

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Т.И. Вишневская, Т.Н. Романова

**ПРАКТИКУМ ПО РАЗРАБОТКЕ
РАСПРЕДЕЛЕННЫХ СИСТЕМ ОБРАБОТКИ ИНФОРМАЦИИ**

Учебно-методическое пособие

для студентов, обучающихся по направлению 09.04.04 «Программная инженерия»

(С) 2019 МГТУ им. Н.Э. БАУМАНА

УДК 004.41

ББК 34.9

*Рекомендовано Научно-методическим советом
МГТУ им. Н.Э. Баумана в качестве учебно-методического пособия*

Рецензент

к.н., доцент Задорожная Наталья Михайловна

Вишневская Т.И., Романова Т.Н.

Практикум по разработке распределенных систем обработки информации : учебно-методическое пособие / Т. И. Вишневская, Т. Н. Романова. – издательство МГТУ им. Н.Э. Баумана, 2018. – объем 90 с.: 25 ил.

Представлены методические указания по организации самостоятельной работы студентов при проведении учебной практики по разработке распределенных систем обработки информации (РСОИ). Даны рекомендации по выбору и использованию методологий и технологий программной инженерии для разработки РСОИ. Сформулированы основные этапы учебной практики и требования к оформлению отчета по итогам практики.

Для студентов специальности «Программная инженерия», изучающих информационные технологии в рамках дисциплины «Практикум по разработке РСОИ».

УДК 004.41

ББК 34.9

© МГТУ им. Н.Э. Баумана, 2019

© Вишневская Т.И., Романова Т.Н., 2019

© Оформление. Издательство МГТУ им. Н.Э. Баумана

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ	4
1. ОРГАНИЗАЦИЯ ПРАКТИКИ	5
1.1 Цель и задачи практики.....	5
1.2 Тема задания по практике	6
1.3 Основные требования к разрабатываемой РСОИ.....	7
2. ОСНОВНЫЕ ЭТАПЫ ПРАКТИКИ	7
2.1 Индивидуальное задание на практику.....	7
2.2 Выполнение проектных и технологических операций.....	8
2.3 Оформление отчета по практике.....	9
3. РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ И ОФОРМЛЕНИЮ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ	10
3.1 Архитектура РСОИ	10
3.2 Аналитический раздел.....	12
3.2.1 Техническое задание.....	12
3.2.2 Топология системы.....	13
3.3 Конструкторский раздел.....	13
3.3.1 Концептуальный дизайн	14
3.3.2 Логический дизайн	14
3.4 Технологический раздел.....	15
3.4.1 Выбор технологии для программной реализации.....	16
3.4.2 Программная реализации РСОИ	18
4. ПРИМЕР ВЫПОЛНЕНИЯ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ.....	22
ЛИТЕРАТУРА.....	88

ПРЕДИСЛОВИЕ

Предлагаемое учебно-методическое пособие предназначено для студентов 1-ого курса МГТУ им. Н.Э. Баумана второго семестра магистратуры по направлению 09.04.04 «Программная инженерия», изучающих дисциплину «Практикум по разработке РСОИ».

Пособие выполнено в соответствии с государственными образовательными стандартами и рабочей программой дисциплины «Практикум по разработке РСОИ» по направлению «Программная инженерия».

Цель учебно-методического пособия – способствовать овладению навыками программной реализации распределенных информационных систем.

Учебная практика «Практикум по разработке РСОИ» - это технологическая практика по получению профессиональных умений и опыта профессиональной деятельности по разработке РСОИ. Обязательной дисциплиной для освоения до проведения практики является учебный курс «Распределенные системы обработки информации».

Учебно-методическое пособие состоит из четырех разделов. В первом разделе сформулировано задание на учебную практику. Во втором разделе подробно описаны все этапы практики и требования к оформлению отчета по итогам практики. В третьем разделе сформулированы методические указания по выбору и использованию методологий и технологий программной инженерии для разработки РСОИ. В четвертом разделе приведен пример выполнения индивидуального задания на учебную практику по разработке распределенной системы «Информационный портал для обмена публикациями» с использованием перспективной сервисно-ориентированной архитектуры.

После изучения материалов учебно-методического пособия студенты смогут самостоятельно выполнить индивидуальное задание по дисциплине «Практикум по разработке РСОИ» и в рамках данной производственно-технологической деятельности овладеть следующими практическими навыками:

- выбора подходящей технологической платформы для разработки ПО;

- выбора правильных компромиссных решений при проектировании архитектур сложных программных комплексов в соответствии с современными стилями;
- разработки требований к характеристикам качества программного продукта;
- использования синхронных и асинхронных способов обмена информацией в распределенных информационных системах (РИС);
- использования приемов синхронизации данных в РИС;
- устранения противоречивых состояний в информационной системе (повторная попытка выполнения, отмена всей операции, распределенные транзакции);
- реализации авторизации, в том числе token-based авторизации;
- написания отказоустойчивых операций в распределенных системах;
- реализации процессов масштабирования системы с использованием сервисно-ориентированной архитектуры.

