2025;20(1):84-104 Информация и инновации / Information and Innovations

Информационные процессы / Information Processes

Оригинальная статья / Original article
УДК 004.021
https://doi.org/10.31432/1994–2443.2025.02

Проектирование системы операционной поддержки сети связи на основе микросервисной архитектуры

А.В. Синицын, А.А. Замошников ✉,
С.А. Селиванов, С.Я. Прокофьев, И.В. Тузов
Федеральное государственное бюджетное образовательное учреждение высшего образования «МИРЭА — Российский технологический университет»
Российская Федерация, г. Москва, проспект Вернадского, 78, 119454,
✉ *zamoshnikov81@mail.ru*

Аннотация. *Актуальность.* Развитие информационных технологий привело к значительному увеличению объемов передаваемых данных, что в свою очередь создает серьезные вызовы для телекоммуникационных сетей. *Цель.* Проектирование архитектуры системы операционной поддержки сети связи, а также выбор целесообразных и актуальных технологий для дальнейшей разработки системы. *Материалы и методы.* Распределенный сервис Apache Zookeeper, распределенная потоковая платформа Apache Kafka, открытая объектно-реляционная СУБД PostgreSQL, язык программирования Java, фреймворк Spring, фреймворк для создания веб-приложений Hilla, открытый формат для хранения и передачи географических данных GeoJSON, библиотека для визуализации данных D3.js, библиотека JavaScript React. *Результаты.* Разработана система, базирующаяся на клиент-серверной архитектуре и микросервисном взаимодействии, которая позволяет эффективно управлять сетевыми ресурсами, минимизировать задержки и адаптироваться к динамичным изменениям в сети. *Выводы.* Использование современных технологий как на стороне клиента, так и на стороне сервера, обеспечивает высокую производительность и масштабируемость решения, что делает его перспективным для дальнейшего развития в условиях стремительно растущих требований информационной инфраструктуры.

Ключевые слова: микросервис, архитектура, маршрут, технологии, интеграция, узлы

Финансирование. Финансирование отсутствовало.

Для цитирования: Синицын А.В., Замошников А.А., Селиванов С.А., Прокофьев С.Я., Тузов И.В. Проектирование системы операционной поддержки сети связи на основе микросервисной архитектуры. *Информация и инновации*. 2025;20(1):84-104. https://doi.org/10.31432/1994–2443.2025.02

Design of the communication network operational support system based on microservice architecture

Alexander V. Sinitsyn, Alexander A. Zamoshnikov ✉,
Sergey A. Selivanov, Sergey Ya. Prokofiev, Ivan V. Tuzov
Federal State Budgetary Educational Institution of Higher Education
"MIREA — Russian Technological University"
Vernadsky Avenue, 78, Moscow, 119454, Russian Federation
✉ *zamoshnikov81@mail.ru*

Abstract. *Relevance.* The development of information technology has led to a significant increase in the volume of transmitted data, which in turn creates serious challenges for telecommunications networks. *Goal.* Designing the architecture of the communication network operational support system, as well as selecting appropriate and relevant technologies for further system development. *Materials and methods.* Apache Zookeeper distributed service, Apache Kafka distributed streaming platform, PostgreSQL open object-relational database, Java programming language, Spring framework, Hilla web application framework, GeoJSON open format for storing and transmitting geographical data, D3 data visualization library.js, the React JavaScript library. *Results.* The built system, based on a client-server architecture and micro-service interaction, allows efficient management of network resources, minimizes delays and adapts to dynamic changes in the network. *Conlusion.* The use of modern technologies on both the client and server sides ensures high performance and scalability of the solution, which makes it promising for further development in the face of rapidly growing information infrastructure requirements.

Keywords: microservice, architecture, route, technologies, integration, nodes

Funding. No funding.

For citation: Sinitsyn A.V., Zamoshnikov A.A., Selivanov S.A., Prokofiev S.Ya., Tuzov I.V. Design of the communication network operational support system based on microservice architecture. *Information and Innovations.* 2025;20(1):84-104. (In Russ.). https://doi.org/10.31432/1994–2443.2025.02

ВВЕДЕНИЕ

Развитие информационных технологий привело к значительному увеличению объемов передаваемых данных, что в свою очередь создает серьезные вызовы для телекоммуникационных сетей. Основные проблемы, такие как перегрузка каналов, задержки в передаче данных и недостаточная отказоустойчивость, требуют внедрения актуальных подходов в маршрутизации сетевого траффика. Эффективное построение маршрутов является ключевым моментом в оптимизации работы сетей, позволяющим минимизировать задержки, снизить потери данных и распределить нагрузку между узлами. Процесс построения маршрутов является довольно трудозатратным для операторов телекоммуникационных сетей. В связи с этим, стоит цель оптимизировать данный процесс путём перекладывания вычислительных операций на автоматизированные системы управления, использующие алгоритмы, решающие задачу построения графа-маршрута. Поэтому актуальной становится разработка автоматизированных систем управления, которые способны адаптивно решать задачи построения оптимальных маршрутов [1].

Целью данного исследования является проектирование архитектуры системы операционной поддержки сети связи, а также выбор целесообразных и актуальных технологий для дальнейшей разработки системы.

МАТЕРИАЛЫ И МЕТОДЫ

На серверной стороне система CNOSS опирается на проверенные и масштабируемые решения:

Apache Zookeeper — это распределенный сервис для управления конфигурацией, синхронизации и координации распределенных систем. Он предоставляет простой и надежный способ управления распределенными приложениями, обеспечивая согласованность данных и координацию между узлами.

Apache Kafka — это распределенная потоковая платформа, предназначенная для обработки, хранения и передачи больших объемов данных в реальном времени. Она широко используется для построения масштабируемых, отказоустойчивых и высокопроизводительных систем обработки данных. Для хранения метаданных Kafka-кластера используют Zookeeper, но следует упомянуть, что начиная с версии 2.8.0 у Kafka появляется режим Kafka Raft Metadata mode (KRaft), что делает использование Zookeeper опциональным [2].

PostgreSQL — открытая объектно-реляционная СУБД, отвечающая принципам ACID и поддерживающая хранение JSON-объектов, используемых для хранения маршрутов [3].

