|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Министерство науки и высшего образования РФ | | | | | |
| Федеральное государственное автономное | | | | | |
| образовательное учреждение высшего образования | | | | | |
| **«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»** | | | | | |
|  | | | | | |
| Институт космических и информационных технологий | | | | | |
| институт | | | | | |
| Программная инженерия | | | | | |
| кафедра | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
| **ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ** | | | | | |
| Понятие структуры в теории систем | | | | | |
| тема | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
| Преподаватель | |  |  |  | Л. М. Коренюгина |
|  | |  | подпись, дата |  | инициалы, фамилия |
| Студент | КИ23-16/1б, 032322546 |  |  |  | Е. А. Гуртякин |
|  | номер группы, зачётной книжки |  | подпись, дата |  | инициалы, фамилия |
|  | | | | | |
|  | | | | | |
| Красноярск 2025 | | | | | |

# ВВЕДЕНИЕ

## Цель работы

Закрепить понимание роли структуры в системах, научиться анализировать и проектировать структуры для решения практических задач в разработке программного обеспечения.

## Задачи

В рамках данной практической работы необходимо выполнить следующие задачи:

1. изучить теоретический материал по предложенной теме;
2. выполнить задания;
3. предоставить отчёт преподавателю.

## Задание 1. Анализ структуры системы

Выберите одну из следующих систем:

1. Онлайн-библиотека (например, LitRes);
2. Система контроля версий (например, Git);
3. Платформа для видеоконференций (например, Zoom).

Выполните следующее:

1. Определите элементы системы: Перечислите основные компоненты (подсистемы, модули, классы);
2. Опишите связи между элементами: Укажите, как элементы взаимодействуют друг с другом (например, вызов методов, обмен данными);
3. Определите тип структуры: Иерархическая, сетевая, матричная или другая. Обоснуйте свой выбор;
4. Проанализируйте свойства структуры:

Устойчивость: Как система behaves при удалении одного элемента?

Гибкость: Можно ли легко добавить новый элемент?

Эффективность: Насколько быстро данные/управление передаются между элементами?

Пример для Git:

Элементы: Репозиторий, коммиты, ветки, индекс, рабочий каталог.

Связи: Коммиты ссылаются на родительские коммиты, ветки ссылаются на коммиты.

Тип структуры: Направленный ациклический граф (DAG).

Устойчивость: Высокая — удаление одного коммита не ломает всю систему.

## Задание 2. Проектирование структуры

Спроектируйте структуру для системы «Умный дом». Система должна включать:

* Управление освещением;
* Контроль температуры;
* Систему безопасности (датчики движения, камеры);
* Голосовое управление.

Выполните следующее:

1. Разработайте иерархическую структуру системы: Определите уровни (например, уровень устройств, уровень управления, уровень пользователя).
2. Опишите связи между уровнями: Как данные и команды передаются между уровнями?
3. Спроектируйте интерфейсы для ключевых элементов.
4. Оцените гибкость структуры: Можно ли легко добавить новое устройство (например, умный чайник)?

Пример к пункту 3. Интерфейс для устройства «Умная лампа»: методы включить(), выключить(), изменитьЯркость().

## Задание 3. Сравнение структур

Сравните две архитектурные стили для веб-приложения:

1. Монолитная архитектура;
2. Микросервисная архитектура.

Выполните следующее:

1 Опишите структуру каждой архитектуры:

Как элементы связаны в монолите?

Как микросервисы взаимодействуют друг с другом?

2 Сравните по критериям:

Связность (Coupling): Насколько элементы зависят друг от друга?

Масштабируемость: Как архитектура ведёт себя при увеличении нагрузки?

Сложность разработки: Легко ли вносить изменения?

3 Сделайте вывод: Для какого типа проектов подходит каждая архитектура?

Пример вывода представлен ниже.

Монолит: Подходит для небольших проектов с простой логикой.

Микросервисы: Для сложных систем, где нужна независимость компонентов.

## Задание 4. Оптимизация структуры

В системе «Социальная сеть» обнаружена проблема: медленная загрузка ленты новостей из-за сложных запросов к базе данных.

Выполните следующее:

1. Предложите изменение структуры данных: Например, денормализация таблиц или использование кэша.
2. Предложите изменение структуры системы: Например, введение отдельного микросервиса для генерации ленты новостей.
3. Обоснуйте свой выбор: Как изменения улучшат производительность?

Пример решения:

Кэширование: Redis для хранения готовой ленты новостей.

Микросервис: Сервис NewsFeedService, который pre-generates ленту для каждого пользователя.