Авторы выражают благодарность студентам факультета «Информатика и системы управления» МГТУ им. Н.Э. Баумана Лавреновой Елизавете Алексеевне, Соломатиной Дарье Игоревне и Дружицкому Ивану Сергеевичу за помощь в разработке программного проекта по предложенной методике.

1. ОРГАНИЗАЦИЯ ПРАКТИКИ

«Практикум по разработке РСОИ» входит в вариативную часть Блока 2 «Практики (Учебная)» образовательной программы магистратуры по направлению 09.04.04 «Программная инженерия» и проводится в течении 2-го семестра.

1.1 Цель и задачи практики

Цель учебной практики: овладение навыками проектирования и программной реализации распределенных информационных систем.

Задачи практики:

- закрепление на практике имеющихся и приобретение новых специализированных знаний, умений, навыков и компетенций по разработке РСОИ;
- анализ существующих проблем в выбранной предметной области и определение возможных путей для их решения;
- анализ существующего программного обеспечения (ПО) предметной области. Обзор методов, алгоритмов и программного обеспечения, которые могут быть использованы для решения выявленных проблем предметной области и разработки нового ПО;
- в случае доработки существующего ПО следует провести анализ методов модификации и определить новые функциональные возможности ПО, которые необходимо разработать в рамках поставленной индивидуальной задачи.

По результатам прохождения практики планируется формирование следующих Собственных профессиональных компетенций, предусмотренных основной профессиональной образовательной программой СУОС МГТУ им. Н.Э. Баумана по направлению подготовки магистра 09.04.04. «Программная инженерия»:

- Способностью проектировать распределенные информационные системы, их компоненты и протоколы их взаимодействия.
- Владение навыками программной реализации распределенных информационных систем.

1.2 Тема задания по практике

Совместно с руководителем учебной практики студент должен выбрать тему индивидуального задания по практике (примеры тем приведены в разделе 2.1 данного пособия). Затем написать Техническое задание (ТЗ) на разработку РСОИ согласно выбранной теме с учетом требований к системе, изложенных в п. 1.3. В соответствии с ТЗ должно быть выполнено проектирование распределенной системы и её реализация. По результатам проведенной работы студент должен предоставить отчет.

1.3 Основные требования к разрабатываемой РСОИ

Используя теоретический материал, изучаемый ранее в рамках дисциплины «Распределенные системы обработки информации», разработать РСОИ по выбранной теме. Система должна удовлетворять следующим общим требованиям:

- для построения системы использовать сервисно-ориентированную архитектуру (COA);
- в основе построения микросервисов должны лежать универсальные технологии, выбор протоколов взаимодействия между сервисами необходимо обосновать;
- данные сервисов должны храниться в базе данных;
- система должна обеспечить сбор статистики о пользовательских операциях в соответствии с выделенной ролью для пользователя;
- используя современные технологии обеспечить отказоустойчивость разработанной системе;
- предусмотреть вход в систему, как через интерфейс приложения, так и через популярные социальные сети;
- система должна легко разворачиваться на нескольких серверах.

2. ОСНОВНЫЕ ЭТАПЫ ПРАКТИКИ

Учебная практика по разработке РСОИ включает следующие этапы:

1. Получение индивидуального задания на практику.
2. Выполнение проектных и технологических операций.
3. Оформление отчета по практике.

2.1 Индивидуальное задание на практику

После проведения организационных собраний студентов по вопросам практики, где студенты знакомятся с целями, задачами, содержанием и формами проведения практики, студенты узнают фамилию своего руководителя практики. Затем студенты совместно со своими руководителями выбирают

тему РСОИ и формулируют конкретные задачи индивидуального задания.

Разработка распределенной системы возможна как индивидуально, так и небольшими группами. Если разрабатываемая система большая и сложная, то допускается коллективная разработка, но не более трёх студентов в одной группе.

Примеры тем индивидуальных заданий для «Практикума по РСОИ»:

- Система «Фотосток с автоматическим аннотированием изображений».
- Система удаленного мониторинга состояния транспортного средства.
- Система для адаптивного распределения заданий в облаке.
- Web-портал краткосрочной аренды автомобиля с поминутной или почасовой оплатой для автовладельцев и клиентов.
- Распределенная система управления медицинской клиникой.
- Веб-сайт для управления планированием задач.
- Интернет-магазин электронных книг.
- Планировщик туристических и досуговых маршрутов по городам России.
- Портал для поиска музыки из фильмов.
- Портал для поиска и бронирования гостиниц.
- Информационная система для поиска и бронирования авиабилетов.
- Мобильное приложение для мониторинга сетей приборов учета
- Сервис по бронированию отелей.

2.2 Выполнение проектных и технологических операций

Процесс разработки РСОИ включает в себя:

- написание Технического задания (ТЗ);
- проектирование системы на языке UML и построение архитектуры распределенной системы;
- программная реализация распределенной системы на одном из кроссплатформенных языков программирования.

В третьем разделе данного пособия даны методические рекомендации по выполнению и оформлению индивидуального задания, а в четвертом разделе, на

примере разработки конкретной Распределенной системы обработки информации «Информационного портал для обмена публикациями», продемонстрированы все этапы разработки РСОИ.