Java — высокоуровневый объектно-ориентированный язык программирования с строгой статической типизацией, поддерживающий принципы WORA («Write Once, Run Anywhere») благодаря исполнению на JVM (Java Virtual Machine). Широко применяется в enterprise-разработке, Big Data (Hadoop, Spark), Android и серверных приложениях.

Spring — модульный фреймворк для Java, реализующий паттерны Inversion of Control (IoC) и Dependency Injection (DI). Предоставляет подпроекты (Spring Boot, Spring MVC, Spring Security, Spring Data), ускоряющие разработку масштабируемых back-end приложений с поддержкой микросервисной архитектуры.

Hilla — фреймворк, интегрирующий Java-бэкенд с React-фронтендом через TypeScript. Использует реактивную модель данных и автоматическую генера-

цию API (на основе аннотаций Java), что сокращает время разработки CRUD-приложений.

GeoJSON — стандартизированный формат обмена геопространственными данными, основанный на JSON. Поддерживает геометрические примитивы (Point, LineString, Polygon) и их коллекции, применяется в GIS (Geographic Information Systems) и веб-картографии.

D3.js — JavaScript-библиотека для динамической визуализации данных на основе веб-стандартов (SVG, Canvas, CSS). Реализует декларативный подход к привязке данных к DOM, обеспечивая высокую гибкость в построении сложных диаграмм, карт и интерактивных дашбордов.

React — декларативная JavaScript-библиотека для построения компонентных пользовательских интерфейсов. Использует Virtual DOM для оптимизации рендеринга и поддерживает односторонний поток данных (unidirectional data flow), что упрощает управление состоянием приложения (в сочетании с Redux или Context API).

РЕЗУЛЬТАТЫ И ОБСУЖДЕНИЕ

1. Постановка задачи

Основной задачей данного исследования является разработка системы автоматизированного построения маршрутов в телекоммуникационных сетях, способной адаптироваться к динамическим изменениям параметров сети и обеспечивать эффективное управление сетевыми ресурсами. Система должна уметь строить маршрут, чтобы минимизировать временные задержки, снизить потери данных и балансировать нагрузку между узлами сети.

Для достижения этой цели необходимо решить следующие подзадачи:

1. Разработка архитектуры системы на основе микросервисного взаимодействия, обеспечивающей гибкость, отказоустойчивость и масштабируемость решения.
2. Создание механизма маршрутизации запросов с разделением авторизованного и неавторизованного трафика, а также проверкой валидности данных авторизации.
3. Разработка клиентского веб-приложения с удобным интерфейсом для управления маршрутами и конфигурацией сети. Интеграция с системами хранения данных и обработки потоков информации, такими как PostgreSQL, Apache Kafka и Zookeeper, для обеспечения надежности и согласованности данных.
4. Обеспечение поддержки масштабируемости и автоматизированного обновления конфигурации системы без необходимости остановки сервиса. Решение поставленной задачи должно учитывать требования операторов телекоммуникационных сетей и предоставлять возможность кастомизации системы под индивидуальные потребности.

2. Архитектура системы CNOSS

В условиях импортозамещения и необходимости доработки существующих решений, таких как Elastic Services Platform, было принято решение о разработке собственной системы операционной поддержки сети связи (CNOSS — Communication Network Operation Support System). Основной особенностью CNOSS является возможность адаптировать систему под специфические требования оператора [4, 5].

2.1 Клиент-серверная архитектура и микросервисное взаимодействие

Система CNOSS реализована по принципу клиент-серверной архитектуры

с разделением логики на несколько микросервисов. При разработке системы автоматизированного построения маршрутов в телекоммуникационных сетях выбор архитектурного подхода играет ключевую роль. Микросервисная архитектура в данном случае является более предпочтительной по сравнению с монолитной по нескольким причинам.

Во-первых, каждый сервис можно масштабировать независимо. Например, если возрастает нагрузка на сервис аутентификации, его можно масштабировать отдельно от других компонентов системы. В монолите же увеличение нагрузки на один из модулей вынуждает масштабировать всё приложение целиком, что приводит к избыточному расходу ресурсов.

Во-вторых, отказ одного микросервиса не приводит к отказу всей системы. Остальные сервисы продолжают функционировать. В монолите же ошибка в одном модуле может привести к отказу всего приложения.

В-третьих, микросервисный подход позволяет распределять ответственность между людьми в команде, каждый из которых разрабатывает и поддерживает отдельный сервис. Это ускоряет разработку и улучшает читаемость кода. В монолите же разработка усложняется из-за высокой связанности компонентов и зависимости между разработчиками.

В-четвертых, обновление отдельных микросервисов можно выполнять без остановки всей системы. Каждый сервис можно тестировать и запускать независимо, что сокращает время вывода новых функций на рынок. В монолите же тестирование и развертывание требуют больше времени и ресурсов.

На рис. 1 представлены все компоненты системы CNOSS.

