МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра Математического и компьютерного моделирования

Разработка сервиса для планирования управления командной работой в организации

АВТОРЕФЕРАТ МАГИСТЕРСКОЙ РАБОТЫ

студентки 2 курса 247 группы

направление 09.04.03 — Прикладная информатика

механико-математического факультета

Замяткин Александр Геннадьевич

Научный руководитель
доцент, к.ф.-м.н. И.В. Плаксина

Зав. кафедрой
зав. каф., д.ф.-м.н., доцент Ю.А. Блинков

Саратов 2025

Введение. В современном мире управление проектами и задачами становится все более важной задачей для организаций, компаний и индивидуальных пользователей. В связи с этим возрастает потребность в эффективных инструментах, которые могут помочь в организации командной работы над проектами и задачами, улучшении производительности и оптимизации работы. Существует множество сервисов для управления проектами и задачами, но не все из них удовлетворяют потребностям пользователей. Большинство сервисов либо имеют слишком сложный интерфейс, либо имеют излишний функционал, либо не предоставляют достаточного функционала для эффективного управления проектами и задачами. Целью данной магистерской работы является разработка сервиса для планирования и управления командной работой в организации, который будет удобен в использовании, обладать достаточным функционалом и позволит эффективно управлять проектами и задачами.

В качестве целей данной работы можно выделить:

- Обоснование актуальности выбранной темы.
- Проектирование сервиса.
- Выбор технологий для реализации серверной части.
- Реализация серверной части.
- Создание пользовательского интерфейса.

Магистерская работа состоит из введения, 4 разделов, заключения, списка использованных источников, а так же приложений. В первом разделе происходит описание предметной области, а так же изучение актуальности разработки сервиса для планирования и управления командной работой в организации. Второй раздел посвящен проектированию сервиса для планирования и управления командной работой в организации, проектированию диаграммы прецедентов, проектированию информационной модели, а так же проектированию диаграммы классов. Третий раздел посвящен выбору технологическому стеку для разработки системы управления базой данных, которая является ядром любого проекта. В четвертом разделе происходит непосредственная реализация серверной части, описывается реализованные механизмы работы с базой данный, реализации системы авторизации и распределения ролей,

выбор паттерна веб-проектирования, а так же непосредственно реализация пользовательского интерфейса.

Описание предметной области. В данном разделе происходит описание того, что такое таск-трекер и для чего он используется. В процессе описания работы таск-трекера поднимается тема того, что в 21 веке, который является эпохой цифровых технологий, цифровизация методов управления позитивно влияет на продуктивность, качество продукта, времени и усилий, прилагаемых для реализации продукта, однако перед внедрением таск-трекера в работу организации возникает следующий вопрос: «Лучше использовать уже существующие решения или создать собственное». Далее на основании научных статьей происходит анализ некоторых популярных сервисов таск-трекеров, таких как YouTrack, Trello и Jira. В процессе анализа выяснилось, что YouTrack более ориентирован на разработку ПО, Trello предназначен для работы с карточками и не всегда удобен для управления проектами большого масштаба, а Jira может быть слишком сложным и неэффективным для малых компаний. Кроме того, создание собственного сервиса позволит компании интегрировать в него все необходимые функции, подстраивая их под свои потребности и процессы. Также это даст возможность разработать уникальные функциональные возможности, которые могут быть важны для данной компании, но не предусмотрены в готовых сервисах.

Проектирование. Данный раздел делится на несколько частей. В первой части происходит проектирование UML диаграммы прецедентов. Данная диаграмма необходима для:

- Идентификации функциональности.
- Определение актеров.
- Изучение взаимодействия.
- Определение границ системы.
- Создание основы для дальнейшего проектирования.

Далее происходит описание ролевой модели. Ролевая модель - это концепция, используемая в информационной безопасности и управлении доступом, которая основывается на присвоении определенных ролей пользователям и назначении разрешений на основе этих ролей. На основе ролевой модели была

разработана диаграмма прецедентов, в которой было определено три основных актера:

- Актёр «Администратор» - Это разработчик или директор. Администратор имеет наивысшие права доступа в системе.
- Актёр «Пользователь с полномочиями» - Это могут быть руководители отделов. Они имеют тот же функционал, что и администратор, за исключением того, что они не могут редактировать данные удалять пользователей.
- Акрёр «Пользователь» - Это обычный рядовой сотрудник. В отличии от «Администратора» и «Пользователя с полномочиями» он имеет доступ к базовому функционалу системы.