2.3 Оформление отчета по практике

По результатам практики студент оформляет отчет и сдает руководителю практики. Руководитель практики проверяет правильность выполнения задания и оформления отчета. Контроль результатов производственной практики студента проходит в форме *дифференцированного зачета*, оценка вносится в зачетную ведомость и зачетную книжку студента (в раздел Практика).

Структура отчета студента по практике

1. Титульный лист

На титульном листе указывается официальное название МГТУ им. Н.Э. Баумана, факультета, выпускающей кафедры, ФИО студента, группа, название практики, должности и ФИО руководителя практики.

2. Содержание (оглавление)

3. Введение

В данном разделе должны быть приведены цели и задачи практикума в контексте индивидуального задания, а также описано назначение системы и обоснование её актуальности.

4. Аналитический раздел.

В разделе должна быть дана характеристика предметной области (в соответствии с целями и задачами программы практикума и индивидуальным заданием). Сделан анализ существующих проблем и возможных методов их решения и обзор методов, алгоритмов и стандартного программного обеспечения, которые могут быть использованы для решения проблем предметной области. Обоснован выбор подходящих методов и платформы для разработки распределенной системы.

5. Конструкторский раздел.

В данном разделе могут быть описаны возможные модификации выбранных методов и их применение для реализации разрабатываемой системы, если это

необходимо. Должен быть подробно представлен проект всей системы на языке UML и IDEF0, а также её архитектура.

6. Технологический раздел.

Обосновывается выбор языка программирования, фреймворка, ОС и СУБД. Приводятся коды всех подсистем РСОИ.

7. Заключение.

В заключении должны быть представлены краткие выводы по результатам практикума.

8. Список использованных источников.

9. Приложения.

В четвертом разделе данного учебно-методического пособия приведен пример отчета по учебной практике.

3. РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ И ОФОРМЛЕНИЮ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

Качество распределенной системы во много зависит от правильного выбора архитектуры информационной системы, подходящей платформы для разработки и языка программирования.

Этим вопросам уделяется особое внимание при проектировании системы.

3.1 Архитектура РСОИ

На сегодняшний день основными стилями архитектуры информационных систем являются монолитный и сервисно-ориентированный. Наиболее предпочтительным архитектурным стилем на сегодняшний день остается сервисно-ориентированная архитектура (СОА) благодаря ряду преимуществ по сравнению с монолитным стилем: раздельное развертывание сервисов, легкость сопровождения, масштабируемость и др. Однако в некоторых случаях разделение системы на отдельные сервисы может быть излишним, например, в случае небольших приложений или тесных связей.

Монолитный простой подход имеет ряд ограничений. Успешные приложения

имеют тенденцию расти со временем и в конечном итоге становятся огромными. Одна из основных проблем заключается в том, что приложение становится чрезвычайно сложным, чтобы любой разработчик мог полностью его понять. В результате исправление ошибок и внедрение новых функций становится сложным и трудоемким процессом. Возрастающий размер приложения также замедлит его развитие и сделает невозможным непрерывное развертывание. Монолитные приложения также могут быть трудно масштабируемыми, поскольку разные модули могут иметь противоречивые требования к ресурсам.

Борьба с всё возрастающей сложностью программного обеспечения привела к появлению нового шаблона архитектуры – СОА. Идея этого подхода состоит в том, чтобы разделить приложение на множество меньших по объему, но взаимосвязанных сервисов. Каждая отдельная функциональность в приложении может быть реализована посредством собственного микросервиса [1]. Веб-сервисы позволяют решать проблемы интеграции на предприятиях, объединяя различные приложения в один производственный процесс. Веб-сервисы представляют собой оболочку, обеспечивающую стандартный способ взаимодействия с прикладными программными средами. Интерфейсы веб-сервисов получают из сетевой среды стандартные XML-сообщения, преобразуют XML-данные в формат, "понимаемый" конкретной прикладной программной системой, и отправляют ответное сообщение (но могут и не отправлять его). Проведенный анализ подтверждает актуальность подхода к разработке РСОИ на основе сервисно-ориентированного подхода.

На сегодняшний день сервисы, предоставляющие возможность чтения статей, написанных разными авторами, становятся все более популярными. Такие приложения предоставляют пользователю возможность самостоятельно формировать набор информационных источников при помощи функции подписки на интересующую категорию или автора. Подобные веб-приложения часто выполняются в виде персонализируемого портала статей, ранжированных по дате публикации. Примером такого портала может служить веб-портал «Информационный портал для обмена публикациями», содержащего аналогичный информационный кон-

тент. Разработку данного портала будем использовать как модельный пример для описания этапов практикума, а в четвертом разделе приведены все разделы отчета по разработке веб-портала «Информационный портал для обмена публикациями».

3.2 Аналитический раздел

По итогам выполнения первого этапа практики необходимо оформить Аналитический раздел отчета по практике. В этом разделе необходимо представить обзор существующих методов и подходов по реализации поставленной задачи и провести анализ предметной области. На основе этого анализа представить Техническое задание на разработку РСОИ и Топологию системы с описанием функциональных требований к сервисам.

