РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ ОБРАБОТКИ КЛИЕНТСКИХ ЗАЯВОК С ПРИМЕНЕНИЕМ БИБЛИОТЕК MPI.NET И ENTITY FRAMEWORK CORE

Научная статья

Кремлева Э.Ш.[1,] *, Зиалтдинов Р.Э.[2]
[1] ORCID : 0000-0003-0858-0575;
[1] Казанский национальный исследовательский технический университет им. А.Н. Туполева – КАИ, Казань, Российская Федерация
[2] Футбольный клуб «Спартак-Москва», Москва, Российская Федерация

* Корреспондирующий автор (e-smile29.04[at]mail.ru)

Аннотация

В статье представлена разработка автоматизированной системы обработки клиентских заявок с использованием библиотек MPI.NET и Entity Framework Core. В работе исследованы ключевые аспекты интеграции этих технологий для обеспечения параллельной обработки данных, что позволило значительно повысить производительность и масштабируемость системы при работе с большими объемами информации. Процесс тестирования показал, что использование MPI.NET в сочетании с Entity Framework Core позволяет эффективно распределять нагрузку между процессами и ускорять выполнение операций. Однако было установлено, что не все типы операций демонстрируют строго линейное ускорение при увеличении числа процессов, что требует тщательного подхода к выбору и настройке числа параллельно выполняемых задач. Приведены практические рекомендации по использованию разработанной системы в реальных корпоративных условиях, что делает предложенное решение перспективным для применения в масштабных информационных системах.

Ключевые слова: автоматизация обработки заявок, параллельная обработка, MPI.NET, Entity Framework Core, масштабируемость, производительность, тестирование, корпоративные информационные системы.

DEVELOPMENT OF AN AUTOMATED SYSTEM FOR PROCESSING CLIENT REQUESTS USING MPI.NET AND ENTITY FRAMEWORK CORE LIBRARIES

Research article

Kremleva E.S.[1,] *, Zialtdinov R.E.[2]
[1] ORCID : 0000-0003-0858-0575;
[1] Kazan National Research Technical University named after A.N. Tupolev – KAI, Kazan, Russian Federation
[2] Spartak Moscow Football Club, Moscow, Russian Federation

* Corresponding author (e-smile29.04[at]mail.ru)

Abstract

The article presents the development of an automated system for processing client requests using MPI.NET and Entity Framework Core libraries. The article studies the key aspects of integration of these technologies to provide parallel data processing, which allowed to significantly increase the performance and scalability of the system when working with large amounts of information. The testing process has shown that using MPI.NET in combination with Entity Framework Core allows to effectively distribute the load between processes and accelerate the execution of operations. However, it was found that not all types of operations demonstrate strictly linear acceleration when the number of processes increases, which requires a careful approach to selecting and setting the number of tasks to be executed in parallel. Practical recommendations for using the developed system in real corporate conditions are given, which makes the proposed solution promising for application in large-scale information systems.

Keywords: application processing automation, parallel processing, MPI.NET, Entity Framework Core, scalability, performance, testing, enterprise information systems.

Введение

В условиях стремительно развивающегося цифрового мира, компании сталкиваются с возрастающими требованиями к эффективности, масштабируемости и быстродействию систем, предназначенных для обработки больших объемов клиентских заявок. По мере роста организаций и увеличения клиентской базы способность обрабатывать запросы клиентов быстро и точно становится критически важным фактором для поддержания удовлетворенности клиентов и оптимизации бизнес-процессов. Интеграция MPI.NET и Entity Framework Core в архитектуру автоматизированных систем обработки клиентских заявок представляет собой перспективный подход к решению этих задач за счет использования возможностей параллельной обработки [1].

Современные предприятия часто работают с огромными объемами данных и большим количеством клиентских взаимодействий, что требует от систем способности справляться с параллельной обработкой задач. Традиционные методы последовательной обработки могут приводить к возникновению узких мест, снижая общую производительность системы и замедляя время отклика на клиентские запросы. Такая неэффективность может напрямую повлиять на уровень удовлетворенности клиентов, привести к упущенным возможностям или ухудшению репутации бренда.

MPI.NET, представляющая собой управляемую библиотеку, позволяющую реализовывать параллельное программирование в рамках .NET, предлагает решение этих проблем, позволяя распределять вычислительные задачи

между несколькими процессорами или узлами. Это распределение критически важно для масштабирования систем без ущерба для их производительности. В сочетании с Entity Framework Core, популярным объектно-реляционным маппером (ORM), который упрощает взаимодействие с базами данных в .NET-приложениях, MPI.NET обеспечивает эффективное выполнение параллельных операций с базами данных, таких как запросы, обновления и управление большими объемами данных.

MPI.NET обладает рядом уникальных преимуществ для параллельной обработки клиентских заявок. Во-первых, использование этой технологии позволяет значительно увеличить пропускную способность системы за счет распределения задач между несколькими процессорами, что дает возможность одновременно обрабатывать большее количество запросов. Это особенно важно для тех систем, которым необходимо масштабироваться в ответ на растущий клиентский спрос. Во-вторых, параллельная обработка способствует минимизации задержек, связанных с обработкой отдельных запросов. Это крайне важно для приложений, работающих в реальном времени, где задержки могут негативно сказаться на опыте пользователей. В-третьих, MPI.NET оптимизирует использование системных ресурсов путем балансировки нагрузки между доступными процессорами, что гарантирует максимально эффективную работу системы и снижает вероятность как недоиспользования ресурсов, так и перегрузки отдельных компонентов.

Entity Framework Core [2], [3] широко известен благодаря своей способности упрощать доступ к данным в .NET-приложениях, абстрагируя сложные операции с базами данных в более управляемую и удобную для разработчиков форму. В сочетании с MPI.NET он улучшает способность системы эффективно управлять и обрабатывать большие объёмы данных.

Интеграция позволяет выполнять параллельные запросы к базе данных, что особенно полезно в ситуациях, когда клиентские запросы требуют сложных операций по извлечению данных или значительных вычислительных ресурсов. Например, обработка нескольких заказов клиентов, выполнение массовых обновлений или проведение аналитики на основе клиентских данных могут быть оптимизированы за счёт использования параллелизма MPI.NET в сочетании с мощными возможностями управления данными Entity Framework Core.

Актуальность применения MPI.NET и Entity Framework Core в разработке автоматизированных систем обработки клиентских заявок заключается в их способности повышать масштабируемость, улучшать производительность и обеспечивать эффективное использование ресурсов. В условиях, когда предприятия сталкиваются с вызовами, связанными с управлением большими объёмами клиентских взаимодействий, данные технологии предоставляют мощную основу для создания систем, способных удовлетворять требованиям современного корпоративного окружения. Используя параллельную обработку с MPI.NET вместе с возможностями управления данными, предлагаемыми Entity Framework Core, разработчики могут создавать решения, которые не только соответствуют текущим потребностям, но и готовы к дальнейшему росту и развитию технологий.

