МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра дифференциальных уравнений и математической экономики

РАЗРАБОТКА ПЛАТФОРМЫ ДЛЯ КУРСОВ С ИСПОЛЬЗОВАНИЕМ МИКРОСЕРВИСОВ

АВТОРЕФЕРАТ МАГИСТЕРСКОЙ РАБОТЫ

студента 2 курса 248 группы
направления 09.04.03 — Прикладная информатика

механико-математического факультета
Гнатюка Андрея Евгеньевича

Научный руководитель
профессор, д ф.-м. н., доцент _____ А. Ю. Трынин

Заведующий кафедрой
зав. кафедрой, д. ф.-м. н., доцент_____ В. С. Рыхлов

Саратов 2026

Введение. **Актуальность работы:** заключается в том, что в условиях цифровой трансформации образования наблюдается стремительный рост спроса на онлайн-обучающие платформы. Современные образовательные системы должны обеспечивать высокую доступность, гибкость в настройке контента, поддержку пиковых нагрузок во время массового обучения и быструю адаптацию к изменяющимся требованиям бизнеса. Традиционные монолитные архитектуры приложений зачастую не способны удовлетворить эти потребности, так как требуют полной пересборки и развертывания системы даже при незначительных изменениях одного из функциональных модулей.

В связи с этим, перспективным направлением является переход к микросервисной архитектуре, которая позволяет разрабатывать, масштабировать и развертывать сервисы независимо друг от друга. Однако проектирование и реализация такой архитектуры сопряжены с рядом сложностей: необходимость обеспечения надежной коммуникации между сервисами (Message Broker), управление транзакциями, обработка больших файлов и создание безопасной системы авторизации, способной работать в распределенной среде.

Целью магистерской работы является разработка и практическая реализация платформы для образовательных курсов на основе микросервисной архитектуры, обеспечивающей высокую масштабируемость, отказоустойчивость и удобство сопровождения.

Объектом исследования являются процессы проектирования, разработки и внедрения распределенных программных систем на основе микросервисной архитектуры в сфере образовательных технологий.

Предметом исследования выступают архитектурные паттерны, технологии и инструменты, используемые при создании масштабируемых образовательных платформ, включая контейнеризацию, брокеры сообщений, объектные хранилища и современные фреймворки разработки.

Для достижения необходимой цели в работе, необходимо решить следующие **задачи**:

— Анализ современных тенденций в разработке серверной части ПО, включая изучение микросервисного подхода и архитектурных паттернов (Clean Architecture, DDD);

— Выбор технологического стека для реализации образовательной платформы;
— Проектирование общей архитектуры приложения, выделение границ микросервисов и описание их взаимодействия;
— Реализация ключевых микросервисов: сервиса авторизации, сервиса управления файлами (MinIO), сервиса курсов и сервиса уведомлений;
— Разработка клиентской части приложения с использованием React для взаимодействия с пользователем.

Практическая значимость данной работы заключается в том, что сфера онлайн-образования продолжает активно развиваться, предоставляя новые возможности для учебных заведений, корпоративных университетов и независимых образовательных проектов. В современном мире образовательные платформы становятся основой для многих сервисов, включая школы программирования, курсы повышения квалификации, университетские LMS и многие другие, поскольку они обеспечивают удобство и доступность для пользователей, имеющих доступ к интернету с любого устройства. Кроме этого, многие традиционные офлайн-образовательные учреждения уже внедряют онлайн-платформы в свою работу, что демонстрирует тенденцию к цифровой трансформации и расширению доступа к знаниям. Это подчеркивает необходимость разработки образовательных платформ, способных адаптироваться к изменяющимся потребностям пользователей и бизнеса, обеспечивая высокую степень масштабируемости, отказоустойчивости и эффективности в использовании, чему в полной мере отвечает предложенное в работе микросервисное решение.

Структура и содеражание магистерской работы. Работа состоит из введения, четырех разделов, заключения и списка использованных источников, содержащего 27 наименований и 4 приложений. Общий объем работы составляет 66 страниц.

Основное содержание работы. Во **введении** обосновывается актуальность темы работы, а также формулируются цели работы.

В **первом** разделе описываются современные тенденции в разработке серверной части приложения.

Во **втором** разделе приводится описание используемых технологий и

инструментов, которые используются для реализации приложения.