3.2.1 Техническое задание

Техническое задание необходимо оформить на основе ГОСТ 19.201-78 «ЕСПД. Техническое задание. Требования к содержанию и оформлению» [2] и методических рекомендаций [3].

Техническое задание должно включать описание следующих разделов:

- Глоссарий;
- Основания для разработки;
- Назначение разработки;
- Существующие аналоги;
- Описание системы:
 - основное назначение системы;
 - область ее использования;
- Общие требования к системе:
 - по модернизации и восстановлению системы;
 - по безопасности системы;
- Требования к функциональным характеристикам:
 - числовые значения функциональных характеристик (времени отклика);

- учет латентности;
- Функциональные требования к portalу с точки зрения пользователя:
 - возможные роли пользователей;
 - функции пользователей с учетом их роли;
 - входные и выходные данные системы.
- Требования к программной реализации

В этом разделе необходимо указать требования согласно п.1.3 задания по практике.
- Требования к надежности и к документации.

3.2.2 Топология системы

Топология разрабатываемой системы позволяет визуализировать связи между ее компонентами. Например, топология модельной системы «Информационного портал для обмена публикациями» изображена на рисунке 1 в разделе 4 данного пособия.

Модельная система состоит из фронт-энда и 6 подсистем (сервисов) :

- Сервис публикаций;
- Сервис подписок;
- Сервис аккаунтов;
- Сервис статистики.
- Сервис регистрации и авторизации;
- Сервис – координатор.

После представления топологии разрабатываемой РСОИ необходимо сформулировать:

- Общие требования к подсистемам;
- Функциональные требования к сервисам.

3.3 Конструкторский раздел

Конструкторский раздел должен содержать более детальное описание функционала приложения по сравнению с тем, что было отражено в техническом зада-

нии. Использование общепринятых схем и диаграмм, а также пояснений к ним позволяет наглядно отобразить не только внешнее взаимодействие пользователя с приложением, но и внутреннее устройство системы, раскрывая детали реализации на более низком уровне.

3.3.1 Концептуальный дизайн

Концептуальный дизайн системы обычно содержит наиболее общие схемы описания функционала приложения с точки зрения пользователей [3]. Одной из таких схем является *IDEF0-модель* и графические модели, входящие в нее [4]. Для разрабатываемой системы необходимо представить контекстную диаграмму верхнего уровня, которая обеспечивает наиболее общее или абстрактное описание работы системы. Данный вид диаграммы позволяет формализовать описание запросов пользователя и ответов системы на данные запросы, отобразив систему в виде “черного” ящика.

Для уточнения деталей работы системы необходимо применить декомпозицию функций, отображенных на диаграмме верхнего уровня, при помощи создания дочерних диаграмм.

Детализированная диаграмма, создаваемая при декомпозиции, охватывает ту же область, что и родительский блок, но описывает ее более подробно. Поэтому такая диаграмма может быть создана для любого из запросов, отображенных на диаграмме верхнего уровня. Каждый из блоков детализированной диаграммы может быть в свою очередь также описан при помощи дочерней диаграммы.

Проектирование концептуального дизайна должно включать описание:

- *сценариев функционирования* или использования системы (конкретную последовательность действий, иллюстрирующую поведение пользователя при работе с приложением);
- *диаграмм прецедентов* (графические сценарии функционирования системы) с указанием спецификаций сценариев.

3.3.2 Логический дизайн

В процессе создания концептуального дизайна системы были отражены ос-

новные сценарии взаимодействия пользователя и системы. В разделе логического дизайна необходимо представить организацию элементов системы и их взаимодействие между собой.

На основе функциональных требований к выделенным подсистемам, а также объектов, о которых необходимо хранить данные в системе, необходимо разработать схему данных приложения. Результат ее проектирования можно отобразить на условной **ER-диаграмме**, где овалами обозначены ключевые сущности, а ромбами - связи между ними [3]. Участие сущности в отношении с другой сущностью отмечается линией, соединяющей их. Число, располагающееся около линии, означает тип связи между соединенными сущностями.

На следующей стадии проектирования, добавив в схему данных атрибуты сущностей, необходимо получить **схему базы данных** с указанием всех спецификаций [3].

На основе разработанной схемы данных можно установить соответствие сущностей и сервисов и разработать диаграмму классов.

Для описания поведения компонентов системы на единой оси времени используются **диаграммы последовательности действий**, при помощи которых можно описать последовательность действий для каждого прецедента, необходимую для достижения цели [3].

Разрабатываемая система предполагает распределенное хранение данных. Поэтому необходимо разработать **диаграмму потоков данных**, которая будет отображать модель информационной системы с точки зрения хранения, передачи и обработки данных во время обработки запроса пользователя [3].

В качестве завершающей диаграммы раздела логического дизайна необходимо представить **схему архитектуры системы**, на которой изобразить протоколы взаимодействия компонентов системы.

