УДК 004.4

ИСПОЛЬЗОВАНИЕ ПРОТОКОЛОВ REST API и WEBSOCKET ДЛЯ СТРУКТУРИЗАЦИИ ТРЕХЗВЕННОГО УРОВНЯ ЭМЕРДЖЕНТНЫХ СИСТЕМ И ОТОБРАЖЕНИЯ МЕДИАСИСТЕМ

М. М. Благирев[1] [0009-0008-2853-3411], А. О. Костыренков[2] [0009-0007-0294-694X]

[1, 2]МИРЭА – Российский Технологический Университет, проспект Вернадского, 78, г. Москва, 119454

[1]blagirevm@list.ru, [2]kostyrenkov@mirea.ru

Аннотация

Проведен анализ скорости и эффективности передачи данных с использованием протоколов WebSocket и REST API. Для сравнения скорости обработки потоковых объектов и выявления более надежной технологии для разработки API-интерфейсов использованы разложения базовых функций в ряды Тейлора и Фурье. В результате выявлено, что REST API является более быстрым и доступным ресурсом для передачи информационных данных в побитовом преобразовании, а масштабируемость этого протокола преобладает в количестве обрабатываемых единиц, что позволяет расширить количество проводимых тестов.

***Ключевые слова**:* масштабируемость, протоколирование, структуризация, REST API, WebSocket.

ВВЕДЕНИЕ

Современные веб-приложения нуждаются в эффективных методах передачи данных между клиентом и сервером. REST API и WebSocket являются двумя ключевыми подходами для решения этой задачи [1, 2].

REST API использует протокол HTTP и шаблон MVC, обеспечивая сбор, обработку и представление данных клиенту, подходит для операций CRUD (CREATE, READ, UPDATE, DELETE) и стандартизирует протоколы и форматы данных. Однако REST API требует для каждого запроса нового соединения с сервером, что может быть неэффективным при больших объемах данных.

WebSocket, напротив, устанавливает постоянное двустороннее соединение между клиентом и сервером, обеспечивая обмен данными в реальном времени без необходимости постоянных запросов, что снижает нагрузку на систему и ускоряет передачу данных. Это особенно важно для приложений с высокой нагрузкой, однако настройка WebSocket требует большего объема программного кода и может столкнуться с проблемами совместимости.

Сравнение двух названных подходов позволяет выявить наиболее эффективные решения для различных сценариев их использования в веб-приложениях.

1. REST API: РЕПРЕЗЕНТАТИВНОЕ СОСТОЯНИЕ МОДЕЛИ ПЕРЕДАЧИ И ОБРАБОТКИ ДАННЫХ

Составление и отработка протокола HTTP при обработке запроса и подхода REST позволяют обеспечить взаимосвязь между сервером и клиент-серверной частью, используя шаблон проектирования MVC при трехуровневой обработке запроса: сбор данных сервером (чтение и обучение математической модели); обработка информации (путем использования инструментов проектирования модели разработки и последующей реорганизации побитового состояния структуры в информационной системе); вывод на клиентскую часть пользователя предоставляемой обработанной информации [3].

Эмерджентные свойства обеспечиваются здесь путем взаимодействия дочерних классов и функций от основной иерархической инфографики, которые способны обеспечить синергетический эффект, связывая основную ступень проектирования и последующий этап отработки (компиляции/интерпретации) программного кода, пост- и препроцессинг для структуризации данных медиа единиц, например, передачи или получения обработанных графических объектов [4].

В настоящей работе рассмотрены использование и применение REST API методов GET, POST, PUT, DELETE для реализации основного принципа акронима CRUD баз данных (CREATE, READ, UPDATE, DELETE), а применение стека технологий позволит разработчикам модернизировать подход к обработке графических медиа с опорой на обширный спектр библиотек взаимодействия команд [5].

При оптимизации появляется возможность применить RESTful-сервисы на многофункциональных языках программирования, которые могут интерпретироваться на сторонних ресурсах или при облачном взаимодействии систем сервера с клиентской частью. Достоинство использования RESTful-сервисов обусловлено

критериями стандартизации протоколирования информационной библиотеки и статическими форматами данных [6].