В **третьем** разделе описывается архитектура микросервиса. В приложении используется чистая архитектура, которая включает в себя:

— уровень домена;
— уровень бизнес-логики;
— уровень инфраструктуры;
— уровень представления.

Уровень домена хранит все бизнес-сущности приложения. Важно понимать, что данный уровень не может ссылаться на вышелажищие уровни. Задача верхних слоев, это инкапсуляция бизнес сущностей от объектов реального мира. Также на данном уровне определены специальные классы DTO (data transfer object), которые содержат только ту информацию, которая необходима для того или иного слоя приложения.

Уровень бизнес-логики содержит в себе классы-сервисы, которые отвечают за обработку данных, полученных от уровня представления, и манипулирование ими до того, как они будут представлены пользователю или сохранены в базе данных.

Инфраструктурный уровень содержит все необходимое для общения приложения с внешним миром (пользователями, сторонними сервисами, железом и т.д). Данный уровень может очень быстро разростаться ввиду настройки большого количества интеграций.

Инфраструктурный код позволяет соединить ядро приложения с:

— Файловой системой;
— Сетью;
— ORM;
— Брокерами сообщений.

На данном уровне не может быть никакой бизнес-логики.

Уровень представления представляет собой Web API и включает в себя следующие элементы:

— Контроллеры - представляют собой endpoint-ы, принимающие запросы от клиента и передающие данные для обработки нижним слоям приложения, а именно в слой Application (приложения);
— Middleware - представляют собой компонент конвеера обработки запро-

са, позволяя обработать запрос до того, как он попадет в метод контроллера или после того, как запрос был обработан контроллера;

— Хранение глобальных настроек приложения;

Кроме этого, проект Web API подключает к приложению все используемые в приложении зависимости, находящиеся на нижих уровнях, а также является точкой запуска приложения.

В **четвертом** разделе описывается реализация ключевых микросервисов приложения. Данный раздел включает в себя 4 подраздела:

— Сервис файлов;

— Сервис уведомлений;

— Сервис авторизации;

— Сервис курсов.

Сервис файлов. Для реализации сервиса файлов реализованые его доменные сущности.

```
public class FileUploadDto
{
    public string BucketName { get; set; }
    public string ObjectName { get; set; }
    public IFormFile File { get; set; }
    public string ContentType { get; set; }
}
```

Для выполнения операций с файловым сервисом в нем реализован специальный класс-сервис, который отвечает за логику работы сервиса.

```
public async Task UploadFileAsync(FileUploadDto dto)
{
    try
    {
        await EnsureBucketExistsAsync(dto.BucketName);

        var tempFilePath = await CreateTempFileAsync(dto.File);

        var putObjectArgs = new PutObjectArgs()
            .WithBucket(dto.BucketName)
            .WithObject(dto.ObjectName)
            .WithFileName(tempFilePath)
            .WithContentType(dto.ContentType);
```

```
            await _minioClient.PutObjectAsync(putObjectArgs);

            CleanupTempFile(tempFilePath);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error uploading file to MinIO");
            throw;
        }
    }
```

Затем в приложении реализованы специальные методы API, к которым обращаются сторонние сервисы для работы с файлами.

Сервис авторизации. Для представления пользователя в приложении реализован специальный класс.

```
public sealed class User : IAuditable
{
    public Guid Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Login { get; set; }
    public string Password { get; set; }
    public Cart Cart { get; set; }
    public UserToken UserToken { get; set; }
    public IEnumerable<Order> Orders { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime? UpdatedAt { get; set; }
}
```

Процесс авторизации и аутентификации пользователя в приложении реализован на основе Bearer авторизации с использованием access и refresh токенов. Использование данного механизма авторизации позволяет пользователю длятельное время иметь доступ к защищенным ресурсам, а также иметь возможность выполнять действия в приложения, требующие авторизации без необходимости повторого входа в аккаунт.

Access - токен - это короткоживущий токен, который используется для получения доступа к ресурсам, требующие авторизацию. В зависимости от приложения, время жизни данного токены может составлять от нескольких

минут до нескольких часов, в зависимости от нужного уровня защиты данных пользователя.

Refresh - токен - это долгоживущий токен, который используется для обновления access токена пользователя. Срок жизни refresh токена может составлять от нескольких дней до нескольких месяцев. После истечения времени жизни refresh токена, приложение запрашивает повторную авторизацию и выдает новую пару access и refresh токенов.