3.4 Технологический раздел

Технологический раздел должен содержать детальное описание выбранного архитектурного стиля взаимодействия сервисов [5], используемых фреймворка и СУБД. А также описание выполнения требований к программной реализации

РСОИ (масштабируемость, отказоустойчивость и т.д.)

3.4.1 Выбор технологии для программной реализации

Архитектура СОА не опирается на какую-либо конкретную технологию и вместо этого предоставляет набор принципов построения и взаимодействия компонентов системы. Использование этого подхода даёт возможность разработчикам обеспечить новый уровень распределенности информационных систем. Главное преимущество такой архитектуры – универсальность. Такая концепция построения архитектуры позволяет более эффективно решать задачу интеграции приложений различной природы и построения распределенных гетерогенных ИС, поскольку может объединять информацию из любых операционных систем, использовать разные языки программирования, серверы приложений, различные модели организации данных. Программная реализация веб-сервисов может быть создана на любом языке программирования с использованием любой операционной системы и любого связующего программного обеспечения (middleware).

Основная идея микросервисного подхода состоит в том, чтобы разделить приложение на множество меньших по объему, но взаимосвязанных сервисов. Каждый микросервис представляет собой мини-приложение, состоящее из бизнес-логики и различных адаптеров. Некоторые микросервисы будут выставять API, которые потребляется другими микросервисами или клиентами приложения. Другие микросервисы могут реализовывать веб-интерфейс. Использование интерфейсов позволяет менять реализацию сервиса, не вызывая необходимости изменять обращения к нему во всей системе. Также это позволяет с легкостью горизонтально масштабировать систему путем добавления сервиса - балансировщика между вызывающим и вызываемым сервисами. Такой балансировщик отвечает интерфейсу того сервиса, нагрузку на который он балансирует, что делает его незаметным для вызывающей стороны, в то время как балансировщик распределяет получаемые запросы на ряд доступных ему сервисов. Проблемы могут возникнуть только при хранении состояния на сервисах, так как в этом случае необходимо дополнительно отслеживать какой из вызывающих сервисом как какому вы-

зываемому обращался.

Архитектурным стилем взаимодействия между сервисами может быть выбран **REST** (Representational State Transfer — «передача состояния представления»). REST – это архитектурный стиль взаимодействия компонентов распределённого приложения в сети. Он представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы. Данный стиль подразумевает отсутствие хранения состояния сервисами и передачу всей необходимой информации в теле запроса. REST опирается на протокол HTTP и использует существующие типы сообщений и их содержимое, например, HTTP GET для получения информации об объектах или HTTP POST для добавления информации. Также REST содержит рекомендации по именованию объектов и запросов, которые позволяют в дальнейшем облегчить сопровождение системы.

В качестве языка реализации серверной части приложения может быть выбран язык **C#, Python, JavaScript или PHP** так как они являются основными языками для веб-разработки на сегодняшний день, а также обладает всеми необходимыми особенностями для реализации СОА. Выбор конкретного языка программирования зависит от выбранного для разработки **фреймворка**. Фреймворк — это набор инструментов, библиотек и готовых модулей, которые веб-разработчик может использовать при создании сайтов.

Например,

.Net Core 2.0 – C#; ASP.NET MVC – C#; Django — Python;

Zend — PHP; Express.js — JavaScript;

Какой из фреймворков выбирать для разработки зависит от требований проекта.

Важный этап разработки РСОИ – это определение способа хранения данных РСОИ. Однако выбор конкретного фреймворка определяет и возможные СУБД, которые он поддерживает. Например, фреймворк .Net Core включает в себя фреймворк работы с реляционными базами данных Entity Framework Core. Данный фреймворк позволяет обращаться к широкому спектру реляционных баз данных при помощи унифицированного интерфейса. Благодаря этому возможно менять СУБД в любой момент при минимальных изменениях в системе.

3.4.2 Программная реализации PCOI

При написании программного кода для PCOI необходимо:

- **Реализовывать сервисную архитектуру системы.**

Для соответствия стилям COA и REST необходимо разбить систему на отдельные сервисы и организовать взаимодействие между ними через интерфейсы, без прямого обращения [1]. Каждый сервис создается как веб-приложение. Основным элементом сервисов являются классы, методы которых вызываются в ответ на запросы извне. Важно понимать, что объекты классов не хранятся в приложении, а создаются при поступлении запроса и удаляются после его обработки. Таким образом, каждый класс имеет изолированные от других переменные, что, с одной стороны, делает систему менее гибкой, но с другой позволяет избежать большого числа ошибок. Для взаимодействия между сервисами можно использовать протокол HTTP. Для общения между сервисами на главном (здесь и далее главным сервисом будет называться сервис, взаимодействующий с пользователем) сервисе необходимо создать классы – интерфейсы сервисов.

- **Разработать дружественный интерфейс с пользователем.**

Возможности организации интерфейса зависят от выбранного фреймворка. Например, В приложениях ASP.Net Core используется технология серверной шаблонизации, при которой страница генерируется на сервере в соответствии с текущими данными, а затем отправляется пользователю. В ответ на его действия генерируются новые страницы.