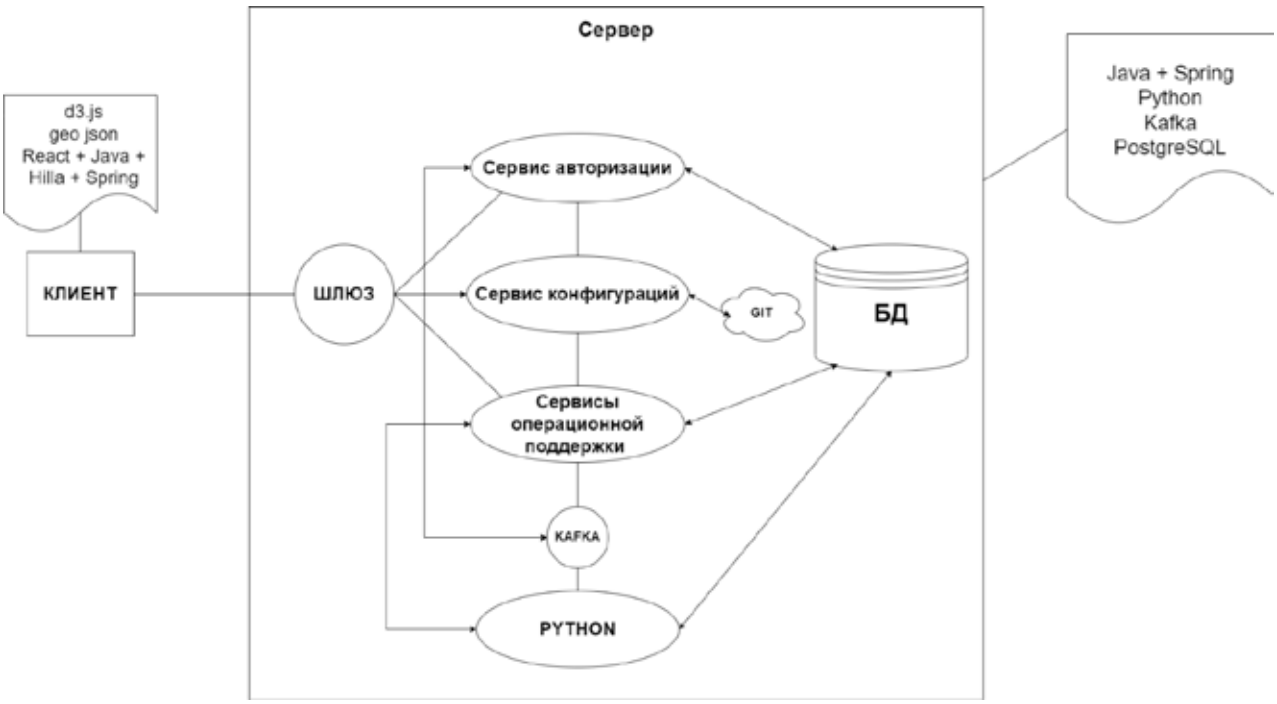

Рис. 1. Компоненты системы CNOSS
Fig. 1. Components of the CNOSS system

2.2 Шлюз

Шлюз — это сервис маршрутизации запросов с клиентской части, который определяет внутренний микросервис для обработки запроса, разделяет авторизованный и неавторизованный трафик, а также проверяет валидность данных авторизации. С помощью данного шлюза можно добавлять новые интеграции без дополнительной настройки авторизации на каждой из интеграций. Достигается путём перенаправления входящего в серверную часть трафика через сервис [6]. Помимо этого, шлюз ограничивает запросы извне, пропуская только допустимые установленные запросы. Сервис написан с использованием библиотеки Spring Cloud Gateway.

Для реализации логики разделения был написан фильтр аутентификации. В его задачи входит проверка валидности JWT-токена путём запроса на сервис авторизации. В случае отсутствия токена запрос отклоняется. В Листинге 1 отражен код фильтра, проверяющий авторизацию входящего запроса.

Листинг 1 — код фильтра для проверки авторизации входящего запроса:

```
public class AuthenticationGatewayFilterFactory extends AbstractGatewayFilterFactory<Aut
henticationGatewayFilterFactory.Config>{

public GatewayFilter apply(Config config) {
 return (exchange, chain) -> {
  //если авторизация выключена, то пропускаем фильтр
  if (!config.securedEnabled) {
   return chain.filter(exchange);
}
  //получение токена из header'a
     String bearerToken = exchange.getRequest().getHeaders().getFirst(Constant.HEADER_
AUTHORIZATION);
  //если токен не найден — вернуть ошибку аутентификации
  if (bearerToken == null) {
     return onError(exchange, HttpStatus.OK, "Ошибка аутентификации", "AUTH: ERROR",
     «Нет токена авторизации»);
}
  //вызов сервиса авторизации для проверки jwt-токена
  return webClientBuilder.build().get()
.uri(url + endpoint)
.header(Constant.HEADER_AUTHORIZATION, bearerToken)
.retrieve()
.bodyToMono(JwtValidationRs.class)
.map(response -> {
        //если проверка jwt-токена не пройдена — вернуть ошибку авторизации
        if (!response.isSuccess()) {
           throw new ResponseValidationException(response);
}
        //если проверка успешна — дополнить запрос данными о клиенте
        exchange.getRequest().mutate().header("Id", response.getId());
        exchange.getRequest().mutate().header("Email", response.getEmail());
        exchange.getRequest().mutate().header("FirstName", response.getFirstName());
```

```
        exchange.getRequest().mutate().header("LastName", response.getLastName());
        exchange.getRequest().mutate().header("MiddleName", response.getMiddleName());

        return exchange;
})
.flatMap(chain:: filter)
};
}
```

Данный фильтр добавляется в файле конфигурации путём добавления в секцию Filters.

Ниже представлен Листинг 2 — запрос на создание маршрута с включенным фильтром авторизации:

```
- id: create-route
    #адрес сервиса для переадресации
     uri: http://localhost:8080
     predicates:
     #эндпойнт
      - Path=/ntwrk-prtnl-spprt-sstm-mddlwr/createRoute
     filters:
     #фильтр авторизации
      - Authentication=true
```

2.3 Сервис аутентификации

Сервис аутентификации отвечает за регистрацию, аутентификацию и авторизацию пользователей. Он генерирует и проверяет JWT-токены на основе пользовательских данных. JWT (JSON Web Token) — это специальный ключ, который подтверждает, что пользователь авторизован. Его используют сайты и сервисы, чтобы не спрашивать логин и пароль при каждом запросе. Токен состоит из трёх частей, соединённых точками:

1. Заголовок — указывает, какой алгоритм использован для подписи.
2. Полезная информация — содержит пользовательские данные и срок действия токена.
3. Подпись — используется для проверки подлинности токена, создаётся с помощью секретного ключа.

Пример JWT-токена в системе:

eyJhbGciOiJIUzI1NiJ9.eyJsYXN0X25hbWUiOiJhZG1pbiIsImlkIjoiYzBhODU1MDEtOTU0OC0xNDdlLTgxOTUtNDhkNDM5NzcwMDAyIiwibWlkZGxlX25hbWUiOiJhZG1pbiIsImZpcnN0X25hbWUiOiJhZG1pbiIsInN1YiI6ImFkbWluIiwiaWF0IjoxNzQwNjgzNjIyLCJleHAiOjE3NDA4Mjc2MjJ9.0m9cdoogoUh-uF9MeOnAGHSJgKKpj6fBcyw3OXlygfU

Расшифровка токена:

- eyJhbGciOiJIUzI1NiJ9 – {"alg":"HS256"} — алгоритм шифровки данных;
- eyJsYXN0X25hbWUiOiJhZG1pbiIsImlkIjoiYzBhODU1MDEtOTU0OC0xNDdlLTgxOTUtNDhkNDM5NzcwMDAyIiwibWlkZGxlX25hbWUiOiJhZG1pbiIsImZpcnN0X25hbWUiOiJhZG1pbiIsInN1YiI6ImFkbWluIiwiaWF0IjoxNzQwNjgzNjIyLCJleHAiOjE3NDA4Mjc2MjJ9 — {"last_name":"admin","id":"c0a85501–9548–147e-8195–48d439770002","middle_name":"admin","first_name":"admin","sub":"admin","iat":1740683622,"exp":1740827622} — данные о пользователе;
- 0m9cdoogoUh-uF9MeOnAGHSJgKKpj6fBcyw3OXlygfU — зашифрованная подпись.