Такой подход представляет собой значительный шаг вперёд в проектировании высокопроизводительных, масштабируемых систем, необходимых для поддержания конкурентоспособности в современной цифровой экономике.

В качестве прикладной сферы применения разрабатываемой системы выбрана задача по автоматизации приема и согласования заявок стюардов на массовые мероприятия в компании АО «ФК «Спартак-Москва». Решение данной задачи сопряжено с использованием СУБД MS SQL Server [4], [5], [6] и платформы разработки веб-приложений ASP.NET Core на языке программирования C# [7], [8].

Актуальность выбранной прикладной сферы обусловлена необходимостью создания новой информационной системы для комплекса задач по работе со стюардами во время спортивных мероприятий, так как в данный момент данная задача выполняется исключительно в ручном режиме, тем самым нерационально используется человеческий труд. Так как данная задача является нетипичной и индивидуальной, то возможности приобрести такую систему не является возможной, поэтому оптимальным решением в данной ситуации будет самостоятельная разработка.

Целью работы является повышение обработки клиентских заявок с применением библиотек MPI.NET и Entity Framework Core. Для достижения данной цели ставятся и решаются следующие задачи:

- разработать основные требования к системе;
- разработать структуру тестовой базы данных;
- сгенерировать тестовую базу данных и провести тестирование разработанной системы при различных объемах базы данных;
- сформировать практические рекомендации по использованию выбранных программных компонентов системы.

Функциональные требования к системе

В экономическом плане крупные компании стали замечать, что с развитием информационных технологий стало возможным автоматизировать ряд рутинных задач, на которые необходимо тратить все возможные ресурсы (трудовые, временные и экономические). При этом от скорости выполнения, например аналитических задач и задач построения отчётности, зависит производственный уровень как департамента, так и организации в целом.

Экономическая сущность задачи автоматизации приема и согласования заявок стюардов заключается в возможности экономии времени сотрудников на выполнение автоматизируемой задачи.

Рассмотрим процесс приема и согласования заявок стюардов на массовые мероприятия в компании «АО «ФК Спартак-Москва» с помощью методологии IDEF0 [9].

На рисунке 1 представлена функциональная модель, отражающая результаты автоматизации бизнес-процесса.

Входной информацией являются личные данные стюарда, информация из заявки, информация по актуальным мероприятиям, информация о посещении.

Участниками процесса являются:

- Сотрудник отдела.
- Старший стюард.

Выходными данными являются:

- Оповещение о статусе заявки.
- Информация о неявках.

- Договор об оплате труда.

Декомпозиция функциональной модели представлена на рисунке 2. На рисунке 3 представлена декомпозиция процесса «Обработка заявки».

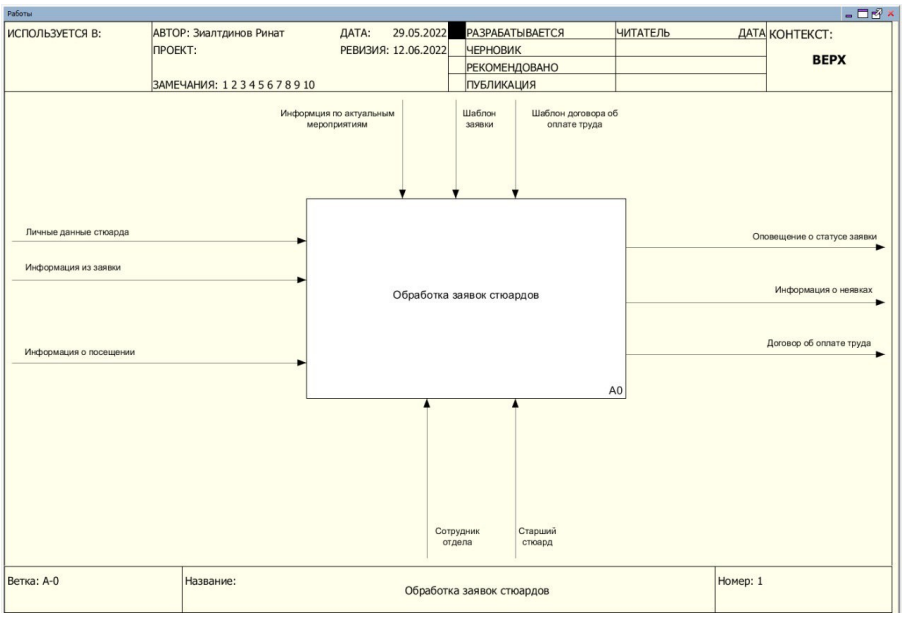

Рисунок 1 - Функциональная модель процесса «Обработка заявок стюардов»

