_info = "ID: " + nodes['radioId'] + "\nВремя безотказной работы " + nodes['uptime'] + "c.\nУровень принимаего сигнала "+nodes['rssi']

    print(id)

    return "Новый граф добавлен"

# ПРИЛОЖЕНИЕ В

Модуль визуализации данных

<html>

    <head>

        <title>Диагностическая информация о состоянии беспроводной многоячейковой сети</title>

        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/vis/4.16.1/vis.css" type="text/css" />

        <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vis/4.16.1/vis-network.min.js"> </script>

        <script src="eel.js"></script>

        <link rel="stylesheet" href="style.css">

        <script type="text/javascript">

            async function init(){

                var name = localStorage.getItem("graph\_name");

                await eel.initGraph(name);

            }

            init();

        </script>

    </head>

    <body>

        <div id="mynetwork">

        </div>

        <div class="centered">

            <a class="scifi-button" href="configurate.html">В меню конфигурации</a>

        </div>

    </body>

    <script type="text/javascript">

        // initialize global variables.

        var edges;

        var nodes;

        var network;

        var container;

        var options, data;

        eel.expose(drawGraph);

        function drawGraph(nodes\_data, edges\_data) {

            var container = document.getElementById('mynetwork');

            // parsing and collecting nodes and edges from the python

            nodes = new vis.DataSet(nodes\_data);

            edges = new vis.DataSet(edges\_data);

            // adding nodes and edges to the graph

            data = {nodes: nodes, edges: edges};

            var options = {

                "edges": {

                    "color": {

                        "inherit": true

                    },

                    "smooth": {

                        "forceDirection": "none"

                    }

                },

                "interaction": {

                    "dragNodes": true,

                    "hideEdgesOnDrag": false,

                    "hideNodesOnDrag": false

                },

                "physics": {

                    "forceAtlas2Based": {

                    "centralGravity": 0,

                    "springLength": 100

                    },

                    "minVelocity": 0.75,

                    "solver": "forceAtlas2Based"

                }

            };

            network = new vis.Network(container, data, options);

            return network;

        }

    </script>

</html>

# ПРИЛОЖЕНИЕ Г

Код модульного тестирования

from cProfile import label

from cgi import test

from tkinter.font import names

from turtle import title

from typing import NewType

from unittest import result

import eel

import networkx as nx

import json

from pyvis.network import Network

import timeit

import sqlite3

from sqlite3 import Error

import unittest

def createConnect(path):

    connection = None

    try:

        connection = sqlite3.connect(path)

        return 1

    except Error as e:

        return e

def GenerateGraph(data):

    G = None

    G = nx.DiGraph()

    for nodes in data['nodes']:

        if nodes['hops'] == '1':

            node\_shape = 'circle'

            node\_color = '#ff7714'

        else:

            node\_shape = 'box'

            node\_color = '#fb497c'

        if nodes['alarm'] == '1':

            node\_color = 'red'

        title\_data = "ID: " + nodes['radioId'] + "\nВремя безотказной работы " + nodes['uptime'] + "c.\nУровень принимаего сигнала "+nodes['rssi']

        G.add\_node(nodes['radioId'], shape = node\_shape, color = node\_color, title = title\_data)

        for devices in nodes['devices']:

            if int(devices['quality']) < 100:

                colorEdge = "#d7263d"

            elif 100 < int(devices['quality']) < 200:

                colorEdge = "yellow"

            else:

                colorEdge = "#0197F6"

            G.add\_edge(nodes['radioId'], devices['radioId'], color = colorEdge, label = devices['quality'])

            result = nodes['radioId']

    return result

def OpenFile(path):

    with open(path) as json\_file:

        json\_data = json.load(json\_file)

    return json\_data

#           UNIT - ТЕСТИРОВАНИЕ

class  TestSystem(unittest.TestCase):

    def test\_unit\_Connect(self):

        actual = createConnect("db.sqlite")

        excepted = 1

        self.assertEqual(actual, excepted)

    def test\_unit\_OpenFile(self):

        actual =  OpenFile("unit/unit.txt")

        excepted = {'Ошибочные данные': 1}

        self.assertEqual(actual, excepted)

    def test\_generate\_graph(self):

        actual = GenerateGraph(OpenFile("json/Рабочий.txt"))

        excepted = '220880900'

        self.assertEqual(actual, excepted)

# ПРИЛОЖЕНИЕ Д

Код тестирования производительности

import timeit

connect = '''

import sqlite3

from sqlite3 import Error

def createConnect(path):

    connection = None

    try:

        connection = sqlite3.connect(path)

    except Error as e:

        error = e

    return connection

connection = createConnect("db.sqlite")

'''

openFile = '''

import json

def OpenFile(path):

    with open(path) as json\_file:

        json\_data = json.load(json\_file)

    return json\_data

OpenFile("json/MeshDiag.txt")

'''

createNet = '''

from pyvis.network import Network

def createNet():

    net = Network(directed=True)

    net.set\_edge\_smooth('dynamic')

    return net

createNet()

'''

selectFromDB = '''

import sqlite3

from sqlite3 import Error

def createConnect(path):

    connection = None

    try:

        connection = sqlite3.connect(path)

    except Error as e:

        error = e

    return connection

connection = createConnect("db.sqlite")

def execute\_read\_query(connection, query):

    cursor = connection.cursor()

    result = None

    try:

        cursor.execute(query)

        result = cursor.fetchall()

        return result

    except Error as e:

        return e

select\_graph\_info = "SELECT \* FROM graph"

execute\_read\_query(connection, select\_graph\_info)

'''

generateGraph = '''

import networkx as nx

from pyvis.network import Network

def GenerateGraph(data):

    G = None

    G = nx.DiGraph()

    for nodes in data['nodes']:

        if nodes['hops'] == '1':

            node\_shape = 'circle'

            node\_color = '#ff7714'

        else:

            node\_shape = 'box'

            node\_color = '#fb497c'

        if nodes['alarm'] == '1':

            node\_color = 'red'

        title\_data = "ID: " + nodes['radioId'] + "Время безотказной работы " + nodes['uptime'] + "c.Уровень принимаего сигнала "+nodes['rssi']

        G.add\_node(nodes['radioId'], shape = node\_shape, color = node\_color, title = title\_data)

        for devices in nodes['devices']:

            if int(devices['quality']) < 100:

                colorEdge = "#d7263d"

            elif 100 < int(devices['quality']) < 200:

                colorEdge = "yellow"

            else:

                colorEdge = "#0197F6"

            G.add\_edge(nodes['radioId'], devices['radioId'], color = colorEdge, label = devices['quality'])

    return G

json = {ЗДЕСЬ ТЕСТОВЫЙ JSON}

GenerateGraph(json)

'''

print("Производительность функции подключения к базе данных:")

print(timeit.timeit(connect, number=10000))

print("Производительность функции открытия файла:")

print(timeit.timeit(openFile, number=10000))

print("Производительность функции генерации графа:")

print(timeit.timeit(generateGraph, number=10000))

print("Производительность функции выборки данных с БД (совместно с временем подключения кБД):")

print(timeit.timeit(selectFromDB, number=10000))

print("Производительность функции создания массива данных")

print(timeit.timeit(createNet, number=10000))