2.4 Сервис агрегации данных

Сервис агрегации данных — это бизнес-сервис, предоставляющий эндпойнты (Эндпойнт — точка доступа приложения, к которой можно обратиться для выполнения нужного действия или получения данных) для работы со справочной информацией об актуальном состоянии системы и агрегирующий посчитанные маршруты для дальнейшей их обработки [7–9].

Задачами сервиса агрегации данных являются:

1. Создание, изменение и удаление информации о точках, бакетов и плат системы.
2. Работа с клиентскими данными, а также предоставление клиенту в пользование портов в платах для его маршрутов.
3. Управление активными маршрутами клиентов.
4. Ведение и подсчет статистики по нагруженности системы.

Сервис построен на принципе событийной модели. Основной сущностью, с которой работает сервис, является сущность Task. С помощью неё агрегируется информация по запросам пользователя и контролируется очерёдность построения маршрутов. Пример создания такой задачи в многопоточной среде можно увидеть в Листинге 3. Класс DistributedLock выполняет задачу блокировки ресурсов, что позволяет при одновременной попытке создания нескольких задач разными инстансами приложения, создать только одну задачу.

Листинг 3 — код создания задачи на расчёт маршрутов:

```
public TaskEntity createTaskWithLock(UUID clientId) {
    //блокировка процесса создания задачи с помощью Apache Zookeeper
      try (DistributedLock lock = new DistributedLock(curatorFramework, Constant.TASK_
LOCK_CODE, waitTime)) {
      //поиск активной задачи в базе
      TaskEntity taskEntity = taskRepository.findByActiveFlagTrue();
      //если есть активная задача — прекратить обработку
      if (taskEntity!= null) {
        throw new TaskException("Найдена активная задача. Попробуйте позже");
}
      //иначе — создание активной задачи
      return taskRepository.save(new TaskEntity()
.setCreatedTime(LocalDateTime.now())
.setActiveFlag(true)
.setClientId(clientId));
} catch (Exception e) {
      log.error(«Ошибка при попытке открыть задачу», e);
     throw new TaskException(«При попытке открыть задачу произошла ошибка. Попробуйте позже»);
}
}
```

Процесс, выполняемый данным примером кода также отображён на диаграмме последовательности, представленной на рис. 2, начиная с этапа «Создание файла задачи» и заканчивая этапом «Файл задачи удалён».

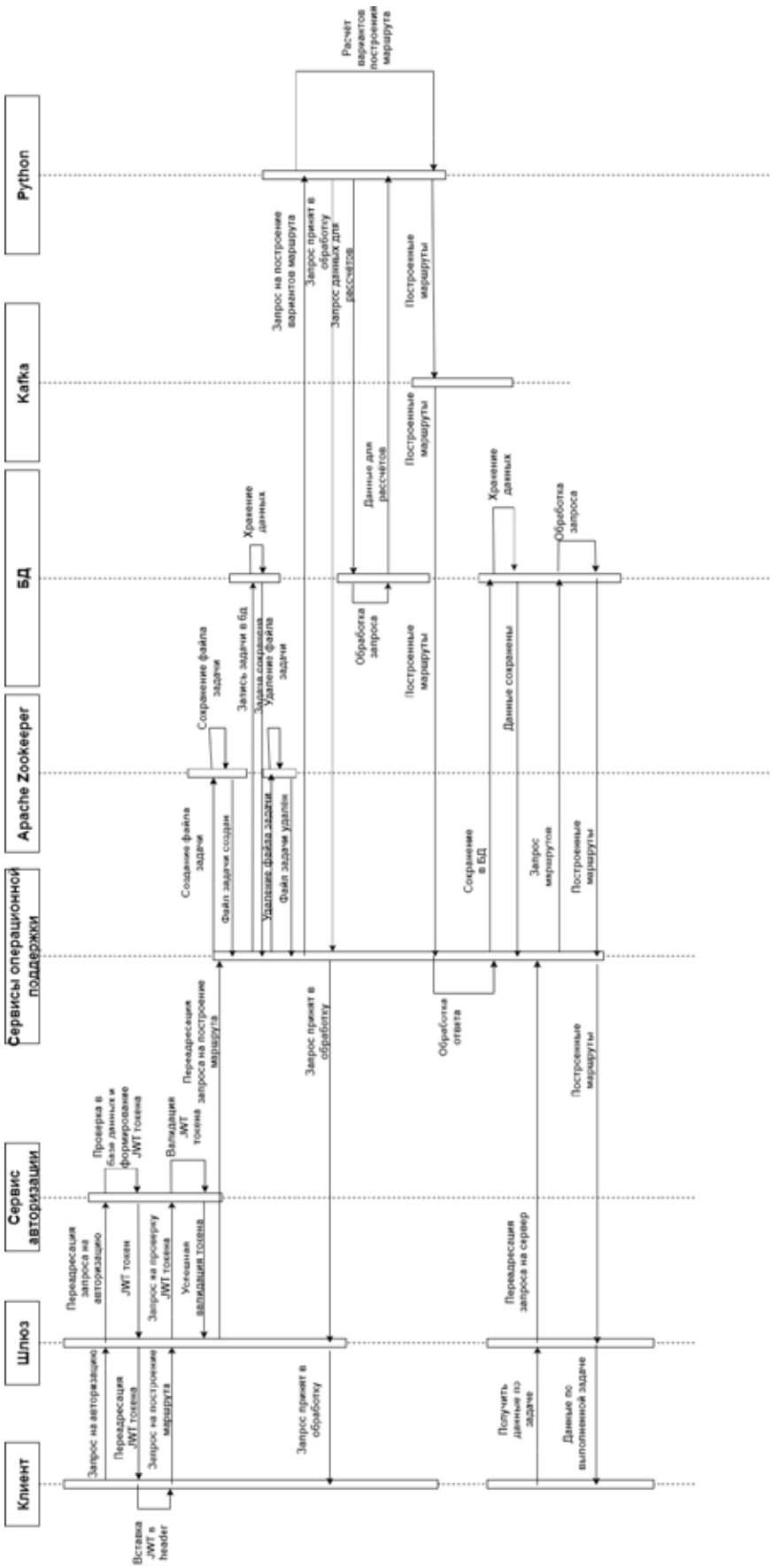

Рис. 2. Диаграмма последовательности работы системы CNOSS
Fig. 2. Diagram of the CNOSS system operation sequence