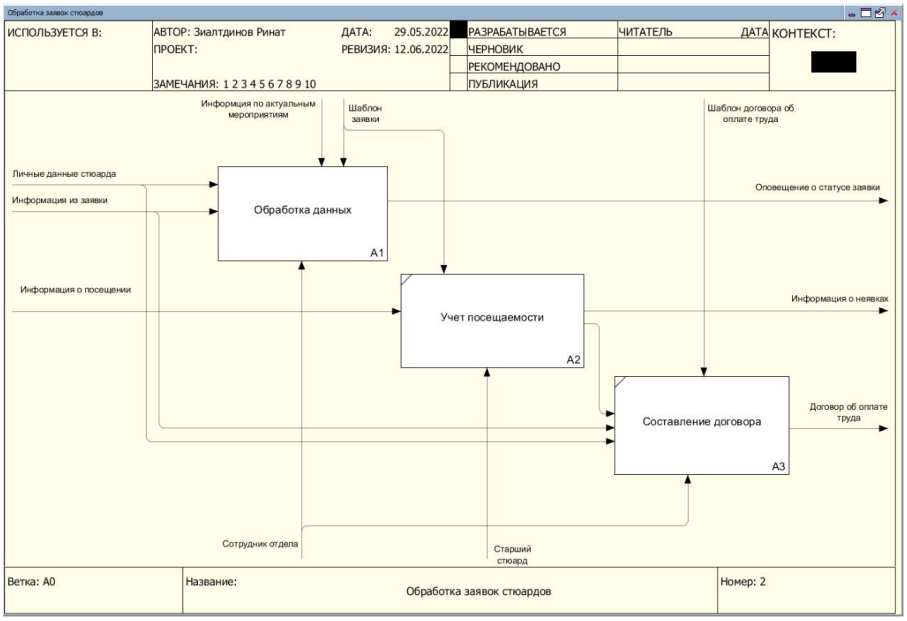

Рисунок 2 - Декомпозиция контекстной диаграммы

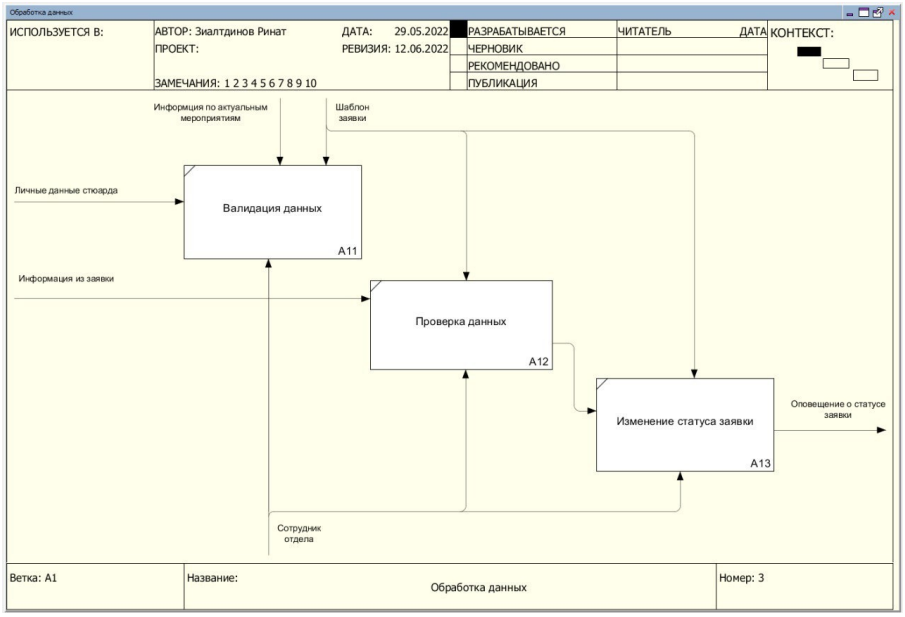

Рисунок 3 - Диаграмма декомпозиции процесса «Обработка данных»

Необходимо предусмотреть возможность работы с информационной системой администраторами, старшими стюардами, стюардами.

Представим функции выделенных ролей пользователей: администратор, старший стюард, стюард.

Администратор может решать следующие задачи:

1. Добавлять пользователям роли «Стюард» и «Старший стюард».
2. Смотреть данные всех стюардов и старших стюардов.
3. Создавать договоры.
4. Добавлять и изменять справочную информацию.
5. Смотреть все существующие заявки.
6. Создавать и изменять актуальные мероприятия.
7. Изменять заявки.
8. Привязывать стюардов к старшим стюардам.
9. Просматривать журнал аудита.
10. Удалять роли у пользователей.
11. Изменять личные данные.

Старший стюард с помощью информационной системы может:

1. Авторизоваться.
2. Изменить личные данные.
3. Создать заявку.
4. Просмотреть заявки привязанных стюардов.
5. Изменять заявки.
6. Редактировать заявки привязанных стюардов.
7. Просмотреть актуальные мероприятия.

Стюард сможет благодаря информационной системе может:

1. Зарегистрироваться.
2. Выбрать роль «Стюард».
3. Авторизоваться.
4. Просмотреть заявки привязанных стюардов.
5. Создать заявку.
6. Изменить заявку.
7. Изменить личные данные.

В соответствии с функциональными задачами информационной системы автоматизации приема и согласования заявок стюардов и требованиями, предъявляемыми к ее проектированию, при разработке системы был использован принцип предметно-ориентированного проектирования, обеспечивающий хорошую масштабируемость и наращивание функциональных возможностей продукта.

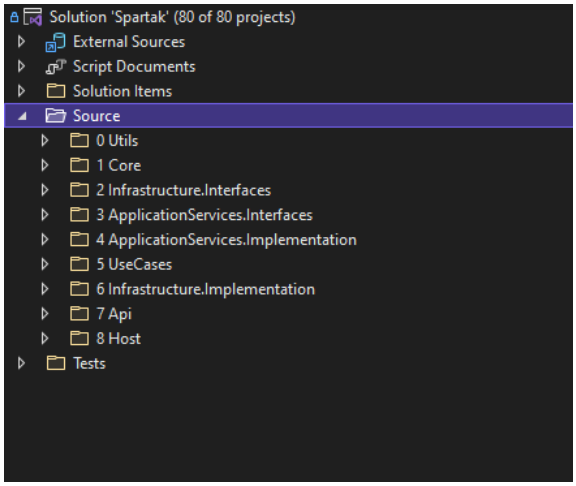

Рисунок 4 - Структурная схема информационной системы

Информационная система разбита на слои в соответствии с «Чистой архитектурой».

В каждом слое есть несколько компонентов – логических элементов решения (иначе говоря, проектов в решении).

Компоненты информационной системы:

- Utils содержит вспомогательные публичные классы (в т.ч. абстрактные), контракты, константы, enum'ы и т.д.: используемые другими проектами общие методы расширения, пагинация, валидация.

- Core содержит бизнес логику, данные, правила их обработки. Применяется анемичная модель и доменные сервисы.

- Infrastructure.Interfaces содержит интерфейсы инфраструктуры (в т.ч. интерфейсы для доступа к данным и интерфейсы сервисов инфраструктуры).

- ApplicationServices.Interfaces содержит интерфейсы уровня приложения.

- ApplicationServices.Implementation содержит реализацию сервисов приложения.

- UseCases содержит логику приложения, разделённую для различных платформ.

- Infrastructure.Implementation содержит реализацию интерфейсов инфраструктуры и доступа к данным.

- Api – корень композиции с контроллерами (а также swagger, обработка исключений, применения настроек и пр.).

- Host – инфраструктура хоста для различных платформ.

Для автоматизации приема и согласование заявок стюардов на массовые мероприятия в компании «АО «Спартак-Москва» необходимо выполнить следующие действия:

- Администратор должен авторизоваться в системе и заполнить справочную информацию, если это необходимо, внести данные о сотрудниках, если они не авторизированы, назначить им роли.

- Далее сотрудники должны авторизоваться в системе под своим логином и паролем и проверить свои разрешения. Если есть страницы, которые необходимы сотруднику, но они недоступны, обратиться к администратору.

- Для создания заявки стюарду необходимо зарегистрироваться в системе, указать, что хочет работать на массовых мероприятиях. Далее после авторизации пользователю будут доступны страница с заявками, в которой он может подать заявку, а на странице с мероприятиями посмотреть ближайшие матчи. На этих же страницах он может смотреть ход рассмотрения его заявки.

- После появления заявок, сотрудники могут изменять их, подтверждать или отклонять. Также они могут привязывать стюардов к нужным старшим стюардам. После окончания мероприятия есть возможность создать договор для оплаты труда.

- Старшие стюарды могут также подавать заявку, смотреть данные стюардов, которые привязаны к ним.

Описание тестовой базы данных

На рисунке 5 представлена инфологическая модель базы данных [10] информационной системы по работе с заявками стюардов на массовые мероприятия. Модель включает сущности и взаимосвязь между сущностями. Также выделены первичные и внешние ключи.