В ходе проведенного исследования был использован один из немаловажных факторов ресурса REST API – масштабируемость. В контексте разработки информационной системы она способствует горизонтальному росту производительности, получая более значительные объемы запросов, а также большее количество клиентских частей пользователей, которые обращаются на сервер через протокол HTTP/HTTPS.

HTTP/HTTPS – соответственно защищенный и незащищенный протоколы передачи гипертекста между клиентом и сервером. Протокол HTTPS медленнее, чем HTTP, из-за потребления системных ресурсов для установки защищенного ресурса процессом SSL/TLS [7]. HTTP-протокол позволяет быстрее осуществлять передачу данных с сервера клиенту, чем HTTPS. RESTful требует установки нового соединения с сервисами или монолит-системами, что приводит к проблеме работоспособности в реальном времени: работа запрос–ответ не является быстропотоковой передачей данных при больших объемах количественно-качественных данных объекта. Таким образом, целесообразно рассмотреть архитектуру REST API (рис. 1) и сравнить ее с технологией WebSocket двустороннего соединения.

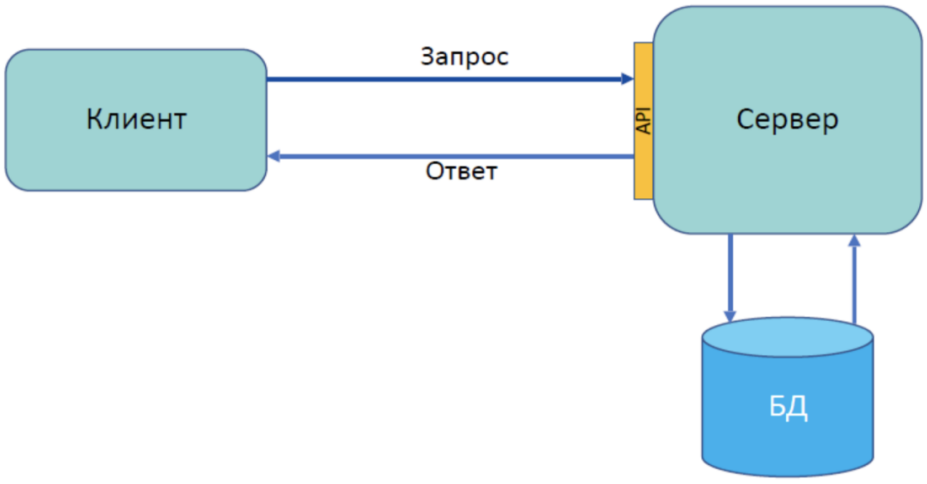


Рис. 1. Архитектура REST API.

2. WEBSOCKET – ДВУСТОРОННЕЕ СОЕДИНЕНИЕ ИНТЕРФЕЙСОВ

WebSocket является протоколом двусторонней связи между клиентом и сервером через единое постоянное соединение. В отличие от REST API, WebSocket позволяет обмениваться данными в реальном времени без необходимости постоянной отправки запросов на сервер для получения потокового ответа, что меньше нагружает систему, и быстрее напрямую отправляет объекты информационных данных на клиентскую часть.

Эффективность передачи данных поддерживается за счет непрерывного соединения между клиентом и сервером, тем самым сокращается объём передаваемых данных путем постоянной связи между клиентом и сервером, снижая нагрузку и на сеть, и на сервер. Масштабируемость для обработки большого количества одновременных подключений позволяет потоково обработать множество обращений пользователей для приложений с высокой нагрузкой [8].

Однако WebSocket, по сравнению с REST API, требует существенно большего количества описанных строк программного кода для подключения клиента к серверу по протоколу и значению ключа/токена. Многофункциональность технологии WebSocket ниже, чем у REST API, ввиду малой поддержки устаревших браузерных решений, так как сетевые прокси не всегда поддерживают соединение WebSocket, что создает пользователям весомые проблемы совместимости передачи пакетов данных.

Ключевым в WebSocket является распределение клиентских подключений в стендах, распределяющих отправку и переадресацию для обработки данных несколькими серверами для повышения стабильности, масштабируемости и надежности использования серверной составляющей [9].