2.5 Сервис конфигурации

Сервис конфигурации — сервис, предназначенный для получения актуальных настроек работы микросервисов системы. При старте приложения каждый микросервис обращается в сервис конфигурации для получения настроек работы. При изменении их в системе управления версиями (например, Git), сервис конфигурации уведомляет связанные компоненты посредством брокера сообщений Apache Kafka, вынуждая их актуализировать свои настройки (рис. 3).

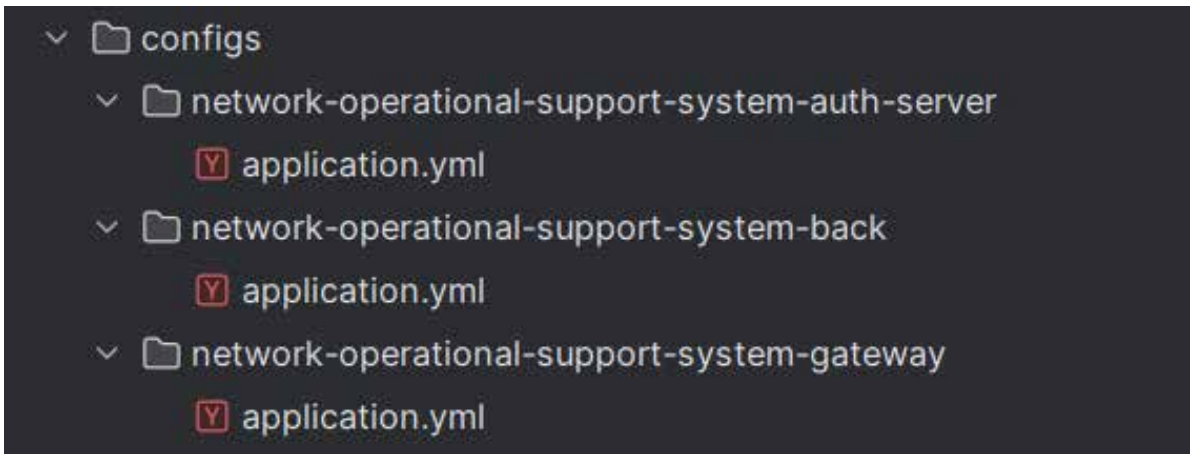

Рис. 3. Структура хранения конфигураций на сервисе Git
Fig. 3. Configuration storage structure on the Github service

В Листинге 4 представлен пример конфигурации сервиса авторизации.

Листинг 4 — пример конфигурации сервиса авторизации:

```
spring:
 jackson:
  date-format: yyyy-MM-dd HH: mm: ss
 #параметры подключения к базе данных
 datasource:
  url: jdbc: postgresql://localhost:5432/db
  username: user
  password: postgres
 cloud:
  stream:
   kafka:
    #параметры подключения к kafka
    binder:
     brokers:
      - localhost:19092
      - localhost:29092
#токен для формирования jwt
token:
 signing:
```

```
    key: bmV0d29yay1vcGVyYXRpb25hbC1zdXBwb3J0LXN5c3RlbS1hdXRoLXRvcC1zZWNyZXQ=
#доступные точки для запросов в сервис
controller:
 jwt:
  validation: /validation
 auth:
  sign-up: /sign-up
  sign-in: /network-operational-support-system-api/auth
```

2.6 Сервис построения маршрутов

Сервис построения маршрутов — сервис, предназначенный для проведения расчёта маршрутов с помощью адаптивных алгоритмов. Написан на языке программирования Python.

3. Клиентская часть

Клиентская часть системы представлена веб-приложением, разработанным с использованием современных и актуальных технологий. Они широко применяются в корпоративной разработке, имеют актуальную поддержку, а также кроссплатформенность — они могут работать на разных платформах (ОС Windows, Linux, MacOS).

3.1 Java и Spring

Java — это кроссплатформенный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (ныне принадлежащей Oracle). Программы, написанные на Java, компилируются в байт-код, который выполняется виртуальной машиной Java (JVM), что позволяет запускать код Java-приложения на любых устройствах и операционных системах [10, 11].

Maven — это инструмент управления зависимостями и автоматизации сборки проектов, созданный Apache Software Foundation. Maven в связке с Java сильно упрощает процессы сборки и упаковки приложений в JAR (JAR — Java Archive — архивный файл, в котором содержится часть программы на языке Java), а также позволяет управлять многомодульными проектами без особых усилий.

Spring — это фреймворк для разработки enterprise-приложений на Java. Он предоставляет комплексную инфраструктуру для создания современных, масштабируемых и легко поддерживаемых приложений. Spring облегчает процесс программирования благодаря применению принципов инверсии контроля (IoC) и внедрения зависимостей (DI), а также предоставляет разнообразные модули для выполнения различных задач.

Представленные технологии используются для построения серверной логики и API, обеспечивая высокую масштабируемость и поддержку принципов инверсии контроля (IoC) и внедрения зависимостей (DI).

3.2 Hilla

Hilla — современный фреймворк для создания полноценных веб-приложений с использованием Java (на сервере) и TypeScript (на клиенте). Он обеспечивает удобное взаимодействие между React-компонентами и серверной частью на Spring Boot. Благодаря данному фреймворку возможно обеспечить удобное и быстрое взаимодействие между микросервисами серверной части и клиентской частью, который развёрнут с использованием Spring Boot. В Листинге 5 представлен метод авторизации пользователя, в котором в объект передачи данных записывается логин и пароль пользователя,

после чего происходит обращение к сервису аутентификации по прописанному эндпойнту (Листинг 6). Данный эндпойнт вызывается на странице авторизации при отправлении формы авторизации (Листинг 7).