Следующая часть посвящена проектирование информационной модели. Информационная модель описывает структуру данных и отношения между ними в системе. Анализ предметной области позволил выделить следующие ключевые сущности проектируемой базы данных: Пользователи, полномочия, пользователи-полномочия, задачи, статусы, приоритеты, проект-задачи, проекты, работа, комментарии, задача-комментарий, файлы и файлы-комментарии. После описания сущностей и атрибутов базы данных была создана ER-диаграмма, которая является графическим инструментом для визуализации и описания логической структуры базы данных.

Последняя часть раздела посвящена проектирование диаграммы классов. Данная диаграмма позволит концептуально понять разработчикам как должны выглядеть классы, а также какие методы они в себе должны содержать.

Выбор технологий для реализации СУБД. Данный раздел посвящен выбору технологического стека, на основе которого будет реализована сервенная часть. Первая часть раздела посвящена описанию важность выбора базы данных. База данных является ключевой составляющей любой информационной системы, отвечая за хранение, обработку и доступ к данным. Ошибки, допущенные на этапе проектирования базы данных, могут привести к серьезным проблемам в будущем, включая низкую производительность, сложность внесения изменений и увеличение затрат на поддержку. Иными словами, выбор базы данных – это не просто техническое решение, а стратегический шаг, определяющий будущее всей системы. Грамотно выбранная

база данных позволяет создать стабильную, надежную и эффективную информационную систему, способную адаптироваться к изменениям и росту нагрузки. Ошибки на этом этапе могут привести к необходимости кардинальной переработки системы в будущем, что повлечет за собой дополнительные финансовые и временные затраты. Следующая часть раздела посвящена реляционной базе данных PostgreSQL. PostgreSQL — одна из самых мощных реляционных систем управления базами данных. В отличие от других баз данных, PostgreSQL сочетает строгую реляционную модель с расширенными возможностями обработки данных, такими как поддержка JSON, хранимых процедур и мощной системы индексов. Кроме того PostgreSQL является полностью открытой системой управления базами данных (open-source), что означает, что его исходный код доступен для свободного использования, модификации и распространения. Это делает PostgreSQL не только экономически выгодным решением по сравнению с коммерческими СУБД, но и обеспечивает его активное развитие благодаря большому сообществу разработчиков и компаний. Таким образом, PostgreSQL является одной из самых мощных и гибких реляционных СУБД, обеспечивающей высокую производительность, безопасность и надежность хранения данных. Последняя часть раздела посвящена технологиям для разработки серверной части. В качестве ядра таск-трекера было принято решение использовать платформу .NET Core.

.NET Core — это кроссплатформенный, высокопроизводительный и модульный фреймворк для разработки различных типов приложений, разработанный компанией Microsoft. Он является открытым исходным кодом и предназначен для создания веб-приложений, микросервисов, облачных решений и других типов приложений. Далее происходит описание основных особеностей и преимуществ данный платформы, а так же обоснование того, почему она подходит для реализации сервиса таск-трекера. Далее происходит описание технологии для взаимодействия с базой данных, а именно Entity Framework Core.

Entity Framework Core (EF Core) представляет собой современную версию фреймворка для работы с базами данных в экосистеме .NET. Этот инструмент позволяет разработчикам работать с данными через объекты, предоставляя удобные средства для создания, обновления, удаления и извлечения

информации из баз данных. EF Core разработан с учётом современных потребностей и тенденций в разработке программного обеспечения, таких как кроссплатформенность, производительность и гибкость. Здесь так же описываются основные преимущества EntityFramework, а также то, что он имеет полностью открытым исходным код, что позволяет разработчикам вносить изменения и улучшения в этот фреймворк. После чего происходит описание механизма миграций, который позволяет внедрять изменения в модель данных постепенно, минимизируя риски и обеспечивая чёткую документацию всех преобразований.

В конце раздела происходит выбор архитектуры информационной системы, которой стала многослойная архитектура.