# ХОД РАБОТЫ

## Задание 1

Для выполнения задания была выбрана платформа для видеоконференций SberJazz.

Элементы системы:

Подсистемы: RTC поток данных аудио, RTC видео, Web-socket чат.

Модули: пользовательский интерфейс (графический интерфейс для использования платформы), управление конференцией (создание конференции, запись видео, ограничение доступа), обмен данными (передача сообщений, файлов, реакций, данных с экрана), безопасность (защита данных, шифрование, аутентификация), искусственный интеллект (транскрипция), мониторинг и аналитика (статистика о видеоконференции).

Связи между элементами представлены в таблице 1.

Таблица 1 – Связи в системе

|  |  |
| --- | --- |
| **Связь** | **Описание** |
| RTC видеопоток → Модуль обмена данных | Видеопоток передаёт закодированные видеоданные модулю обмена данных. Тот в свою очередь передаёт их всем участникам конференции. Также инициирует установку соединения и занимается отправкой метаданных. |
| RTC аудиопотока → Модуль обмена данных | Аудиопоток отправляет обработанные аудиоданные модулю обмена данных, который транслирует их участникам видеоконференции. |
| Модуль обмена данных → Подсистема видеопотока и Подсистема аудиопотока | Вызывает методы для установки подключения через потоки. Передаёт инструкции. |
| Модуль управления конференцией → Модуль обмена данных | Управление конференцией отправляет различные команды модулю обмена данных. Обмен данных подтверждает выполнение команд. |
| Модуль обмена данными → Модуль обмена данных и Модуль интерфейса пользователя | Модуль обмена данных передаёт текстовые или файловые сообщения через сигнализацию с целью маршрутизации. Также уведомляет пользователей о новых данных (например, через события). |
| Подсистема хранения данных → Модуль управления конференцией и Модуль мониторинга и аналитики | Хранилище принимает записи и логи от модуля управления и предоставляет данные мониторинга аналитике по запросу. |

Окончание таблицы 1

|  |  |
| --- | --- |
| **Связь** | **Описание** |
| Модуль безопасности → Все элементы (видеопоток, аудиопоток, сигнализация и т.д.) | Обеспечивает защищённый обмен данными путём шифрования данных перед передачей. Проверяет аутентификацию пользователей с помощью сигнализации. |
| Модуль искусственного интеллекта → Подсистема аудиопотока и Подсистема видеопотока | ИИ модуль получает аудио и видео данные для анализа (транскрипция). Возвращает данные в интерфейс. |
| Модуль интерфейса пользователя → Все элементы | Интерфейс отправляет команды пользователя (например, включить микрофон) через вызовы методов (REST API или WebSocket) ко всем соответствующим элементам (видеопоток, аудиопоток, управление) и отображает полученные данные (например, чат, видео). |
| Модуль мониторинга и аналитики → Модуль сигнализации, Подсистема хранения данных и Модуль интерфейса пользователя | Мониторинг собирает метрики от сигнализации и хранилища, формирует отчёты и уведомления, рассылает их через события. |

Визуально связи можно представить в виде произвольной диаграммы, представленной на рисунке 1.

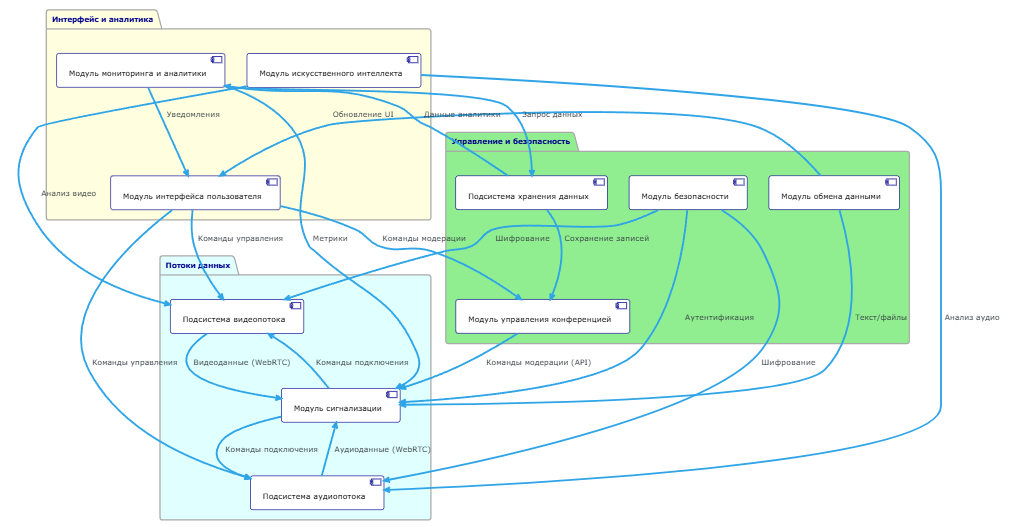


Рисунок 1 – Связи в системе

Тип структуры в платформе – Иерархический.