Load Balancer как шаблон масштабируемости принимает входящие интернет-соединения и распределяет потоки подключения на имеющиеся сервера, равномерно распределяя нагрузку на сетевую установку. Серверы во время обработки сигнала, по которому отправляют запрос на клиентскую часть, соединяют пользователя с ответами сервера через балансировщик нагрузки и тем самым снижают нагрузку на сеть и требуют меньшее количество ресурсов ЭВМ [10].

На рис. 2 представлена архитектура WebSocket двустороннего соединения.

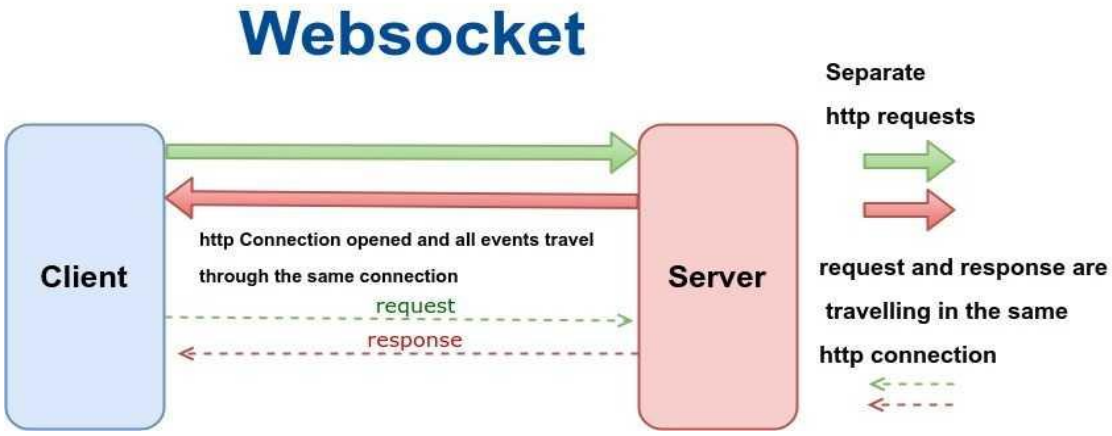


Рис. 2. Архитектура WebSocket двустороннего соединения.

Ниже на основе математического моделирования проведен анализ скорости обработки и передачи ответа протоколами двух технологий WebSocket и REST API для выявлений более быстрой и надежной технологии в разработке API интерфейсов.

3. АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ И ЭФФЕКТИВНОСТИ REST API И WEBSOCKET

Для анализа производительности и эффективности REST API и WebSocket были использованы разложения базовых функций в ряды Тейлора и Фурье. Рассмотрим, как каждый из этих методов может быть применен, и проиллюстрируем это примерами вычислений [11].

Как известно, ряд Тейлора используется для аппроксимации функций в некоторой точке *a* и для функции *f(x)* в окрестности этой точки имеет вид

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \ldots \qquad (1)$$

Выбор точки a для аппроксимации функций рядом Тейлора влияет на точность аппроксимации вблизи этой точки: чем ближе значение x к точке a, тем точнее аппроксимация. Обычно эту точку выбирают в зависимости от области, в которой требуется высокая точность аппроксимации.

Пусть x – размер запроса в килобайтах, а t – время обработки запроса в миллисекундах. Предположим, что у нас есть функции $f(x, t)$ и $g(x, t)$ времени отклика REST API и WebSocket соответственно, зависящие как от размера запроса, так и от времени его обработки. Будем считать, что эти функции представлены в виде суммы отрезков двух рядов Фурье $b + cx + dx^2 + et + kt^2$. Коэффициенты в

этих выражениях зависят от точной реализации и характеристик системы и настраиваются на основе измерений производительности и анализа работы системы в реальных условиях. Они могут быть получены путем сбора статистики времени отклика при различных входных данных и их анализа с помощью методов регрессии или других статистических методов [12]. В итоге имеем

$$f(x,t) = 0.5 + 0.005x + 0.00015x^2 + 0.2t + 0.05t^2,$$
$$g(x,t) = 0.85 + 0.002x + 0.00001x^2 + 0.3t + 0.04t^2.$$

Ниже приведены расчеты (таблицы 1 и 2), где коэффициенты перед переменными были взяты как средние значения за время проведения 24 тестов при использовании вычислительных ресурсов системы и запуска программного обеспечения.