Многослойная архитектура (также называемая трехуровневой архитектурой) - это подход к проектированию информационных систем, который разделяет систему на три отдельных уровня: представления (presentation), бизнес логики (business logic) и уровня доступа к данным (data access). Каждый уровень выполняет свои функции и взаимодействует с другими уровнями через строго определенные интерфейсы.

В процессе происходит описание основных особеностей и преимуществ данной архитектуры и подводится итог, что в случае создания сервиса таск-трекера можно использовать как клиент-серверную архитектуру, так и многослойную. Однако использование многослойной архитектуры будет наиболее подходящим выбором.

Реализация серверной части. В данном разделе происходит описание реализованного сервиса таск-трекера и его основных модулей. Данный раздел разделен на несколько частей. Первая часть данного раздела посвящена описанию механизмов и принципов работы базового класса DAL. Данный класс реализует концепцию обобщённого репозитория, позволяющего осуществлять типобезопасную и настраиваемую работу с различными типами сущностей. За счёт использования обобщённых параметров (generics) достигается высокая степень гибкости, что позволяет адаптировать реализацию для разных доменных моделей и контекстов данных.

В обобщённом классе BaseDal для этой цели реализован вспомогательный метод DisposeContextAsync в соответствии с рисунком 1, обеспечивающий

асинхронное освобождение ресурсов, занятых контекстом, в случаях, когда текущий экземпляр контекста отличается от внутренне используемого.

```
protected async Task<bool> DisposeContextAsync(TDbContext context)
{
    if (ReferenceEquals(context, _context)) return false;
    await context.DisposeAsync();
    return true;
}
```

Рисунок 1 — Асинхронное управление временем жизни контекста данных

Одной из фундаментальных задач в слое доступа к данным является извлечение уникального идентификатора сущности. Это необходимо для реализации базовых операций, таких как обновление, удаление и сравнение объектов. В рамках обобщённого класса BaseDal данная задача решается с использованием двух методов: абстрактного метода GetIdByEntityExpression, который определяет правило получения идентификатора, и вспомогательного метода GetIdByEntity в соответствии с рисунком 2, который выполняет извлечение значения на основе этого правила.

```
protected abstract Expression<Func<TEntity, TObjectId>> GetIdByEntityExpression();

protected TObjectId GetIdByEntity(TEntity entity)
{
    return GetIdByEntityExpression().Compile()(entity);
}
```

Рисунок 2 — Извлечение идентификатора сущности

В задачах выборки данных из базы часто требуется получить не одну конкретную сущность, а множество объектов, идентификаторы которых входят в заданный набор. Для решения этой задачи в обобщённом классе BaseDal реализован метод GetCheckDbObjectIdInArrayExpression в соответствии с рисунком 3, возвращающий предикат (выражение), пригодный для использования в LINQ-запросах, выполняемых на стороне базы данных.

Метод GetCheckDbObjectIdInArrayExpression предназначен для построения предиката, проверяющего, содержится ли идентификатор объекта TDbObject в заданном массиве TObjectId[] arr.

```
protected Expression<Func<TDbObject, bool>> GetCheckDbObjectIdInArrayExpression(TObjectId[] arr)
{
    var p = Expression.Parameter(typeof(TDbObject));
    Expression<Func<TObjectId, bool>> expr = item => arr.Contains(item);
    var method = (expr.Body as MethodCallExpression)?.Method;
    return Expression.Lambda<Func<TDbObject, bool>>(Expression.Call(null, method,
        Expression.Property(Expression.Constant(new { Arr = arr }), "Arr"),
        Expression.MakeMemberAccess(p, GetDbObjectIdMember())), p);
}
```

Рисунок 3 — Построение выражения фильтрации по массиву идентификаторов

Для выполнения точечной выборки объекта из базы данных по его уникальному идентификатору в обобщённом классе BaseDal реализован метод GetCheckDbObjectIdExpression в соответствии с рисунком 4. Он возвращает предикат в виде выражения, пригодного для использования в LINQ-запросах, трансформируемых в SQL, что особенно важно при работе с Entity Framework.

```
protected Expression<Func<TDbObject, bool>> GetCheckDbObjectIdExpression(TObjectId objectId)
{
    var p = Expression.Parameter(typeof(TDbObject));
    return Expression.Lambda<Func<TDbObject, bool>>(Expression.Equal(
        Expression.MakeMemberAccess(p, GetDbObjectIdMember()),
        Expression.Property(Expression.Constant(new { Id = objectId }), "Id")),
    p);
}
```