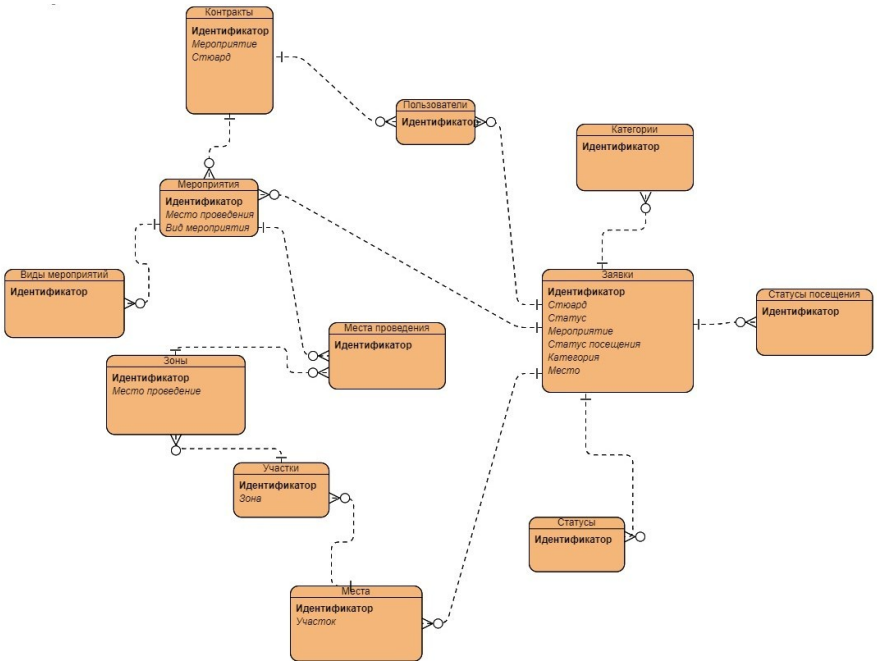

Рисунок 5 - Инфологическая модель базы данных

Для таблицы видов мероприятий заданы такие атрибуты, как идентификатор вида мероприятия, наименование, описание, признак «только для чтения», статус активности, версия, идентификатор пользователя, создавшего запись, дата создания, дата последнего изменения, идентификатор пользователя, внесшего последние изменения, и дата удаления пользователя. В качестве ключевых полей используются идентификатор вида мероприятия (первичный) и идентификатор пользователя (внешний).

В таблице мероприятий определены атрибуты, включающие идентификатор вида мероприятия, наименование, описание, статус активности, признак «только для чтения», версию, идентификатор пользователя, кем создана запись, дата создания, дата последнего изменения, идентификатор пользователя, кто сделал последнее изменение, дата удаления пользователя, идентификатор пользователя, кто удалил запись, мягкое удаление, учет посещаемости, коэффициент для расчёта рейтинга, даты начала и окончания мероприятия, идентификатор места проведения и вида мероприятия.

Атрибуты таблицы категорий включают идентификатор категории, наименование, описание, статус активности, признак «только для чтения», версию, идентификатор пользователя, кем создана запись, дата создания, дата последнего изменения, идентификатор пользователя, кто сделал последнее изменение, дата удаления пользователя, идентификатор пользователя, кто удалил запись, мягкое удаление, ставка оплаты труда и коэффициент для расчёта рейтинга.

В таблице договоров предусмотрены атрибуты, такие как идентификатор договора, наименование, описание, статус активности, признак «только для чтения», версия, идентификатор пользователя, кем создана запись, дата создания, дата последнего изменения, идентификатор пользователя, кто сделал последнее изменение, дата удаления пользователя, идентификатор пользователя, кто удалил запись, мягкое удаление, идентификатор мероприятия и стюарда, а также поле оплаты.

Таблица участков включает в себя такие атрибуты, как идентификатор участка, наименование, описание, статус активности, признак «только для чтения», версия, идентификатор пользователя, кем создана запись, дата создания, дата последнего изменения, идентификатор пользователя, кто сделал последнее изменение, дата удаления пользователя, идентификатор пользователя, кто удалил запись, мягкое удаление, внешний идентификатор и идентификатор зоны.

В таблице мест проведения представлены следующие атрибуты: идентификатор места проведения, наименование, описание, статус активности, признак «только для чтения», версия, идентификатор пользователя, кем создана запись, дата создания, дата последнего изменения, идентификатор пользователя, кто сделал последнее изменение, дата удаления пользователя, идентификатор пользователя, кто удалил запись, мягкое удаление и адрес.

Для таблицы мест определены атрибуты, включающие идентификатор места, наименование, описание, статус активности, признак «только для чтения», версию, идентификатор пользователя, кем создана запись, дата создания, дата последнего изменения, идентификатор пользователя, кто сделал последнее изменение, дата удаления пользователя, идентификатор пользователя, кто удалил запись, мягкое удаление, внешний идентификатор и идентификатор участка.

Атрибуты таблицы заявок включают идентификатор заявки, наименование, описание, статус активности, признак «только для чтения», версию, идентификатор пользователя, кем создана запись, дата создания, дата последнего изменения, идентификатор пользователя, кто сделал последнее изменение, дата удаления пользователя, идентификатор пользователя, кто удалил запись, мягкое удаление, идентификатор стюарда, идентификатор статуса, идентификатор мероприятия, идентификатор статуса посещения, идентификатор категории, начисленные баллы за мероприятие, оплаты за мероприятие, QR-код [11], [12], идентификатор места и фото стюарда.

В таблице статусов представлены такие атрибуты, как идентификатор заявки, наименование, описание, статус активности, признак «только для чтения», версия, идентификатор пользователя, кем создана запись, дата создания, дата последнего изменения, идентификатор пользователя, кто сделал последнее изменение, дата удаления пользователя, идентификатор пользователя, кто удалил запись, мягкое удаление, внешний идентификатор и цвет.

Таблица статусов посещения содержит атрибуты идентификатор заявки, наименование, описание, статус активности, признак «только для чтения», версию, идентификатор пользователя, кем создана запись, дата создания, дата последнего изменения, идентификатор пользователя, кто сделал последнее изменение, дата удаления пользователя, идентификатор пользователя, кто удалил запись, мягкое удаление, внешний идентификатор и цвет.

В таблице зон предусмотрены такие атрибуты, как идентификатор заявки, наименование, описание, статус активности, признак «только для чтения», версия, идентификатор пользователя, кем создана запись, дата создания, дата последнего изменения, идентификатор пользователя, кто сделал последнее изменение, дата удаления пользователя, идентификатор пользователя, кто удалил запись, мягкое удаление, внешний идентификатор и идентификатор места проведения.