Листинг 5 — эндпойнт для получения доступа токену аутентификации:

```
@BrowserCallable
@AnonymousAllowed
@RequiredArgsConstructor
public class AuthenticationEndpoint {

   private final AuthorizationService authorizationService;

   public SignInJwtDto auth() {
      SignInDto dto = new SignInDto();
      dto.setLogin("admin");
      dto.setPassword("admin");
      SignInJwtDto auth = authorizationService.auth(dto);
      System.out.println(auth.getToken());
      return auth;
}
}
```

Листинг 6 — сервис для обращения к сервису аутентификации:

```
@FeignClient(url = "http://localhost:8085/network-operational-support-system-api", value =
"authorization")
public interface AuthorizationService {

   @PostMapping(value = "/auth")
   SignInJwtDto auth(@RequestBody SignInDto dto);
}
```

Листинг 7 — обращение к эндпойнту и получение токена в React-компоненте:

```
const serverResponse = await AuthenticationEndpoint.auth();
     if (serverResponse.token) {
       localStorage.setItem("token", serverResponse.token);
} else {
      console.error(«Ошибка: токен отсутствует!»);
}
```

На рис. 4 представлена форма авторизации системы.

Процесс авторизации также отображён на диаграмме последовательности на рис. 2, начиная с этапа «Запрос на авторизацию» и заканчивая этапом «Переадресация JWT-токена».

3.3 React и TypeScript

React используется для разработки динамичных и отзывчивых пользовательских интерфейсов, а TypeScript обеспечивает статическую типизацию, что повышает надежность кода [12].

Рис. 4. Форма авторизации системы
Fig. 4. System authorization form

В Листинге 8 представлен переиспользуемый компонент ввода пароля, который используется на страницах авторизации и регистрации (см. рис. 4). Данный компонент присутствует в процессе запроса авторизации, представленной на диаграмме последовательности на рис. 2.

Листинг 8 — переиспользуемый компонент для ввода пароля пользователя:

```
import {useState} from 'react';
import styles from 'Frontend/components/CustomInput/CustomPasswordInput.module.css';
import {Eye, EyeOff} from 'lucide-react';

const CustomPasswordInput = () => {
 const [showPassword, setShowPassword] = useState(false);

 return (
  <div className={styles.passwordDiv}>
   <h5>Пароль</h5>
   <div className={styles.inputWrapper}>
    <input
     type={showPassword? "text": "password"}
```

```
        name="password"
        className={styles.passwordInput}
      />
      <button
        type="button"
        onClick={() => setShowPassword((prev) =>!prev)}
        className={styles.toggleButton}
      >
        {showPassword? <EyeOff size={18} />: <Eye size={18} />}
      </button>
    </div>
  </div>
);
}

export default CustomPasswordInput;
```

3.4 D3.js

D3.js — библиотека для визуализации данных, которая применяется для отображения карт и построенных графов маршрутов. Благодаря использованию SVG, D3.js позволяет создавать насыщенные графические структуры с анимацией и интерактивностью [13].

В Листинге 9 представлено графическое отображение карты с графом (рис. 5) с помощью библиотеки D3.js:

```
import React, {useEffect, useRef, useMemo} from "react";
import {select, zoom, ZoomBehavior, geoPath, geoNaturalEarth1} from "d3";

type Point = {id: number; name: string; coordinates: [number, number]};
type Edge = {source: number; target: number};

type MarksProps = {
  data: any; // GeoJSON для карты
  points: Point[];
  edges: Edge[];
};

export const Marks = ({data, points, edges}: MarksProps) => {
  const gRef = useRef<SVGGElement | null>(null);
  const svgRef = useRef<SVGSVGElement | null>(null);

  useEffect(() => {
    if (!data) return;

    const svg = select(svgRef.current!);
```

```
  const g = select(gRef.current!);

  const zoomBehavior: ZoomBehavior<Element, unknown> = zoom()
.scaleExtent([1, 8])
.on("zoom", (event) => {
    g.attr("transform", event.transform);
});

  svg.call(zoomBehavior as any);
}, [data]);

 const projection = useMemo(() => geoNaturalEarth1(), []);
 const pathGenerator = useMemo(() => geoPath().projection(projection), [projection]);

 const getCoords = (coordinates: [number, number]) => {
  const projected = projection(coordinates);
  return {x: projected? projected[0]: 0, y: projected? projected[1]: 0};
};

 return (
  <svg ref={svgRef} width={2500} height={5000} style={{border: "1px solid black"}}>
   <g ref={gRef}>
    {/* Отображение карты */}
    {data.features.map((feature: any, i: number) => {
     const path = pathGenerator(feature);
     return path? (
      <path key={i} d={path} fill="#ccc" stroke="#333" strokeWidth={0.1} />
): null;
})}

    {/* Отображение рёбер (линий) */}
    {edges.map((edge, i) => {
     const sourcePoint = points.find((p) => p.id === edge.source);
     const targetPoint = points.find((p) => p.id === edge.target);

     if (sourcePoint && targetPoint) {
      const sourceCoords = getCoords(sourcePoint.coordinates);
      const targetCoords = getCoords(targetPoint.coordinates);

      return (
       <line
        key={i}
        x1={sourceCoords.x}
        y1={sourceCoords.y}
        x2={targetCoords.x}
        y2={targetCoords.y}
        stroke="#000"
```

```
            strokeWidth={0.1}
           />
);
}
       return null;
})}

      {/* Отображение точек */}
      {points.map((point) => {
       const {x, y} = getCoords(point.coordinates);
       return (
        <circle
         key={point.id}
         cx={x}
         cy={y}
         r={1}
         fill="red"
         stroke="#333"
         strokeWidth={0.5}
        >
         <title>{point.name}</title>
        </circle>
);
})}
    </g>
   </svg>
);
};
```

Рис. 5. Результат выполнения кода в Листинге 9:
графическое отображение карты с графом
Fig. 5. The result of executing Listing code 9:
graphical representation of the map with the graph

3.5 GeoJSON

GeoJSON — открытый формат для хранения и передачи географических данных, используемый для отображения карт с маршрутизированными графами. Данная библиотека позволяет задействовать в системе визуализацию карт, оформленных в формате JSON-файла без потери качества при масштабировании.