Рисунок 4 — Построение выражения фильтрации по конкретному идентификатору

В процессе сохранения данных в базу важно предусмотреть возможность выполнения дополнительной логики — как до момента сохранения, так и после. В обобщённом классе BaseDal предусмотрены два метода, позволяющие гибко внедрять такую логику в зависимости от специфики бизнес-объекта или доменной модели. Речь идёт о методах UpdateBeforeSavingAsync и UpdateAfterSavingAsync в соответствии с рисунком 5.

```
protected abstract Task UpdateBeforeSavingAsync(TDbContext context, TEntity entity, TDbObject dbObject, bool exists);

protected virtual Task UpdateAfterSavingAsync(TDbContext context, TEntity entity, TDbObject dbObject, bool exists)
{
    return Task.CompletedTask;
}
```

Рисунок 5 — Методы предварительной и постобработки

В рамках архитектуры обобщённого репозитория BaseDal особое внимание уделяется унифицированной обработке запросов к базе данных. Метод GetAsync реализует типовой сценарий — получение списка бизнес-сущностей (TEntity), соответствующих заданному предикату, с учётом параметров преобразования в соответствии с рисунком 6. Он инкапсулирует процесс фильтрации, трансформации и управления временем жизни контекста базы данных.

```
internal virtual async Task<IList<TEntity>> GetAsync(Expression<Func<TDbObject, bool>> predicate,
    TConvertParams convertParams = null)
{
    var data = GetContext();
    try
    {
        return await BuildEntitiesListAsync(data, data.Set<TDbObject>().Where(predicate),
            convertParams, false);
    }
    finally
    {
        await DisposeContextAsync(data);
    }
}
```

Рисунок 6 — Извлечение бизнес-сущностей по предикату: метод GetAsync

Метод GetAsync, принимающий параметры поиска типа TSearchParams, представляет собой базовую виртуальную реализацию механизма выборки данных с поддержкой фильтрации, сортировки и постраничной загрузки. Он предназначен для асинхронного получения набора бизнес-сущностей TEntity, соответствующих заданным условиям, и формирования результата в виде структуры SearchResult<TEntity>, содержащей как список объектов, так и сопутствующую информацию о параметрах выборки.

Кроме вышеописанных методов также происходит детальное описание методов предназначеных для получения объектов базы данных путём использования сортировочных параметров, методы как для множественного удаления, так и удаления по определенному идентификатору, так же описание не менее важной группы методов про добавление и обновление как определенного объекта, а так же группы объектов.

В следующих частяхра раздела приводится пример реализации сущностей базы данных на примере сущности User. Здесь демонстрируется и описывается реализация данной сущности на каждом слое многослойной архитектуры,

а именно реализация на уровнях DAL и BL. Кроме того описывается механизм авторизации и распределению полномочий при помощи Claim.

Последняя часть раздела посвящена реализацию пользовательского интерфейса. В первую очередь описывается причины выбора паттерна веб-проектирования MVC.

Далее демонстрируются страницы реализованного сервиса таск-трекера. Для того, чтобы воспользоваться сервисом таск-трекером необходимо будет авторизоваться на главной странице в соответствии с рисунком 7.

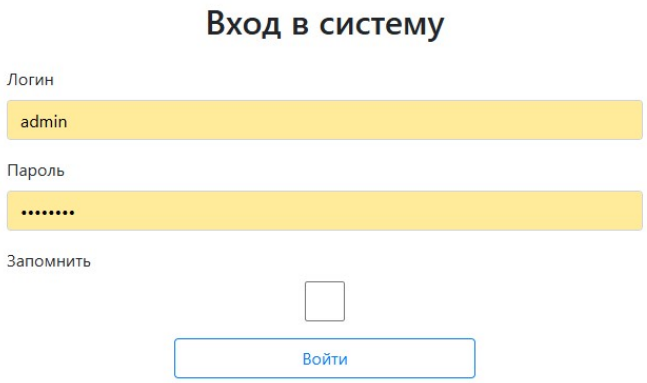

Рисунок 7 — Страница авторизации

Далее описывается то, что получить доступ к данной системе можно только после того, как произойдет регистрация нового пользователя другим пользователем имеющим для этого достаточные права доступа.