Такая система требует координации множества потоков данных. В таких системах требуется чётка иерархия и правила взаимодействия между элементами. В моём случае, модуль сигнализации выступает в роли координатора потоков данных и в роли управителя взаимодействием с другими командами.

На верхнем уровне находится модуль сигнализации. Ниже располагаются функциональные подсистемы и модули (например, подсистема видеопотока, модуль управления конференцией, модуль интерфейса пользователя), которые подчиняются сигнализации и выполняют специализированные задачи.

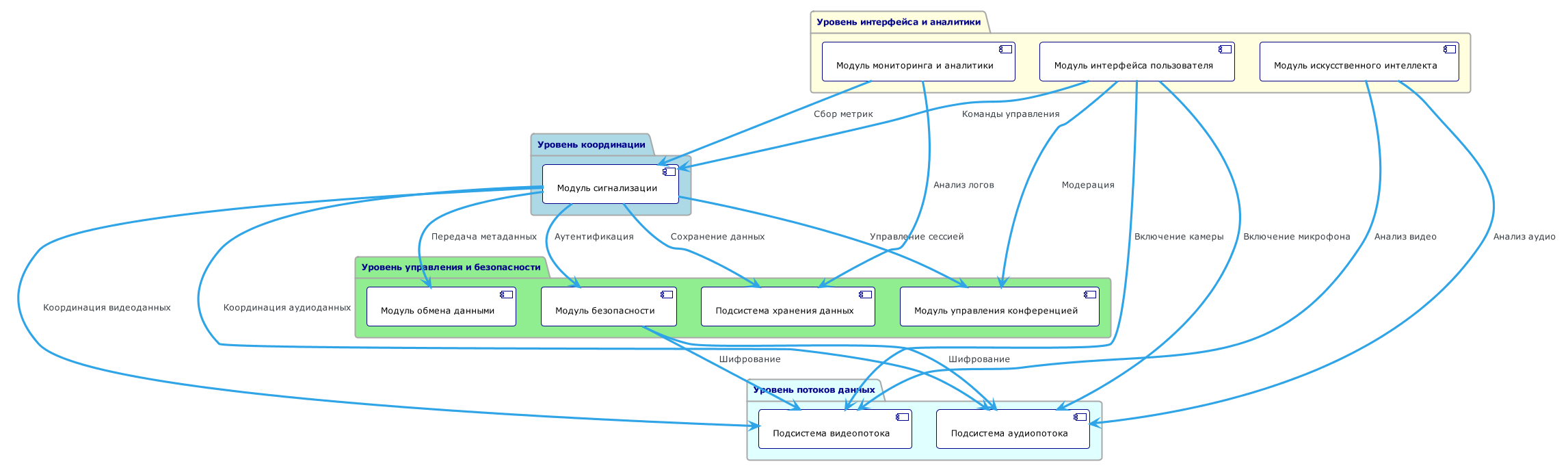


Рисунок 2 – Иерархия системы

Свойства структуры:

Устойчивость:

Удаление подсистемы видеопотока или аудиопотока. Система переключится в режим «только аудио» или «только видео».

Удаление модуля сигнализации. Критический фактор, ведущий к неработоспособности системы, по сколько он координирует работу других модулей.

Удаление модуля интерфейса пользователя. Система останется работоспособной для в данный момент проходящих трансляций. Будущие же участники воспользоваться платформой не смогут.

Удаление второстепенных элементов (например, модуля мониторинга). Система сохранит основную функциональность, но потеряет аналитику и уведомления.

При удалении ключевых элементов (модуль сигнализации) система приходит в неработоспособность. При удалении иных элементов, в частичную неработоспособность. Такие проблемы характерны в иерархических структурах. Проблему удаления любых элементов можно частично решить резервированием.

Гибкость:

Касаемо добавления нового элемента, возникнут умеренные сложности. Потребуется определить его роль в иерархии и правила взаимодействия с другими элементами. Чем выше уровень добавления, тем сложнее добавление.