Таблица 1. Расчет для REST API

Коэффициенты	*b*	*c*	*d*	*e*	*k*
Среднее значение	0.5	0.005	0.00015	0.2	0.05
Тест 1	0.5072	0.0035	0.00005	0.1664	0.0552
Тест 2	0.4946	0.0009	0.00014	0.2349	0.0534
Тест 3	0.5021	0.0021	0.00027	0.1957	0.0462
Тест 4	0.4956	0.0006	0.00012	0.2065	0.0533
Тест 5	0.4984	0.0034	0.00004	0.2113	0.0489
Тест 6	0.4999	0.0019	0.00024	0.1920	0.0487
Тест 7	0.5013	0.0040	0.00010	0.2068	0.0653
Тест 8	0.4985	0.0004	0.00011	0.1961	0.0563
Тест 9	0.5001	0.0037	0.00032	0.1987	0.0442
Тест 10	0.4953	0.0040	0.00023	0.2048	0.0426
Тест 11	0.4987	0.0018	0.00011	0.2193	0.0519
Тест 12	0.5002	0.0045	0.00017	0.1973	0.0564
Тест 13	0.4978	0.0010	0.00026	0.2204	0.0601
Тест 14	0.5006	0.0027	0.00020	0.1948	0.0655
Тест 15	0.4987	0.0010	0.00025	0.1991	0.0446
Тест 16	0.4982	0.0006	0.00007	0.1832	0.0556

Тест 17	0.4998	0.0043	0.00031	0.2218	0.0444
Тест 18	0.4992	0.0021	0.00011	0.2069	0.0633
Тест 19	0.5014	0.0036	0.00036	0.2252	0.0518
Тест 20	0.4983	0.0003	0.00012	0.1941	0.0518
Тест 21	0.4995	0.0046	0.00012	0.2287	0.0428
Тест 22	0.4976	0.0032	0.00006	0.2206	0.0361
Тест 23	0.4971	0.0034	0.00013	0.1954	0.0565
Тест 24	0.5034	0.0003	0.00015	0.2234	0.0578

Таблица 2. Расчет для WebSocket

Коэффициенты	*b*	*c*	*d*	*e*	*k*
Среднее значение	0.85	0.002	0.00001	0.3	0.04
Тест 1	0.92647443	0.00141946	0.0000084	0.307672	0.0552
Тест 2	0.93419272	0.00164982	0.0000195	0.295145	0.0534
Тест 3	0.9577212	0.00107814	0.0000077	0.301847	0.0463
Тест 4	0.77473834	0.00164851	0.0000217	0.298783	0.0533
Тест 5	0.86819741	0.00196397	0.0000158	0.298532	0.0364
Тест 6	0.84029113	0.00150823	0.0000071	0.298520	0.0487
Тест 7	0.81684921	0.00179711	0.0000192	0.293258	0.0653
Тест 8	0.94923999	0.00155853	0.0000186	0.301621	0.0536
Тест 9	0.83967457	0.00131092	0.0000216	0.293761	0.0442
Тест 10	0.92380555	0.00204906	0.0000094	0.296559	0.0426
Тест 11	0.81506841	0.00110384	0.0000212	0.298147	0.0519
Тест 12	0.87191482	0.00191879	0.0000119	0.294806	0.0564
Тест 13	0.92505202	0.00071384	0.0000124	0.303355	0.0601
Тест 14	0.84436529	0.00111471	0.0000163	0.29347	0.0655
Тест 15	0.87484921	0.00145167	0.0000116	0.301448	0.0446
Тест 16	0.86617473	0.00086182	0.0000062	0.295909	0.0556

Тест 17	0.83200251	0.00158115	0.0000213	0.295078	0.0444
Тест 18	0.93615828	0.00121738	0.0000105	0.299004	0.0633
Тест 19	0.94317695	0.00209512	0.0000094	0.304752	0.0518
Тест 20	0.80872571	0.00186102	0.0000113	0.293135	0.0518
Тест 21	0.79983723	0.00203884	0.0000141	0.294890	0.0428
Тест 22	0.87970266	0.00062239	0.0000126	0.295250	0.0361
Тест 23	0.83465939	0.00168432	0.0000067	0.301058	0.0564
Тест 24	0.89280647	0.00156213	0.0000119	0.296284	0.0538

Ниже приведено обобщение результатов, с опорой на ранее проведенные тесты (таблица 3), где представлены производные функций $f(x, t)$ и $g(x, t)$ времени отклика REST API и WebSocket.