После авторизации сотрудник попадает на главную страницу в соответствии с рисунком 8. Главная страница содержит в себе список проектов из таблицы «Проекты». С ростом количества проектов возникает сложность в поиске конкретного проекта из списка. Для этого реализована функция поиска проектов по ключевым словам.

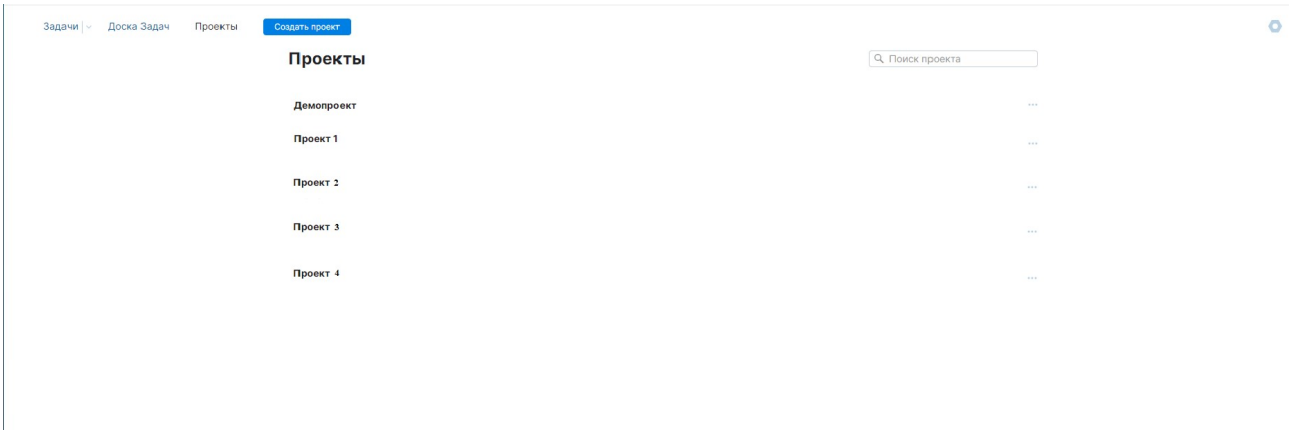

Рисунок 8 — Главная страница

Далее демонстрируется страница со списком всех задач в соответствии с рисунком 9. На этой страницы реализован функционал поиска задач по разным фильтрам.