Эффективность:

Передача данных. По протоколу WebRTC задержка будет составлять от 50 до 200 мс плюс накладные расходы на маршрутизацию через сигнализацию.

Управление. WebSocket и/или RestAPI и чёткая иерархия будут выдавать время отклика менее чем в 100 мс.

Проблема бутылочного горлышка. По скольку много данных маршрутизируется через сигнализатор, при большом числе людей в конференции могут возникнуть проблемы с качеством данных.

Таким образом, при малых и средних числах людей в конференции достигается наибольшая эффективность. При масштабировании возникают проблемы с производительностью из-за зависимости от сигнализации. Возможно решение в виде резервирования и кэширования.

## Задание 2

Разработаем иерархическую структуру системы. Сверху вниз будут идти следующие уровни: уровень пользователя, уровень управления, уровень устройств. Такой вид был выбран исходя из направления потоков данных.

В уровень пользователя входят: интерфейс управления системой, голосовой ассистент.

В уровень управления входят: центральный хаб, модуль обработки голосовых команд, модуль аналитики.

В уровень устройств входят: устройства освещения, сенсоры, датчики, система безопасности.

Связи между уровнями представлены в таблице 2.

Таблица 2 – Связи в уровнях системы

|  |  |
| --- | --- |
| **Связь** | **Описание** |
| Уровень пользователя → Уровень управления | Пользователь отправляет различные команды через приложение (REST API) или с помощью голосового ассистента (gRPC). |
| Уровень управления → Уровень устройств | Полученные от пользователя данные инициируют команды, отправляемые устройствам через специализированные протоколы (Zigbee, Wi-Fi, MQTT). |
| Уровень устройств → Уровень управления | Устройства отправляют данные о состоянии через MQTT или HTTP. |
| Уровень управления → Уровень пользователя | Хаб отправляет обновленные данные состояния, оповещения, голосовой ответ, команды на подтверждение действий. |

Интерфейсы для ключевых элементов представлены в таблице 3.

Таблица 3 – Интерфейсы ключевых элементов

|  |  |  |
| --- | --- | --- |
| **Элемент** | **Команда** | **Описание** |
| Умная лампа | turnOn() | Включить лампу. |
| turnOff() | Выключить лампу. |
| setBrightness(x) | Установить яркость в значение «x». 0% <= x <= 100% |

Продолжение таблицы 3

|  |  |  |
| --- | --- | --- |
| **Элемент** | **Команда** | **Описание** |
| Термостат | turnOn() | Включить термостат. |
| turnOff() | Выключить термостат. |
| getTemperature() | Получить данные о температуре. |
| setTemperature(x) | Установить температуру в уровень «x» градусов системы, выбранной в настройках.  -275 оС <= x если иное на задано термостатом. |
| enableHeating() | Включить обогрев до целевой температуры. |
| disableHeating() | Выключить обогрев до целевой температуры. |
| Датчик движения | turnOn() | Включить датчик движения. |
| turnOff() | Выключить датчик движения. |
| getMotionStatus() | Получить данные о движении (true / false). |
| setSensitivity(x) | Установить чуствительность датчика в положение «x».  0 < x. |
| triggerAllert() | Переключить сигнализацию в режим опасности. |
| Умная камера | turnOn() | Включить камеру. |
| turnOff() | Выключить камеру. |
| setResolution(x) | Установить качество видео.  «x» – разрешение в формате «ширинаХвысота» в пикселях. |
| setFrameRate(x) | Установить кол-во кадров в секунду при записи на «x».  «x» – кол-во кадров. |
| startRecording() | Начать запись |
| stopRecording() | Остановить запись. Сохранить запись. Отправить запись в хаб. |
| getLiveFeed() | Получить трансляцию с камеры в реальном времени. |
| getSnapshot() | Сделать снимок с камеры. Сохранить снимок. Отправить снимок в хаб. |

Окончание таблицы 3

|  |  |  |
| --- | --- | --- |
| Центральный хаб | getStatus() | Получить информацию о состоянии устройств. |
| updateStatus() | Обновить информацию о состоянии устройств. |
| sendCommand(device, command) | Отправить на устройство «device» команду «command». |
| recieveData(device, data) | Получить с устройства «device» данные «data». |
| processVoiceCommand(command) | Выполнить голосовую команду «command». |

Данная система обладает умеренной гибкостью. Что касается подключения умного чайника, интерфейс хаба (sendCommand()) позволяет интегрировать устройство с методами вроде boilWater(), getWaterLevel(), что требует только обновления конфигурации хаба. Это относительно просто, если протоколы совместимы. Если чайник использует новый протокол (например, нестандартный IoT-стандарт), потребуется адаптация хаба (уровня управления), что увеличивает сложность.