Таблица 3. Обобщение результатов

x	*t*	*f(x,t)*	*f'(x,t)*	*f'' x,t)*	*g(x,t)*	*g'(x,t)*	*g''(x,t)*	*REST API*	*WebSocket*
1000	52	301.1	8.405	0.1003	135.46	4.482	0.80002	309.6	140.7
10000	143	1661.01	44.505	0.1003	1877.9	11.942	0.80002	1705.6	1889.6
50000	782	40598.3	228.41	0.1003	49781	63.682	0.80002	40726	49844.0

Таким образом, в ходе экспериментов по результатам расчетов на малых объемах данных эффективнее оказался WebSocket. На больших объемах лучше справляется с задачей REST API.

Ряд Фурье можно использовать для анализа периодических колебаний в данных, например, в передаче сообщений. Если время отклика или задержки передается периодически, можно разложить соответствующую функцию в ряд Фурье.

Представим, в качестве примера, что задержка *T(t)* передается как 2π-периодическая четная функция *T(t)*=cos *t*. Ниже приведён расчет коэффициентов для ряда Фурье [13], полученные результаты представлены в таблице 4.

Таблица 4. Расчет для ряда Фурье

REST API	
t	*T(t)*
52	0.16
143	0.057
782	0.967
234	0.047
112	0.455
7654	0.471
2134	0.653
WebSocket	
t	*T(t)*
43	0.555
164	0.804
642	0.440
154	0.998
164	0.804
5839	0.343
1782	0.754

Для исследования адаптационных алгоритмов на базе интерфейсов REST API и WebSocket была разработана система, включающая три уровня структуризации данных: сбор данных, их обработка и отображение. В качестве данных использовались текстовые потоки данных. Выполнение работы происходило в следующем порядке:

- Подготовка данных: исходные данные были преобразованы в удобный для анализа формат с использованием предобработки, включающей нормализацию и фильтрацию.
- Представление данных: преобразованные данные представляются в виде отрезков рядов Тейлора и Фурье для аппроксимации и анализа периодических составляющих.
- Интеграция с REST API и WebSocket: обработанные данные передавались через REST API для хранения и дальнейшего анализа, а также через WebSocket для реалтайм-обновлений и визуализации на клиентской стороне.
- Структуризация данных: обработанные данные структурировались в трехзвенной модели, включающей уровень начальных данных, промежуточный

уровень обработанных данных и финальный уровень визуализированных данных.

Сводка итогов применения рядов Тейлора и Фурье представлена в таблице 5.

Таблица 5. Сводка итогов применения ряда Тейлора и ряда Фурье

Метод	Преимущества	Недостатки	Применимость
Ряд Тейлора	Гладкие переходы между уровнями	Ограниченная точность при больших значениях	Аппроксимация функций
Ряд Фурье	Анализ периодических сигналов	Неэффективен для непериодических сигналов	Анализ временных рядов

Таким образом, если система в первую очередь нуждается в мгновенном обновлении данных, при этом в режиме реального времени и при эффективной передаче данных, лучше использовать WebSocket.

Если система нуждается в более простой реализации, то REST API станет отличным решением рассмотренной задачи.

ЗАКЛЮЧЕНИЕ

REST API требует нового соединения для каждого запроса, что может привести к снижению эффективности при больших объемах данных.

WebSocket обеспечивает постоянное двустороннее соединение между клиентом и сервером, что позволяет обмениваться данными в реальном времени без необходимости постоянных запросов. Это значительно снижает нагрузку на систему и ускоряет передачу данных, что особенно важно для приложений с высокой нагрузкой. Тем не менее, настройка WebSocket может потребовать большего количества программного кода и столкнуться с проблемами совместимости.