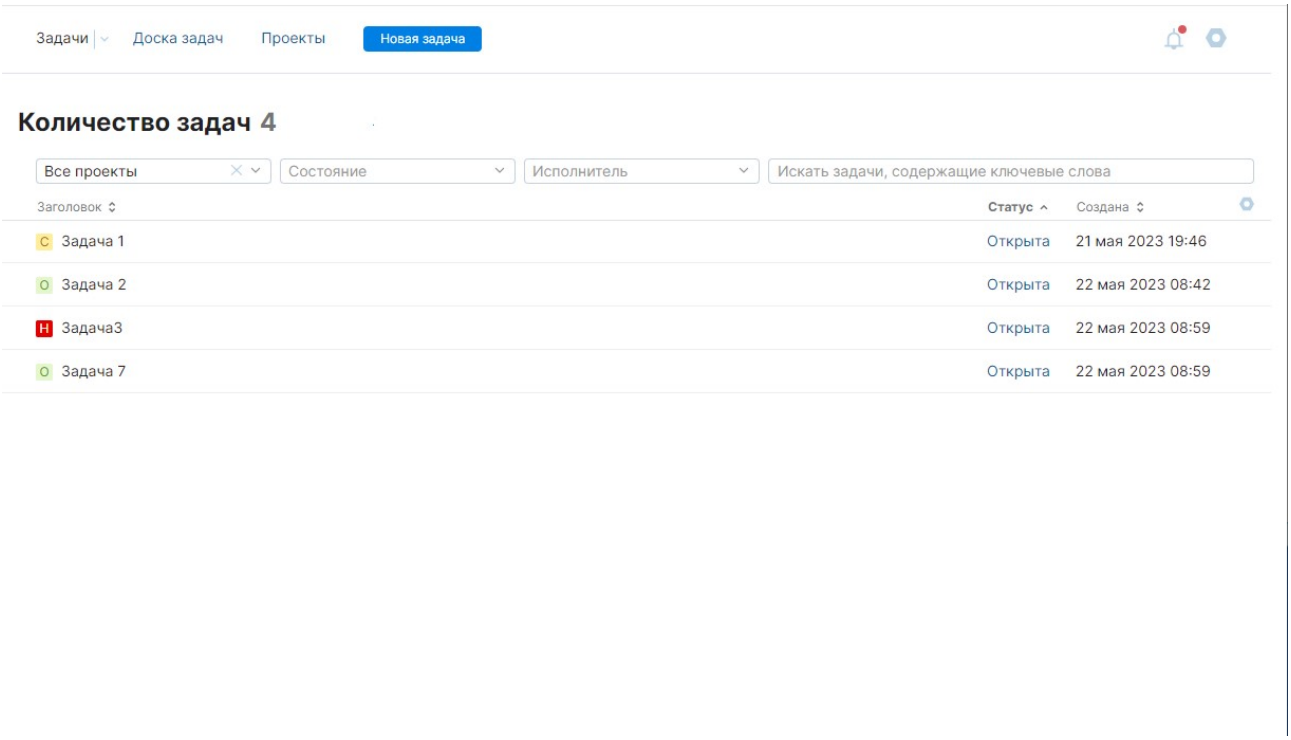

Рисунок 9 — Страница со списком задач

В заключении демонстрируется страница задачи в соответствии с рисунком 10, а так же описывается функционал позволяющий пользователям оставлять комментарии, прикреплять файлы и прочее.

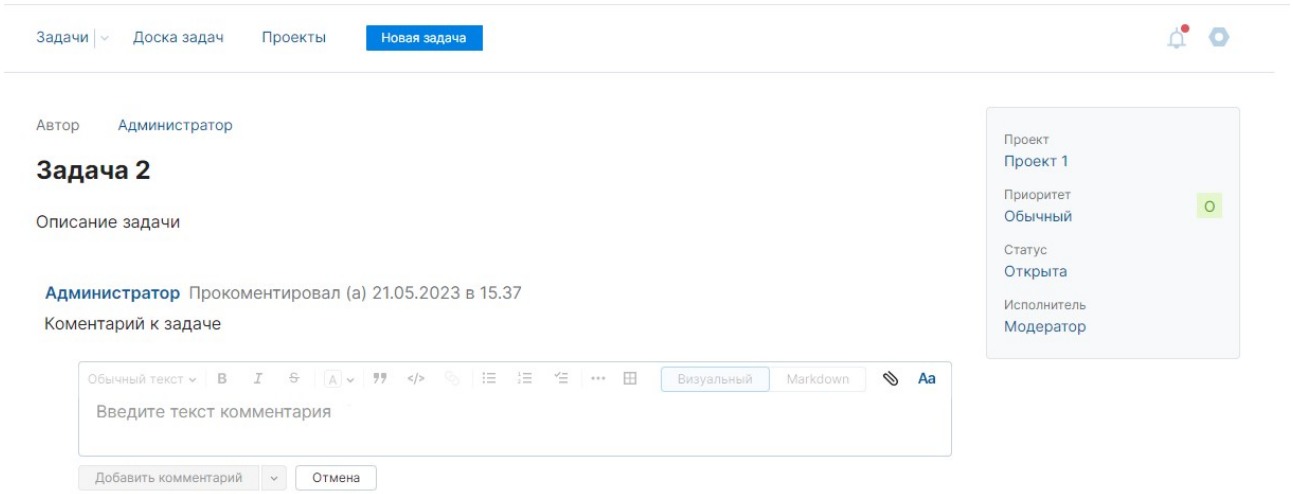

Рисунок 10 — Страница задачи

Заключение. В результате выполнения магистерской работы был реализован сервис таск-трекер. В процессе реализации был изучен ряд научных источников, получен ряд теоретических и практических навыков. Таким образом, все поставленные задачи магистерской работы были успешно выполнены, а цель достигнута.