Наконец, в таблице пользователей указаны атрибуты идентификатор пользователя, дата создания, идентификатор пользователя, кем создана запись, дата последнего изменения, идентификатор пользователя, кто сделал последнее изменение, дата удаления пользователя, идентификатор пользователя, кто удалил запись, мягкое удаление, имя, фамилия, отчество, URL фото, статус активности, внешний идентификатор, электронная почта, статус подтверждения электронной почты, хэш пароля, номер телефона и статус подтверждения номера телефона.

Взаимодействие MPI-процессов

В контексте разработанной системы обработки клиентских заявок, взаимодействие MPI-процессов [13] играет ключевую роль в обеспечении параллельной обработки и повышения производительности системы. В основе системы лежит архитектура, построенная с использованием MPI.NET для параллельного программирования и Entity Framework Core для взаимодействия с базой данных.

Система обработки клиентских заявок построена на принципе распределенной обработки задач, где мастер-процесс (Master Process) управляет и координирует выполнение задач между несколькими подчиненными процессами (Worker Processes). Такой подход позволяет эффективно масштабировать систему и обеспечивать высокую производительность при обработке большого объема клиентских запросов.

Когда клиент инициирует запрос, мастер-процесс принимает его и разделяет на несколько подзадач, которые могут быть параллельно выполнены. Например, в случае обработки заявок, запрос клиента может включать поиск, обновление или удаление записей в базе данных, содержащей такие таблицы, как «Журнал аудита», «Мероприятия», «Заявки» и другие. Каждая из этих операций требует выполнения одного или нескольких запросов к базе данных, которые могут быть параллелизированы с использованием MPI.NET.

Мастер-процесс распределяет подзадачи между подчиненными процессами. Каждый подчиненный процесс, получив задачу, взаимодействует с базой данных через Entity Framework Core, выполняя необходимые операции, такие как чтение, обновление или вставка данных. Entity Framework Core абстрагирует работу с базой данных, что позволяет подчиненным процессам выполнять запросы на уровне объектов, обеспечивая безопасность и удобство работы с данными.

После выполнения подзадач, подчиненные процессы отправляют результаты обратно в мастер-процесс. Мастер-процесс собирает все полученные результаты, агрегирует их и формирует окончательный ответ, который затем отправляется клиенту. В случае возникновения ошибок в процессе обработки подчиненные процессы сообщают о них мастер-процессу. Мастер-процесс может решить повторить задачу, обработать ошибку или передать сообщение об ошибке клиенту, что обеспечивает надежность системы.

Таблицы «Заявки», «Мероприятия» и «Категории» содержат большие объемы данных, которые часто подвергаются изменениям и запросам. Благодаря использованию параллельных процессов, операции с этими таблицами могут выполняться одновременно несколькими подчиненными процессами, что значительно сокращает время обработки запросов.

Entity Framework Core позволяет подчиненным процессам эффективно управлять сложными операциями с данными, такими как фильтрация, агрегация и объединение данных из нескольких таблиц. Использование внешних ключей в таких таблицах, как «Журнал аудита» и «Заявки», позволяет поддерживать целостность данных и упрощает выполнение связных операций между таблицами.

В контексте этой архитектуры, параллельная обработка с использованием MPI.NET и Entity Framework Core обеспечивает высокий уровень масштабируемости и производительности системы, что особенно важно в условиях постоянно растущих объемов данных и увеличивающегося числа клиентских запросов. Такая система способна адаптироваться к изменениям нагрузки и эффективно обрабатывать большие объемы данных, что делает ее надежным решением для современных информационных систем.

На рисунке 6 представлен в упрощенной виде механизм взаимодействия MPI-процессов в разработанной системе.

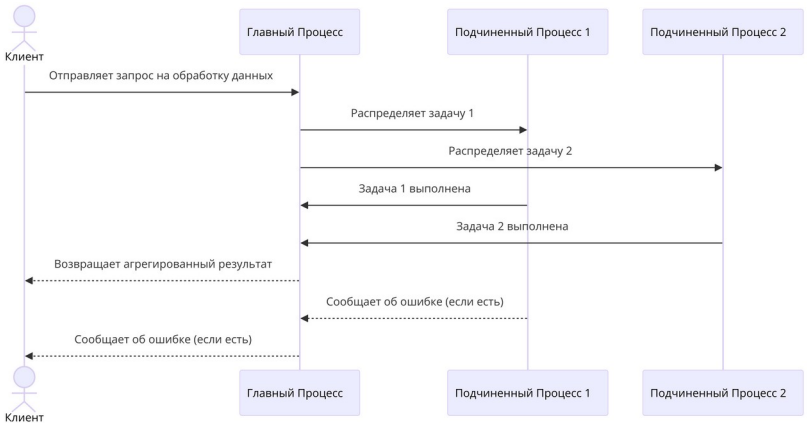

Рисунок 6 - Диаграмма взаимодействия MPI-процессов

Тестирование

Для оценки масштабируемости и производительности системы обработки клиентских заявок, разработанной с использованием MPI.NET и Entity Framework Core, было проведено тестирование на базе данных разного объема. Цель тестирования заключалась в определении влияния объема данных на эффективность параллельной обработки и времени выполнения основных операций системы.

Тестирование проводилось на трех различных объемах базы данных: 10000, 50000 и 100000 записей. Для каждого объема базы данных оценивалось время выполнения следующих операций:

- Генерация данных (команда «create»).
- Обновление данных на сервере (команда «save»).
- Изменение статуса всех пользователей на «offline» (команда «soff»).
- Подсчет количества заявок, поданных первыми 100 пользователями (команда «sum»).

Каждая операция выполнялась в двух сценариях: с одним процессом и с четырьмя процессами, что позволило оценить преимущества параллельной обработки при увеличении объема данных. Тестирование проводилось на платформе с характеристиками: процессор Intel Core i7 с частотой 3,85 ГГц [14], оперативная память DDR4 объемом 32 ГБ [15], операционная система Windows 11 Pro.

Для базы данных объемом 10000 записей генерация данных при одном процессе заняла 10 секунд, а при четырех процессах время сократилось до 2,8 секунд, что демонстрирует почти линейное ускорение. Однако для базы данных объемом 50000 записей время генерации составило 50 секунд при одном процессе и 14,2 секунды при четырех процессах. При работе с базой данных объемом 100000 записей время выполнения составило 100 секунд для одного процесса и 30 секунд для четырех процессов. Несмотря на улучшение производительности, наблюдается, что при увеличении объема данных накладные расходы на координацию процессов начинают замедлять прирост производительности, делая ускорение менее линейным.

Время обновления данных на сервере для базы данных объемом 10000 записей составило 8 секунд при одном процессе и 2,5 секунды при четырех процессах, показывая линейное ускорение. Однако для базы данных объемом 50000 записей время выполнения составило 40 секунд и 12 секунд соответственно, что также указывает на почти линейное ускорение. Но для базы данных объемом 100000 записей время выполнения составило 80 секунд для одного процесса и 25 секунд для четырех процессов. Здесь видно, что увеличение количества процессов дает значительное ускорение, но не достигает строго линейного улучшения из-за усложнения процесса синхронизации данных между процессами.