- **Реализовать возможность авторизации в системе (в том числе, через социальные сети).**

Для реализации регистрации и авторизации в приложении прежде всего необходимо создать сервис работы с хранилищем данных пользователя. В основе его реализации лежит модель, которая может содержать следующие поля: (Имя, Пароль, Роль). В классе сервиса необходимо реализовать методы работы с данными пользователя, например, сохранения информации в базу данных при регистрации или проверки данных пользователя при попытке авторизации и другие сопутствующие методы.

После реализации сервиса авторизации необходимо внедрить функционал регистрации и авторизации в главный сервис, во-первых, чтобы в системе мог работать только авторизованный пользователь, во-вторых, для проверки актуальности сессии клиента, работающего с приложением. В связи с тем, что проверка авторизации пользователя должна производиться перед выполнением любого запроса, требуется встроить данную проверку в процесс обработки запросов при помощи добавления компонентов промежуточного слоя.

На сегодняшний день многие пользователи имеют множество аккаунтов в различных приложениях, в связи с этим иногда гораздо удобнее авторизоваться в новом приложении при помощи уже существующего аккаунта другого сервиса, нежели заново регистрироваться. Возможность такой авторизации предоставляет фреймворк OAuth 2, который позволяет приложениям осуществлять ограниченный доступ к пользовательским аккаунтам на HTTP-сервисах, например, аккаунтам в популярных социальных сетях. Он работает по принципу делегирования аутентификации пользователя сервису, на котором находится аккаунт пользователя, позволяя стороннему приложению получать доступ к аккаунту пользователя. В четвертом разделе пособия подробно описана реализация авторизации пользователя через Google в разрабатываемом модельном приложении.

- **Реализовать основные функциональные требования к системе.**

На основе логического дизайна системы и с использованием возможностей выбранного фреймворка написать методы для реализации функциональных требований системы.

- **Обеспечить масштабируемость системы.**

Несмотря на то, что сервисная архитектура и стиль REST позволяют легко горизонтально масштабировать систему, для автоматизации процесса необходимо приложить усилия [6]. В разрабатываемой системе необходимо выделить отдельный сервис-координатор, отвечающий за балансировку нагрузки между сервисами. Принцип работы координатора прост: анализируя полученные от сервиса статистики данные о нагрузке на сервисы, а также время доступа одного сервиса к другому, координатор меняет связи между сервисами для равномерного распре-

деления нагрузки на систему. Использование координатора позволяет отказаться от записи необходимых адресов в конфигурационный файл каждого сервиса. Вместо этого при запуске координатора происходит инициализация всех известных координатору сервисов. В соответствии с заданными зависимостями каждому сервису отправляются необходимые для его работы адреса. При добавлении администратором новых сервисов они инициализируются, а периодически проходит балансировка на основании ранее описанных данных.

- **Обеспечивать отказоустойчивость системы.**

Отказоустойчивость гарантирует работоспособность системы, несмотря на внутренние неполадки в системе. Существуют разные способы сохранения функционирования, например, деградация функциональности, полный откат операции, очередь сообщений и другие [6].

Деградация функциональности - принцип сохранения работоспособности всего приложения при частичной потере функциональности. В разрабатываемом приложении отказоустойчивость распространяется на недоступность сервисов. То есть в том случае, когда один из сервисов не способен дать ответ на запрос из-за недоступности, необходимо вывести пользователю информацию, которую удалось обработать без участия недоступного сервиса, и сообщение об ошибке. Деградация функциональности не решает проблему в том случае, когда запрос влечет изменение данных. Потому что операция изменения данных является единой транзакцией, а значит, частичная работа над ними невозможна.

Откат операции означает, что метод, который агрегирует запрос пользователя в запросы к сервисам, перед внесением изменений должен сохранять снимок исходного состояния изменяемых данных. Это необходимо для того, чтобы в процессе выполнения запроса при возникновении ошибки система могла стереть только что внесенные изменения, вернуть исходное состояние и выдать пользователю ошибку о невозможности обработки запроса.

Метод обеспечения отказоустойчивости системы с кратким названием «*Очередь сообщений*» состоит в следующем. Если какой-то из сервисов недоступен в данный момент времени и невозможно выполнить запрос на изменение данных, то этот запрос ставится в очередь на вы-

полнение. Когда запрос выделяется из очереди, то происходит попытка его выполнения. Если попытка оказалась успешной, то задача считается выполненной, в противном случае этот запрос снова ставится в очередь.

- **Написать Руководство пользователя.**

Руководство пользователя должно включать следующие разделы, описывающие варианты работы пользователя в разработанной системе:

- авторизация и регистрация пользователя;
- описание вариантов работы в системе при авторизации через аккаунт, для любой роли пользователя.

4. ПРИМЕР ВЫПОЛНЕНИЯ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

Тема индивидуального задания: «Информационный портал для обмена публикациями».