Листинг 10 — сервис для загрузки данных:

```
import {FeatureCollection} from "geojson";
export const fetchGeoData = async (): Promise<FeatureCollection> => {
  const response = await fetch('https://raw.githubusercontent.com/codeforgermany/click_
that_hood/main/public/data/russia.geojson');
 if (!response.ok) {
   throw new Error('Ошибка при загрузке GeoJSON');
}
 return await response.json();
};
```

Листинг 11 — вызов сервиса загрузки данных:

```
import {useState, useEffect} from "react";
import {FeatureCollection} from "geojson";
import {fetchGeoData} from "../services/geoService";

export const useMapData = (): FeatureCollection | null => {
 const [data, setData] = useState<FeatureCollection | null>(null);

 useEffect(() => {
  // Вызов сервиса для загрузки данных
  fetchGeoData()
.then((geojsonData) => {
     setData(geojsonData);
})
.catch((error) => {
     console.error("Ошибка при загрузке GeoJSON:", error);
});
}, []);

 return data;
};
```

Карта, загружаемая из кода, представленного в Листингах 10–11, изображена на рис. 5.

Разработанная система CNOSS демонстрирует значительные преимущества перед существующими решениями. Основными преимуществами являются:

• Масштабируемость. Система может масштабироваться как вертикально, так и горизонтально, благодаря микросервисной архитектуре.

• Адаптивность. В отличие от закрытых решений, таких как Elastic Services Platform, система CNOSS может быстро адаптироваться и обрастать новым функционалом под требования пользователя.

• Технологическая независимость. Благодаря микросервисной архитектуре система способна использовать разные технологии и языки программирования для разных сервисов. Например, для построения маршрутов используется язык Python с высоко развитыми библиотеками нейронных сетей. Для серверной части используется язык программирования Java и фреймворк Spring, который ориентирован на работу и обслуживания масштабной микросервисной архитектуры. Для клиентской части используется фреймворк Hilla, сочетающий в себе язык TypeScript с библиотекой React, а также язык программирования Java и фреймворк Spring.

Дальнейшие исследования могут быть направлены на новые способы поддержки сетей связи и улучшение старых, а также расширение функциональности системы с учетом новых требований операторов телекоммуникационных сетей [14].

ЗАКЛЮЧЕНИЕ

Предложено решение в виде системы операционной поддержки сети связи (CNOSS). Построенная система, базирующаяся на клиент-серверной архитектуре и микросервисном взаимодействии, позволяет эффективно управлять сетевыми ресурсами, минимизировать задержки и адаптироваться к динамичным изменениям в сети. Использование современных технологий как на стороне клиента, так и на стороне сервера, обеспечивает высокую производительность и масштабируемость решения, что делает его перспективным для дальнейшего развития в условиях стремительно растущих требований информационной инфраструктуры.

ВКЛАД АВТОРОВ

А.А. Замошников — сбор данных, концепция, анализ информации, подготовка текста.
И.В. Тузов — сбор данных, подготовка текста.
С.Я. Прокофьев — сбор данных, редактирование текста.
А.В. Синицын — концепция, сбор и анализ информации.
С.А. Селиванов — концепция, редактирование текста.

CONTRIBUTION OF THE AUTHORS

Alexander A. Zamoshnikov — data collection, concept, text preparation..
Ivan V. Tuzov — data collection, text preparation.
Sergey Ya. Prokofiev — data collection, text editing.
Alexander V. Sinitsyn — concept, information collection and analysis.
Sergey A. Selivanov — concept, text editing.

КОНФЛИКТ ИНТЕРЕСОВ

Авторы заявляют об отсутствии конфликта интересов.

CONFLICT OF INTEREST

The authors declare no conflict of interests.

СПИСОК ИСТОЧНИКОВ / REFERENCES

1. Сорокин Г.О., Синицын А.В. Алгоритм построения маршрутизации для обеспечения отказоустойчивости сети связи. *Информация и инновации*. 2024;19(2):84–95. https://doi.org/10.31432/1994–2443–2024–19–2–84–95. EDN: SQJTEE.
   Sorokin G.O., Sinitsyn A.V. Routing algorithm to ensure fault tolerance of the communication network. *Information and innovation*s. 2024;19(2):84–95. https://doi.org/10.31432/1994–2443–2024–19–2–84–95
2. Летов Н.К. Метод снижения конкуренции при параллельном доступе к данным в БД на основе распределения сообщений в Apache Kafka. *Вестник науки*. 2024;4(12):1598–1603. https://doi.org/10.24412/2712–8849–2024–1281–1598–1603. EDN: LLAIIA
   Letov N.K. A method for reducing competition in parallel access to data in a database based on message distribution in Apache Kafka. *Bulletin of Science*. 2024;4(12): 1598–1603. https://doi.org/10.24412/2712–8849–2024–1281–1598–1603
3. Платонова А.И. Сравнение производительности PostgreSQL и ее расширения TimescaleD B. *Современные инновации, системы и технологии*. 2024;4(3):0121–0133. https://doi.org/ 10.47813/2782–2818–2024–4–3–0121–0133. EDN: LSWLKJ
   Platonova A.I. Performance Comparison of PostgreSQL and its TimescaleDB extension. *Modern innovations, systems and technologies*. 2024;4(3):0121–0133. https://doi.org/10.47813/2782–2818–2024–4–3–0121–0133
4. Панасенко Н.Д., Дьяченко Н.В. Алгоритм маршрутизации трафика. *Молодой исследователь Дона*. 2021;№ 4(31):12–13. EDN SGWKQU.
   Panasenko N.D., Dyachenko N.V. Traffic routing algorithm. *Young researcher of the Don*. 2021; No 4(31):12–13. EDN SGWKQU.
5. Болдыревский П.Б., Зюзин В.Д. Формализация расчета вариантов маршрута в одноранговых сетях для задач информационной безопасности. *Экономика и качество систем связи*. 2025; № 35:66–76.
   Boldyrevsky P.B., Zyuzin V.D. Formalization of calculation of route options in peer-to-peer networks for information security tasks. *Economics and quality of communication systems*. 2025; No.35:66–76.
6. Дилмуродов З.Д. Протоколы маршрутизации и их применение в компьютерных сетях. *Экономика и социум*. 2025; № 1–2(128):1018–1022. EDN: UJUKYR.
   Dilmurodov Z.D. Routing protocols and their application in computer networks. *Economics and Society.* 2025; No.1–2(128):1018–1022. EDN: UJUKYR.
7. Новиков А.С., Пестин М.С. Распределённая маршрутизация трафика в беспроводных децентрализованных самоорганизующихся сетях связи. *Известия ТулГУ. Технические науки*. 2021; № 5:149–154. https://doi.org/10.24412/2071–6168–2021–5–149–154
   Novikov A.S., Pestin M.S. Distributed traffic routing in wireless decentralized self-organizing communication networks. *News of TulSU. Technical sciences*. 2021; No. 5: 149–154. https://doi.org/10.24412/2071–6168–2021–5–149–154
8. Чертков А.А., Каск Я.Н., Очина Л.Б. Маршрутизация потоковой сети на основе модификации алгоритма беллмана — форда. *Вестник государственного универси-*