Данные токены хранятся в cookie на стороне пользователя и имеют флаг HttpOnly, что позволяет избежать межсайтового скриптинга (XSS - внедрение вредоносного клиентского кода), делая недопустимым получение данных кук через JavaScript.

В приложении на стороне клиента определены специальные формы для регистрации/авторизации. Для отправки данных форм на клиенте опредены специальные методы, которые отправляют запрос для обработки на сервер.

На стороне сервера определены специальные конечные точки, которые обрабатывают данные запросы (на регистрацию/авторизацию). Данные конечные точки вызывают соответствующие методы сервиса, которые отвечают за логику ругистрации или авторизации пользователя.

Важно отметить, что конфиденциальная информация, такая как пароль пользователя не хранится в базе данных в чистом виде. Для защиты пароля в приложении реализован механизм его хэширования.

При обращения клиента к конечным точкам, которые требует авторизации, то токены приложению необходимо получать из отправляемых на сервер cookie. Для этого реализован специальный middleware, который встроен в конвеер обработки запросов, поэтому, при каждом обращении к приложению данный middleware будет извлекать и валидировать содержащийся в cookie токен и, в случае, если токен не валиден, отправлять соотвествующий статус код клиенту.

В случае, если access - токен истек, а refresh нет, то на стороне клиента встроен интерсептор, который автоматически отправляет запрос на сервер для обновления access токена и повторяет предыдущий запрос. А в случае, если истек и access токен, и refresh токен, то пользователю необходимо будет заново авторизоваться в приложении.

Реализация сервиса уведомлений. Для реализации сервиса уведомлений реализованы его доменные сущности.

```
public class Notification
{
    public Guid Id { get; set; }
    public Guid UserId { get; set; }
    public Channel Channel { get; set; }
    public string Subject { get; set; }
    public string Body { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime? SentAt { get; set; }
    public Status Status { get; set; }
    public int RetryCount { get; set; }
    public string? ErrorMessage { get; set; }
}


public class NotificationTemplate
{
    public Guid Id { get; set; }
    public string TemplateKey { get; set; }
    public string Channel { get; set; }
    public string SubjectTemplate { get; set; }
    public string BodyTemplate { get; set; }
    public Dictionary<string, string> DefaultParams { get; set; }
}
```

Также в сервисе определены настройки подключения к брокеру kafka, в который поступают события об отправки уведомлений. Для получения и их дальнейшей обработки в сервисе реализован worker, который вычитывает сообщения из kafka и обрабатывает их. Кроме этого, для хранения данных, которые не меняются на протяжении долгого времени, используется кэш Redis.

Реализация сервиса курсов. Для реализации сервиса курсов реализованы его доменные сущности.

```
public class Course
{
    public Guid Id { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public string ShortDescription { get; set; }
```

```
    public decimal Price { get; set; }
    public string CoverImageUrl { get; set; }
    public Level Level { get; set; }
    public int DurationInHours { get; set; }
    public string InstructorId { get; set; }
    public string InstructorName { get; set; }
    public bool IsPublished { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime? UpdatedAt { get; set; }
    public DateTime? PublishedAt { get; set; }

    public virtual ICollection<Lesson> Lessons { get; set; }
    public virtual ICollection<UserCourseProgress>
        UserProgresses { get; set; }
}
```

Также определены классы-сервисы с бизнес-логикой работы с курсами. Для взаимодействия с сервисом в нем реализованы API-методы.

```
[HttpGet]
[AllowAnonymous]
public async Task<IActionResult> GetCourses(
    [FromQuery] int page = 1,
    [FromQuery] int pageSize = 10,
    [FromQuery] string? search = null,
    [FromQuery] Level? level = null,
    [FromQuery] bool? isPublished = true)
{
    var result = await _courseService
        .GetCoursesAsync(page, pageSize, search, level, isPublished);

    return Ok(result);
}
```

В **заключении** приведены результаты магистерской работы. **Основные резльтаты:**

1. Спроектирована масштабируемая архитектура приложения, обеспечивающая его гибкость и возможность расширения, а также отказоустойчивость.
2. Создан интуитивно понятный и удобный пользовательский интерфейс (UI) для приложения.

3. Реализован полный набор функций и сервисов, необходимых для работы образовательной платформы.
4. Обеспечен надежный обмен данными между клиентом и сервером, а также межсервисной коммуникации.