## Задание 3

Рассмотрим монолитную архитектуру.

По структуре, компоненты монолита (презентационный слой, бизнес логика, доступ к данным) объединяются в единый исполняемый файл или процесс.

По связи, в монолите в рамках одного приложения компоненты тесно интегрированы. Презентационный слой вызывает методы бизнес-логики напрямую. Та в свою очередь обращается к базе данных через встроенные модули (например, ORM). Обмен данными происходит через общую память или вызовы функций, без внешних интерфейсов.

Такие системы подходят для небольших, средних решений из-за быстроты разработки и развёртывания.

Пример. Онлайн-магазин, где обработка заказов, каталог и авторизация реализованы в одном проекте (с помощью веб-фреймворка Django).

Рассмотрим микросервисную структуру. В такой структуре приложение разбивается на независимые сервисы. Каждый сервис отвечают за отдельную функцию приложения. Например, авторизацию, каталог, платежи. Каждый микросервис имеет соётвенную базу данных и запускается как отдельный процесс или контейнер.

Связи в микросервисах. Сервисы обмениваются данными через легковесные протоколы, такие как REST API, gRPC или сообщения (Kafka, RabbitMQ). Координация может осуществляться через API-шлюз или оркестратор (Kubernetes).

Сравнение двух структур представлено в таблице 4.

Таблица 4 – Сравнение монолитных и микросервисных структур

|  |  |  |  |
| --- | --- | --- | --- |
| **Структура** | **Связность** | **Масштабируемость** | **Сложность разработки** |
| Монолит | Высокая. Все модули (UI, логика, данные) зависят друг от друга. Изменение одной части требуют пересборки всего приложения. | Ограниченная. При увеличении нагрузки потребуется масштабировать весь монолит, что не всегда является эффективным с точки зрения распределения нагрузки на ресурсы. Если нагрузка приходится на одну функцию (например, авторизацию), это создаёт лишние накладные расходы на поддержание неиспользуемой структуры. | Низкая на начальном этапе. Все этапы разработки происходят в единой среде. Однако с развитием проекта внесение изменений будет требовать больших усилий. |
| Микросервис | Низкая. Каждый сервис изолирован и может развиваться независимо в случае сохранения API контракта с другими сервисами. | Высокая. Масштабируется только та часть сервисов, на которые приходится избыточная нагрузка. Существуют такие платформы, как Kubernetes, который позволяет в автоматическом режиме разворачивать или убирать сервисы в зависимости от приходящейся на них нагрузки. | Высокая. Требуются отдельные команды для каждого сервиса, сложная настройка (CI/CD, мониторинг). В сравнении с монолитными системами, распределённые системы куда сложнее отлаживать. При условии сохранения API контракта изменения внутри сервиса останутся локальными по отношению к нему. |

## Задание 4

Предложим решение, связанное с изменением структуры данных.

Проведём денормализацию таблиц путём создания новой, в которой будет хранится предсгенерированные ленты новостей, обновляемые по триггеру или событию в асинхронном режиме. Ко всему вышеперечисленному добавим кэширование с помощью Redis. Лента будет хранится в кэше 5 минут и обновляться в случае появления новых постов.

Таким образом, нагрузка на базу данных снизится, запросы для получения данных упростятся, кэширование возьмёт повторяющиеся запросы к данным на себя.

Предложим решение, связанное с изменением структуры системы.

Добавим микросервис, который будет генерировать и обслуживать ленту новостей. Назовём его NewsFeedService. Этот сервис в фоновом режиме обновляет ленты пользователей путём получения событий о небходимости обновления из, например, Apache Kafka. Он взаимодействует с основным сервисом БД через API или сообщения, хранит ленту в Redis.

Основное приложение вызывает NewsFeedService через gRPC или REST API для получения ленты. События же публикуются в Kafka для асинхронной обработки.

Такой подход позволяет масштабировать только микросервис. С ростом нагрузки можно будет развернуть больше таких сервисов. Благодаря этому, сервис перестаёт большую часть времени пребывать в перегруженном состоянии и уменьшает время отклика для получения лент.

# ЗАКЛЮЧЕНИЕ

По результатам работы был изучен теоретический материал по теме «Понятие структуры в теории систем». Все поставленные цели и задачи были выполнены. Задания были выполнены и помогли лучше усвоить пройденный материал.