тета морского и речного флота им. адмирала С.О. Макарова. 2022; 14(4):615–627. https://doi.org/10.21821/2309–5180–2022–14–4–615–627
Chertkov A.A., Kask Ya.N., Ochina L.B. Streaming network routing based on Bellman–Ford algorithm modification. *Vestnik Gosudarstvennogo universiteta morskogo i rechnogo flota imeni admirala S.O. Makarova.* 2022;14(4):615–627. https://doi.org/10.21821/2309–5180–2022–14–4–615–627

9. Листопад Н.И., Лавшук О.А. QoS маршрутизация в сетях телекоммуникаций. *Доклады БГУИР*. 2022;20(3):45–53. https://doi.org/10.35596/1729–7648–2022–20–3–45–53
Listopad N.I., Lavshuk O.A. QoS routing in telecommunications networks. *Reports of BSUIR*. 2022;20(3):45–53. (In Russ.) https://doi.org/10.35596/1729–7648–2022–20–3–45–53
10. Чиганов Д.Р. Sping: мощный фреймворк для разработки Java-приложений. *Вестник науки*. 2023;4(7):273–275. EDN: KOINJB.
Chiganov D.R. Spring: a powerful framework for developing Java applications. *Bulletin of Science*. 2023;4(7):273–275. EDN: KOINJB.
11. Ляшов Е.И. Проектирование высоконагруженных реактивных систем с использованием Java и Spring Webflux: преимущества перед традиционным стеком. *Международный журнал гуманитарных и естественных наук*. 2025. № 1–3(100):180–185. https://doi.org/10.24412/2500–1000–2025–1–3–180–185
Lyashov E.I. Designing highly loaded reactive systems using Java and Spring Webflux: advantages over the traditional stack. *International Journal of Humanities and Natural Sciences*. 2025; No.1–3(100):180–185. https://doi.org/10.24412/2500–1000–2025–1–3–180–185
12. Кравцов Е.П. Разработка высокопроизводительных React-приложений: методы и практики оптимизации. *European science*. 2024; № 1(69):53–58.
Kravtsov E.P. Development of high-performance React applications: optimization methods and practices. *European science*. 2024; No.1(69):53–58.
13. Волков А.Н. Задача маршрутизации в сети динамических туманных вычислений. *Труды учебных заведений связи*. 2024;10(4):27–37. https://doi.org/10.31854/1813–324X-2024–10–4–27–37. EDN: QWBVQY.
Volkov A.N. The routing problem in a dynamic fog computing network. *Proceedings of educational institutions of communication*. 2024;10(4):27–37. (In Russ.). https://doi.org/10.31854/1813–324X-2024–10–4–27–37
14. Утепбергенов А.О. Большие данные в управлении человеческими ресурсами. *Экономика и социум*. 2022; № 4–3(95):513–519. EDN: VGFUNU.
Utepbergenov A.O. Big data in human resource management. *Economics and Society*. 2022; No.4–3(95):513–519. EDN: VGFUNU.

ИНФОРМАЦИЯ ОБ АВТОРАХ

Александр Алексеевич Замошников, программист научной лаборатории кафедры Инструментального и прикладного программного обеспечения РТУ МИРЭА, Москва, Российская Федерация; ORCID: https://orcid.org/0009–0007–9745–6301; e-mail: zamoshnikov81@mail.ru

Александр Владимирович Синицын, кандидат физико-математических наук, РТУ МИРЭА Институт информационных технологий, Москва, Российская Федерация; ORCID: https://orcid.org/0000–0001–7392–1837; e-mail: a@sinitsyn.info

Сергей Александрович Селиванов, кандидат технических наук, доцент, заместитель директора по ИТ Международного центра по информатике и электронике (ИнтерЭВМ), Москва, Российская Федерация; ORCID: https://orcid.org/0000–0002–1229–9025; e-mail: selivanov@inevm.ru

Иван Владимирович Тузов, РТУ МИРЭА, Москва, Российская Федерация; ORCID: https://orcid.org/0009–0006–8554–3224; e-mail: tuzov.i.v@mail.ru

Сергей Яковлевич Прокофьев, РТУ МИРЭА, Москва, Российская Федерация; ORCID: https://orcid.org/0009-0002-3427-2963; e-mail: forester477@hotmail.com

INFORMATION ABOUT THE AUTHORS

Alexander A. Zamoshnikov, programmer at the Scientific Laboratory of the Department of Instrumental and Applied Software of RTU MIREA, Moscow, Russian Federation; ORCID: https://orcid.org/0009–0007–9745–6301; e-mail: zamoshnikov81@mail.ru

Alexander V. Sinitsyn, PhD of Physico-Mathematical Sciences, RTU MIREA Institute of Information Technology, Moscow, Russian Federation; ORCID: https://orcid.org/0000–0001–7392–1837; e-mail: a@sinitsyn.info

Sergey A. Selivanov, PhD of Technical Sciences, Associate Professor, Deputy Director of the Department of the International Center for Informatics and Electronics (InterEVM), Moscow, Russian Federation; ORCID: https://orcid.org/0000–0002–1229–9025; e-mail: selivanov@inevm.ru

Ivan V. Tuzov, RTU MIREA, Moscow, Russian Federation; ORCID: https://orcid.org/0009–0006–8554–3224; e-mail: tuzov.i.v@mail.ru

Sergey Ya. Prokofiev, RTU MIREA, Moscow, Russian Federation; ORCID: https://orcid.org/0009-0002-3427-2963; e-mail: forester477@hotmail.com

Поступила / Received 06.03.2025
Принята / Accepted 03.04.2025