Для базы данных объемом 10000 записей команда «soff» завершилась за 1 секунду при одном процессе и за 0,4 секунды при четырех процессах. При 50000 записей время составило 5 секунд и 1,7 секунды соответственно, а при 100000 записей – 10 секунд для одного процесса и 3,5 секунды для четырех процессов. В этом случае прирост производительности ограничивается накладными расходами на координацию, и даже наблюдается меньшее улучшение по сравнению с другими операциями из-за того, что сама операция менее сложна и меньше выигрывает от параллелизма.

Подсчет количества заявок для базы данных объемом 10000 записей занял 4 секунды при одном процессе и 1,5 секунды при четырех процессах. Для базы данных объемом 50000 записей время составило 20 секунд и 6 секунд соответственно. При 100000 записей время выполнения составило 40 секунд для одного процесса и 14 секунд для четырех процессов. В этом случае наблюдается значительное ускорение, но оно также ограничено накладными расходами на обработку больших объемов данных, что приводит к небольшому отклонению от линейного роста.

Тестирование показало, что использование MPI.NET с Entity Framework Core в целом обеспечивает значительное ускорение при параллельной обработке данных. Однако ускорение не всегда является строго линейным, особенно при увеличении объема данных. В случаях генерации данных и обновления базы данных ускорение было близким к линейному, но усложнение координации процессов привело к тому, что прирост производительности с увеличением числа процессов начал снижаться.

Для менее сложных операций, таких как изменение статуса пользователей, прирост производительности был ограничен, так как данные операции менее выгодны для параллельного выполнения. Здесь накладные расходы на координацию процессов оказывали большее влияние, чем сами операции, что приводило к меньшему ускорению.

Таким образом, результаты тестирования показывают, что эффективность параллельной обработки зависит не только от объема данных, но и от сложности и характера выполняемой операции. Это подчеркивает важность

тщательного выбора количества процессов и понимания специфики задач при разработке параллельных систем на базе MPI.NET и Entity Framework Core.

Заключение

В ходе исследования была разработана автоматизированная система обработки клиентских заявок с применением библиотек MPI.NET и Entity Framework Core. Основным результатом работы стало значительное повышение производительности системы при параллельной обработке запросов благодаря интеграции MPI.NET, которая позволила эффективно распределять задачи между процессами. Это дало возможность обрабатывать большие объемы данных в кратчайшие сроки, что особенно актуально для корпоративных информационных систем с высокой нагрузкой. Научная новизна работы заключается в применении библиотеки MPI.NET в сочетании с Entity Framework Core для решения задачи обработки клиентских заявок в корпоративных системах. Данный подход не только увеличил масштабируемость системы, но и оптимизировал распределение ресурсов, что позволило снизить время отклика системы при увеличении числа параллельно выполняемых операций. Впервые проведено системное тестирование, показавшее, что для сложных операций, таких как генерация и обновление данных, система демонстрирует почти линейное ускорение при увеличении числа процессов. Однако для менее ресурсоемких задач, таких как изменение статуса пользователей, накладные расходы на координацию процессов ограничивают ускорение.

Предложенное решение подтверждает результаты исследований в области параллельной обработки данных с использованием MPI.NET, как было показано в работах Сабирова, Гибадуллина (2024) [1] и Лавиной, Матвеева (2023) [2]. Их исследования также подтверждают высокую эффективность параллельной обработки SQL-запросов и объектно-ориентированного доступа к данным в условиях увеличивающейся нагрузки на системы. Однако отличительной особенностью данного исследования является практическая реализация в реальной корпоративной системе и применение комбинированного подхода с Entity Framework Core, что делает предложенную систему более универсальной и эффективной в широком диапазоне сценариев использования.

Проведенное тестирование показало, что использование MPI.NET в сочетании с Entity Framework Core позволяет эффективно распределять нагрузку между процессами и обеспечивать ускорение выполнения операций. Однако, как показали результаты, не все виды операций демонстрируют строго линейное ускорение при увеличении числа процессов. Для более сложных операций, таких как генерация данных и обновление базы данных, прирост производительности близок к линейному, хотя и усложняется за счет необходимости синхронизации данных между процессами. В то же время, для операций, требующих меньших вычислительных ресурсов, таких как изменение статуса пользователей, прирост производительности был ограничен накладными расходами на координацию.

Таким образом, результаты исследования подчеркивают важность выбора подходящего количества процессов и их распределения в зависимости от типа выполняемых задач. Использование MPI.NET и Entity Framework Core в разработке системы обработки клиентских заявок подтвердило свою эффективность, обеспечив не только высокую производительность и масштабируемость, но и адаптивность к изменениям нагрузки, что делает предложенное решение перспективным для использования в реальных корпоративных информационных системах.

Для эффективного использования системы обработки клиентских заявок, разработанной с применением библиотек MPI.NET и Entity Framework Core, важно учитывать несколько ключевых аспектов. Во-первых, необходимо тщательно подбирать количество MPI-процессов в зависимости от объема данных и сложности выполняемых операций. Для задач, требующих значительных вычислительных ресурсов, таких как генерация данных или сложные обновления базы данных, увеличение числа процессов может значительно повысить производительность системы. Однако для менее сложных операций, таких как изменение статуса пользователей, избыточное увеличение числа процессов может привести к дополнительным накладным расходам на координацию и не обеспечить значительного прироста производительности.

Важным аспектом является мониторинг и управление нагрузкой на систему. Регулярное отслеживание распределения ресурсов между процессами позволяет поддерживать оптимальную производительность. В случае увеличения объема данных или количества запросов, система должна быть настроена на автоматическое увеличение числа задействованных процессов.

Использование Entity Framework Core должно быть направлено на упрощение работы с базой данных и обеспечение безопасного и эффективного взаимодействия с данными. Важно применять лямбда-выражения и LINQ-запросы для построения сложных запросов, что способствует улучшению читаемости кода и упрощает его сопровождение. Важно избегать частых вызовов к базе данных из разных процессов, так как это может привести к избыточной нагрузке и замедлению работы системы. В таких случаях целесообразно использовать кэширование данных.

Синхронизация данных и управление транзакциями требуют особого внимания, особенно в многопроцессорной среде. Это позволяет избежать конфликтов и обеспечить целостность данных. Важно настроить правильное управление транзакциями с использованием Entity Framework Core, чтобы минимизировать длительные блокировки таблиц и обеспечить эффективную работу системы.