Сравнение двух подходов позволило выявить наиболее эффективные решения для различных сценариев использования в веб-приложениях. REST API подходит для структурированных, стандартизированных операций с данными, тогда как WebSocket более эффективен для приложений, требующих быстрой и постоянной передачи данных. Оптимальное применение настоящих технологий способствует оптимизации трёхуровневых звеньев по кластеризации передачи протоколов

клиент-серверной информационной системы и обработки файловых составляющих. Таким образом, использование WebSocket, в отличие от REST API, позволяет систематически быстрее преобразовать передачу данных в многопоточном режиме пользования клиентских запросов на сервер, тем самым было получено наиболее эффективное решение для отправки протоколированных пакетов в реальном времени.

СПИСОК ЛИТЕРАТУРЫ

1. REST API Tutorial // REST API. URL: https://restfulapi.net/rest/

2. API WebSocket (WebSockets) // MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

3. *Куляшова Н.М.* Программное обеспечение и алгоритмы распознавания адресных структур информационные технологии // Информационные технологии. 2021. Т. 27, № 5. С. 275–280.

4. *Oznamets V.V., Ndayishimiye D., Kovalenko N.I.* Spatial modeling when creating a coordinate base (on the example of the republic of burundi) // Slavic Forum. 2023. № 1 (39). С. 302–314.

5. *Chakraborty S., Aithal P.S.* CRUD operation on wordpress database using C# and REST API // International Journal of Applied Engineering and Management Letters. 2023. С. 130–138.

6. *Анисимов В.И., Васильев С.А., Евдокимов И.А., Тарасова О.Б.* Обзор методов автоматизации тестирования и документирования серверного интерфейса, основанного на архитектуре RESTFUL // Информационные технологии в проектировании и производстве. 2020. № 2 (178). С. 45–48.

7. *Мартыненков И.В.* Основные этапы развития криптографических протоколов SSL/TLS и IPSEC // Прикладная дискретная математика. 2021. № 51. С. 31–67.

8. *Малсугенов О.В., Чипига А.Ф., Львова А.П.* Повышение эффективности беспроводного оптического канала передачи данных в видимом диапазоне световых волн // В сборнике: Информационные технологии интеллектуальной поддержки принятия решений. Труды VII Всероссийской научной конференции (с приглашением зарубежных ученых). В 3-х томах. 2019. С. 65–70.

9. *Дерябин Е.А., Готская И.Б.* Анализ круговой задержки при использовании веб-протоколов HTTP И WEBSOCKET // Студенческий. 2018. № 10-2 (30). С. 32–34.

10. *Yang L., Yao H., Wang J. et al.* Multi-UAV-Enabled Load-Balance Mobile-Edge Computing for IoT Networks // IEEE Internet of Things Journal. 2020. V. 7, No. 8. P. 6898–6908.

11. *Кузьмичев Н.Д.* Применение рядов Тейлора–Фурье для численного и экспериментального определения производных изучаемой зависимости // Журнал Средневолжского математического общества. 2011. Т. 13, № 2. С. 70–80.

12. *Касаткина Т.И., Гречишников Е.В., Дидрих В.Е., Соловьев А.С.* Математические методы моделирования и алгоритмы систем анализа и обработки информации для исследований динамики сложных систем // Вестник Воронежского института ФСИН России. 2017. № 4. С. 48–58.

13. *Романова Л.Д., Шаркунова Т.А., Елисеева Т.В.* Интегральные преобразования: учеб. пособие. Пенза: Изд-во ПГУ, 2015. 80 с.

USE OF REST API AND WEBSOCKET INTERFACES ALGORITHMS FOR STRUCTURING THE THREE-LINK LEVEL OF EMERGENT SYSTEMS AND DISPLAYING MEDIA SYSTEMS

M. M. Blagirev[1] [0009-0008-2853-3411], A. O. Kostyrenkov[2] [0009-0007-0294-694X]

[1, 2]*MIREA – Russian Technological University, Vernadsky Avenue, 78, Moscow, 119454*

[1]*blagirevm@list.ru,* [2]*kostyrenkov@mirea.ru*

Abstract

An analysis of the speed and efficiency of data transfer using the WebSocket and REST API protocols was carried out. To compare the speed of processing stream objects and identify a more reliable technology for developing APIs, expansions of basic functions in Taylor and Fourier series were used. As a result, it was revealed that the REST API is a faster and more accessible resource for transmitting information data in a bitwise transformation, and the scalability of this protocol prevails in the number of processed units, which allows expanding the number of tests performed.