Аналитический раздел

На сегодняшний день сервисы, предоставляющие контент для чтения в удобной форме, становятся все более популярными. Приложения, которые содержат такие сервисы, позволяют пользователю самостоятельно формировать набор новостных источников при помощи функции подписки на интересующий канал. Такие веб-приложения часто выполняются в виде персонализируемого портала статей, ранжированных по дате публикации. Техническое задание на создание РСОИ «Информационный портал для обмена публикациями» разработано в соответствии с ГОСТ РФ 19.201-78 [1].

Техническое задание

Глоссарий

- 1.Узел системы - региональный сервер, содержащий данные авторов и читателей указанного региона.
- 2.“Горячее” переконфигурирование системы - способность системы применять изменения без перезапуска и перекомпиляции.
- 3.Медиана времени отклика - среднее время предоставления данных Пользователю.
- 4.Латентность географического положения - увеличение времени отклика приложения, обуславливаемое географическим положением элементов системы или пользователя.
- 5.Статья - заголовок и соответствующий текст определенного автора.
- 6.Валидация - проверка данных на соответствие заданным условиям и ограничениям.
- 7.REST - архитектурный стиль взаимодействия компонентов распределённого приложения в сети.
- 8.Категория - общая тематика, позволяющая объединить статьи в одну группу.

9.Подписка - указание системе автоматически предоставлять статьи подписчику от авторов или из категорий, на которые производится подписка.

10.Новость – статья, содержащая имя автора, заголовок и тело статьи.

Основания для разработки

Разработка ведется в рамках проведения учебной практики по разработке РСОИ на кафедре «Программное обеспечение ЭВМ и информационные технологии» факультета «Информатика и системы управления» МГТУ им. Н. Э. Баумана.

Назначение разработки

Главное назначение разрабатываемого портала - предоставление пользователю возможности просмотра публикаций различных авторов, историю прочитанных статей, а также подборок популярных за месяц статей. Приложение должно обеспечивать пользователю, как автору, публикацию и хранение его статьи, а как читателю - удобство восприятия и ранжирования информации по авторам и интересам. Пользователь использует публикации для привлечения широкого круга читателей и просматривает портал для личных целей.

Существующие аналоги

Среди аналогов разрабатываемого проекта можно отметить такие порталы, как “Стаканчик”, “Идеономика” и “Лайфхакер”. Данный проект должен иметь следующие преимущества перед существующими аналогами:

- поиск авторов по категориям;
- предоставление доступа к истории прочитанных статей;
- создание подборки популярных статей за месяц.

Описание системы

Проект должен представлять собой портал для соединения пользователей, которые хотят делиться информацией друг с другом и самостоятельно формировать список источников информации для изучения. Пользователь может выступать в качестве автора, публикуя статьи и имея подписчиков. Каждая статья имеет счетчик просмотров, на основе данных счетчиков всех статей каждый месяц формируется топ-лист самых читаемых статей. Пользователь также формирует свою сферу интересов, подписываясь на категории и интересных авторов, и каждый раз полу-

чает персональную подборку статей.

Общие требования к системе

1. Система должна поддерживать возможность «горячего» переконфигурирования системы. Необходимо поддерживать возможность добавления нового узла во время работы системы без рестарта;
2. Время восстановления системы после сбоя не должно превышать 15 минут;
3. Каждый узел должен автоматически восстанавливаться после сбоя;
4. Обеспечить безопасность работоспособности системы за счет отказоустойчивости узлов;
5. Система должна автоматически выбирать наиболее подходящие серверы из доступных с целью минимизации латентности географического положения.

Требования к функциональным характеристикам

1. По результатам работы модуля сбора статистики медиана времени отклика системы на запросы пользователя на получение информации не должна превышать 3 секунд без учета латентности географического расположения узла;
2. По результатам работы модуля сбора статистики медиана времени отклика системы на запросы, добавляющие или изменяющие информацию на портале не должна превышать 5 секунд без учета латентности географического расположения узла;
3. Медиана времени отклика системы на действия пользователя должна быть менее 800мс при условии работы на рекомендованной аппаратной конфигурации, задержках между взаимодействующими сервисами менее 200мс и одновременном числе работающих пользователей менее 100 на каждый сервер, обслуживающий внешний интерфейс.

Функциональные требования к portalу с точки зрения пользователя

Portal должен обеспечивать реализацию следующих функций:

1. Система должна обеспечивать регистрацию и авторизацию пользователей с валидацией вводимых данных как через интерфейс приложения, так и через популярные социальные сети.

2. Система должна обеспечивать аутентификацию пользователей.
3. Система должна обеспечивать разделение пользователей на две роли:
 - Пользователь;
 - Администратор.
4. Система должна предоставлять **Пользователю** следующие функции:
 - добавление статьи;
 - удаление статьи;
 - получение списка всех статей;
 - изменение информации аккаунта;
 - управление подписками;
 - подробный просмотр статьи;
 - просмотр подборок статей.
5. Система должна предоставлять **Администратору** следующие функции:
 1. управление доступными категориями;
 2. функционал для настройки, удаления, добавления узлов;
 3. возможность просмотра статистики.

Входные данные Пользователя:

- ФИО, не более 256 символов;
- Статьи:
 - Заголовок;
 - Текст;
 - Категория.