Регулярное тестирование системы с разными объемами данных и в различных сценариях использования позволяет выявить возможные узкие места в производительности и устранить их на этапе разработки. Это особенно важно при внесении изменений в код или обновлении версий используемых библиотек. Отладка параллельных процессов требует особого подхода, так как ошибки, возникающие при параллельной обработке, могут быть сложно уловимыми. Использование инструментов для отладки и мониторинга параллельных приложений облегчает выявление и устранение таких ошибок.

Система должна быть спроектирована с учетом будущего роста данных и нагрузки. Важно предусмотреть возможность масштабирования как горизонтального, путем добавления новых серверов, так и вертикального, за счет увеличения ресурсов существующих серверов. В условиях изменяющейся нагрузки необходимо предусмотреть механизмы адаптации системы, такие как динамическое распределение задач между процессами или автоматическое регулирование числа процессов в зависимости от текущей нагрузки.

Следуя этим рекомендациям, можно обеспечить эффективное использование системы обработки клиентских заявок на основе MPI.NET и Entity Framework Core, максимизировать производительность и надежность работы в условиях постоянно растущих объемов данных и требований к быстродействию.

Конфликт интересов

Не указан.

Рецензия

Белашова Е.С., Казанский национальный исследовательский технический университет им. А.Н. Туполева – КАИ, Казань, Российская Федерация

Conflict of Interest

None declared.

Review

Belashova E.S., Kazan National Research Technical University named after A.N. Tupolev – KAI, Kazan, Russian Federation

Список литературы / References

1. Sabirov N.A. Parallel Processing of SQL Queries Using MPI.NET / N.A. Sabirov, R.F. Gibadullin // 2024 International Russian Smart Industry Conference (SmartIndustryCon). — Sochi, 2024. — P. 344–349. — DOI: 10.1109/SmartIndustryCon61328.2024.10516133.
2. Лавина Т.А. Анализ применения технологии объектно-ориентированного доступа к данным Entity Framework Core для создания приложений на платформе.Net / Т.А. Лавина, Е.С. Матвеев // Новые компетенции цифровой реальности: теория и практика их развития у обучающихся : сборник докладов и научных статей IV Всероссийской научно-практической конференции, Чебоксары, 06 апреля 2023 года. — Чебоксары: Чувашский государственный университет имени И.Н. Ульянова, 2023. — С. 212–222.
3. Бабушкина Н.Е. Сравнительная характеристика технологий доступа к данным Entity Framework / Н.Е. Бабушкина, А.А. Рачев // Инновационные технологии в машиностроении, образовании и экономике. — 2020. — Т. 27. — № 2 (16). — С. 8–11.
4. Алькаев Р.Р. Анализ возможностей систем управления базами данных MS access, MYSQL, MS SQL server / Р.Р. Алькаев, Д.Э. Водяков, В.А. Варюхин // APRIORI. Серия: Естественные и технические науки. — 2015. — № 6. — С. 2.
5. Маркелов М.А. Сравнительный анализ производительности СУБД MS SQL Server и PostgreSQL при одинаковой нагрузке / М.А. Маркелов, Д.А. Коростелев // Новые горизонты : сборник докладов IX научно-практической конференции с международным участием, Брянск, 07 апреля 2022 года. — Брянск: Брянский государственный технический университет, 2022. — С. 163–166.
6. Солодухо М.С. Разработка информационной системы книжного магазина в среде MS SQL Server / М.С. Солодухо, Г.Е. Ливянт // Актуальные проблемы социально-экономического развития современного общества : Сборник статей IV международной научно-практической конференции, Киров, 25 мая 2023 года / Под ред. М.П. Разина, Л.Н. Шмаковой, Н.С. Семено [и др.]. — Киров: Кировский государственный медицинский университет, 2023. — С. 647–654.
7. Волков М.Р. Идентификация и аутентификация в ASP.NET Core с использованием ASP.NET Core Identity / М.Р. Волков // Развитие науки и технологий в современной России (шифр — ВКРН). — Москва: Академическая среда, 2024. — С. 41–56.
8. Эсеналиева Г.А. Анализ применения ASP.NET при разработке информационной системы analysis of the ASP.NET development information system / Г.А. Эсеналиева, А.К. Супибекова // Вестник Кыргызстана. — 2017. — № 2 (4). — С. 280–283.
9. Лапина М.А. Исследование функционального моделирования бизнес-процессов на основе нотаций IDEF0 и еерс / М.А. Лапина, Н.В. Ржевская, А.С. Медведева // Вопросы современной науки и практики. Университет им. В.И. Вернадского. — 2023. — № 3 (89). — С. 65–75. — DOI: 10.17277/voprosy.2023.03.pp.065-075.
10. Мукина И.А. Проектирование инфологической, даталогической и физической моделей базы данных для хранения результатов расчета модифицированного метода анализа иерархий / И.А. Мукина, Д.С. Соловьев, Ю.В. Литовка // Наука сегодня: фундаментальные и прикладные исследования. — Вологда: Маркер, 2017. — Т. 1. — С. 51–53.
11. Кухарев Г.А. Алгоритмы формирования цветных QR-кодов для задач биометрии / Г.А. Кухарев, Н. Казиева // Научно-технический вестник информационных технологий, механики и оптики. — 2019. — Т. 19. — № 5. — С. 955–958. — DOI: 10.17586/2226-1494-2019-19-5-955-958.
12. Утеев Г. Разработка децентрализованной системы идентификации личности по биометрическим данным с помощью технологии блокчейн и компьютерного зрения / Г. Утеев, Р.Ф. Гибадуллин // Международный научно-исследовательский журнал. — 2024. — № 4 (142). — DOI: 10.23670/IRJ.2024.142.6.
13. Keller R. Memory debugging of MPI-parallel applications in open MPI / R. Keller, S. Fan, M. Resch // Advances in Parallel Computing. — 2008. — Vol. 15. — P. 517–523.
14. Акиншин Л. Процессоры Intel Core i3/i5/i7: что нового / Л. Акиншин // Компоненты и технологии. — 2010. — № 6 (107). — С. 109–112.
15. Begalin A.Sh. Perfomance testing of DDR4 and DDR3 memory cards / A.Sh. Begalin // 3i: Intellect, Idea, Innovation. — 2018. — № 1-2. — P. 3–9.

Список литературы на английском языке / References in English

1. Sabirov N.A. Parallel Processing of SQL Queries Using MPI.NET / N.A. Sabirov, R.F. Gibadullin // 2024 International Russian Smart Industry Conference (SmartIndustryCon). — Sochi, 2024. — P. 344–349. — DOI: 10.1109/SmartIndustryCon61328.2024.10516133.