***Keywords**:* scalability, logging, structuring, REST API, WebSocket.

REFERENCES

1. REST API Tutorial // REST API. URL: https://restfulapi.net/rest/

2. API WebSocket (WebSockets) // MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

3. *Kulyashova N.M.* Software and algorithms for recognizing address structures, information technologies // Information Technology. 2021. T. 27, no. 5. P. 275–280.

4. *Oznamets V.V., Ndayishimiye D., Kovalenko N.I.* Spatial modeling when creating a coordinate base (on the example of the republic of burundi) // Slavic Forum. 2023. No. 1 (39). P. 302–314.

5. *Chakraborty S., Aithal P.S.* CRUD operation on wordpress database us-ing C# and REST API // International Journal of Applied Engineering and Management Letters. 2023. P. 130–138.

6. *Anisimov V.I., Vasiliev S.A., Evdokimov I.A., Tarasova O.B.* Review of methods for automating testing and documenting a server interface based on the RESTFUL architecture // Information technologies in design and production. 2020. No. 2 (178). P. 45–48.

7. *Martynenkov I.V.* Main stages of development of cryptographic protocols SSL/TLS and IPSEC // Applied discrete mathematics. 2021. No. 51. P. 31–67.

8. *Malsugenov O.V., Chipiga A.F., Lvova A.P. I*ncreasing the efficiency of a wireless optical data transmission channel in the visible range of light waves // In the collection: Information technologies for intelligent decision support. Proceedings of the VII All-Russian Scientific Conference (with the invitation of foreign scientists). In 3 volumes. 2019. P. 65–70.

9. *Deryabin E.A., Gotskaya I.B.* Analysis of round-trip delay when using web protocols HTTP and WEBSOCKET // Student. 2018. No. 10-2 (30). P. 32–34.

10. *Yang L., Yao H., Wang J, et al.* Multi-UAV-Enabled Load-Balance Mobile-Edge Computing for IoT Networks // IEEE Internet of Things Journal. 2020. V. 7, No. 8. P. 6898–6908.

11. *Kuzmichev N.D.* Application of Taylor-Fourier series for the numerical and experimental determination of derivatives of the studied dependence // Journal of the Middle Volga Mathematical Society. 2011. V. 13, No. 2. P. 70–80.

12. *Kasatkina T.I., Grechishnikov E.V., Diedrich V.E., Solovyov A.S.* Mathematical methods of modeling and algorithms for analysis and information processing systems for studying the dynamics of complex systems // Bulletin of the Voronezh Institute of the Federal Penitentiary Service of Russia. 2017. No. 4. P. 48–58.

13. *Romanova L.D., Sharkunova T.A., Eliseeva T.V.* Integral transformations: textbook. allowance. Penza: PSU Publishing House, 2015. 80 p.

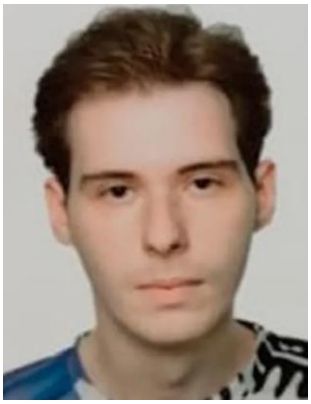
СВЕДЕНИЯ ОБ АВТОРАХ

БЛАГИРЕВ Михаил Михайлович – ассистент кафедры инструментального и прикладного программного обеспечения Института информационных технологий МИРЭА – Российского Технологического Университета.

Mikhail Mikhailovich BLAGIREV *– assistant of the department of instrumental and applied software of the Institute of Information Technologies MIREA – Russian Technological University.*

email: blagirevm@list.ru;

ORCID: 0009-0008-2853-3411



КОСТЫРЕНКОВ Алексей Олегович – ассистент кафедры инструментального и прикладного программного обеспечения Института информационных технологий МИРЭА – Российского Технологического Университета.

Alexey Olegovich KOSTYRENKOV *– assistant of the department of instrumental and applied software of the Institute of Information Technologies MIREA – Russian Technological University.*

email: kostyrenkov@mirea.ru;

ORCID: 0009-0007-0294-694X

Материал поступил в редакцию 20 июня 2024 года