2. Lavina T.A. Analiz primenenija tehnologii ob'ektno-orientirovannogo dostupa k dannym Entity Framework Core dlja sozdanija prilozhenij na platforme.Net [Analysis of the application of object-oriented data access technology Entity Framework Core for creating applications on the platform.Net] / T.A. Lavina, E.S. Matveev // Novye kompetencii cifrovoj real'nosti: teorija i praktika ih razvitija u obuchajushhihsja : sbornik dokladov i nauchnyh statej IV Vserossijskoj nauchno-prakticheskoj konferencii, Cheboksary, 06 aprelja 2023 goda [New competences of digital reality: theory and practice of their development in students: collection of reports and scientific articles of the IV All-Russian Scientific and Practical Conference, Cheboksary, 06 April 2023]. — Cheboksary: Chuvash State University named after I.N. Ulyanov, 2023. — P. 212–222. [in Russian]

3. Babushkina N.E. Sravnitel'naja harakteristika tehnologij dostupa k dannym Entity Framework [Comparative characteristics of Entity Framework data access technologies] / N.E. Babushkina, A.A. Rachev // Innovacionnye tehnologii v mashinostroenii, obrazovanii i jekonomike [Innovative technologies in engineering, education and economics]. — 2020. — Vol. 27. — № 2 (16). — P. 8–11. [in Russian]

4. Al'kaev R.R. Analiz vozmozhnostej sistem upravlenija bazami dannyh MS access, MYSQL, MS SQL server [Analysis of possibilities of database management systems MS access, MYSQL, MS SQL server] / R.R. Al'kaev, D.Je. Vodjakov, V.A. Varjuhin // APRIORI. Cerija: Estestvennye i tehnicheskie nauki [APRIORI. Series: Natural and Technical Sciences]. — 2015. — № 6. — P. 2. [in Russian]

5. Markelov M.A. Sravnitel'nyj analiz proizvoditel'nosti SUBD MS SQL Server i PostgreSQL pri odinakovoj nagruzke [Comparative analysis of MS SQL Server and PostgreSQL DBMS performance under the same load] / M.A. Markelov, D.A. Korostelev // Novye gorizonty : sbornik dokladov IX nauchno-prakticheskoj konferencii s mezhdunarodnym uchastiem, Brjansk, 07 aprelja 2022 goda. — Brjansk: Brjanskij gosudarstvennyj tehnicheskij universitet [New Horizons: collection of reports of the IX Scientific and Practical Conference with International Participation, Bryansk, 07 April 2022. - Bryansk: Bryansk State Technical University], 2022. — P. 163–166. [in Russian]

6. Soloduho M.S. Razrabotka informacionnoj sistemy knizhnogo magazina v srede MS SQL Server [Development of information system of bookstore in MS SQL Server environment] / M.S. Soloduho, G.E. Livjant // Aktual'nye problemy social'no-jekonomicheskogo razvitija sovremennogo obshhestva : Sbornik statej IV mezhdunarodnoj nauchno-prakticheskoj konferencii, Kirov, 25 maja 2023 goda [Current problems of socio-economic development of modern society: Collection of articles of the IV International Scientific and Practical Conference, Kirov, 25 May 2023] / Ed. by M.P. Razin, L.N. Shmakova, N.S. Semeno [et al.]. — Kirov: Kirov State Medical University, 2023. — P. 647–654. [in Russian]

7. Volkov M.R. Identifikacija i autentifikacija v ASP.NET Core s ispol'zovaniem ASP.NET Core Identity [Identification and authentication in ASP.NET Core using ASP.NET Core Identity] / M.R. Volkov // Razvitie nauki i tehnologij v sovremennoj Rossii (shifr — VKRN) [Development of Science and Technology in Modern Russia (cipher – VKRN)]. — Moscow: Academic Environment, 2024. — P. 41–56. [in Russian]

8. Jesenalieva G.A. Analiz primenenija ASP.NET pri razrabotke informacionnoj sistemy analysis of the ASP.NET development information system [ASP.NET application analysis of the ASP.NET development information system] / G.A. Jesenalieva, A.K. Supibekova // Vestnik Kyrgyzstana [Bulletin of Kyrgyzstan]. — 2017. — № 2 (4). — P. 280–283. [in Russian]

9. Lapina M.A. Issledovanie funkcional'nogo modelirovanija biznes-processov na osnove notacij IDEF0 i eepc [Research of functional modelling of business processes on the basis of IDEF0 and eers notations] / M.A. Lapina, N.V. Rzhevskaja, A.S. Medvedeva // Voprosy sovremennoj nauki i praktiki. Universitet im. V.I. Vernadskogo [Issues of Modern Science and Practice. V.I. Vernadsky University]. — 2023. — № 3 (89). — P. 65–75. — DOI: 10.17277/voprosy.2023.03.pp.065-075. [in Russian]

10. Mukina I.A. Proektirovanie infologicheskoj, datalogicheskoj i fizicheskoj modelej bazy dannyh dlja hranenija rezul'tatov rascheta modificirovannogo metoda analiza ierarhij [Designing infological, datalogical and physical database models for storing the calculation results of the modified method of hierarchy analysis] / I.A. Mukina, D.S. Solov'ev, Ju.V. Litovka // Nauka segodnja: fundamental'nye i prikladnye issledovanija [Science Today: Fundamental and Applied Research]. — Vologda: Marker, 2017. — Vol. 1. — P. 51–53. [in Russian]

11. Kuharev G.A. Algoritmy formirovanija cvetnyh QR-kodov dlja zadach biometrii [Algorithms of forming coloured QR codes for biometric tasks] / G.A. Kuharev, N. Kazieva // Nauchno-tehnicheskij vestnik informacionnyh tehnologij, mehaniki i optiki [Scientific and Technical Bulletin of Information Technologies, Mechanics and Optics]. — 2019. — Vol. 19. — № 5. — P. 955–958. — DOI: 10.17586/2226-1494-2019-19-5-955-958. [in Russian]

12. Uteev G. Razrabotka decentralizovannoj sistemy identifikacii lichnosti po biometricheskim dannym s pomoshh'ju tehnologii blokchejn i komp'juternogo zrenija [Development of decentralized system of personal identification by biometric data using blockchain technology and computer vision] / G. Uteev, R.F. Gibadullin // Mezhdunarodnyj nauchno-issledovatel'skij zhurnal [International Research Journal]. — 2024. — № 4 (142). — DOI: 10.23670/IRJ.2024.142.6. [in Russian]

13. Keller R. Memory debugging of MPI-parallel applications in open MPI / R. Keller, S. Fan, M. Resch // Advances in Parallel Computing. — 2008. — Vol. 15. — P. 517–523.

14. Akinshin L. Processory Intel Core i3/i5/i7: chto novogo [Intel Core i3/i5/i7 processors: what's new] / L. Akinshin // Komponenty i tehnologii [Components and Technologies]. — 2010. — № 6 (107). — P. 109–112. [in Russian]

15. Begalin A.Sh. Perfomance testing of DDR4 and DDR3 memory cards / A.Sh. Begalin // 3i: Intellect, Idea, Innovation. — 2018. — № 1-2. — P. 3–9.