Входные данные Администратора:

- ФИО, не более 256 символов.

Выходные данные

Выходными данными системы являются веб-страницы. В зависимости от запроса **Пользователя** они содержат:

- Список доступных для чтения статей;
- Список подписок пользователя;

- Информацию об аккаунте пользователя;
- Подробную информацию о статье;
- Информацию о статистике использования системы;
- И другие.

Требования к программной реализации

1. Требуется использовать СОА (сервис-ориентированную архитектуру) для реализации системы;
2. Система состоит из микросервисов. Каждый микросервис отвечает за свою область логики работы приложения;
3. Взаимодействие между сервисами осуществляется посредством HTTP-запросов и/или очереди сообщений;
4. Данные сервисов должны храниться в базе данных. Каждый сервис взаимодействует только со своей схемой данных. Взаимодействие сервисов происходит по технологии REST;
5. При недоступности систем портала должна осуществляться деградация функционала или выдача пользователю сообщения об ошибке;
6. Необходимо предусмотреть авторизацию пользователей, как через интерфейс приложения, так и через популярные социальные сети;
7. Для запросов, выполняющих обновление данных на нескольких узлах распределенной системы, в случае недоступности одной из систем, необходимо выполнять полный откат транзакции;
8. Необходимо сохранять профили всех пользователей в базе данных в хэшированном виде.
9. Приложение должно поддерживать возможность горизонтального и вертикального масштабирования за счет увеличения количества функционирующих узлов и совершенствования технологий реализации компонентов и всей архитектуры системы.

Топология системы

Топология разрабатываемой системы, изображена на рисунке 1.

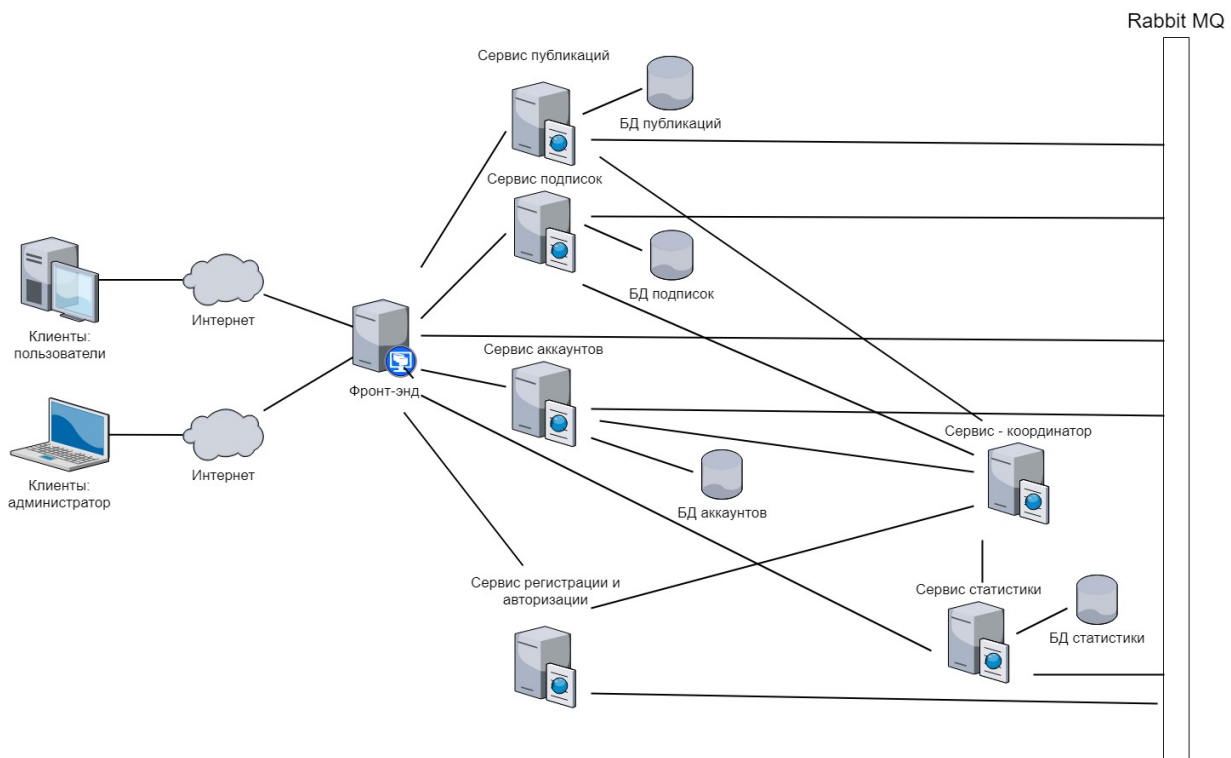


Рисунок 1. Топология системы.

Разрабатываемая система состоит из фронт-энда и 6 подсистем:

- Сервис публикаций;
- Сервис подписок;
- Сервис аккаунтов;
- Сервис статистики.
- Сервис регистрации и авторизации;
- Сервис – координатор;

Топология системы позволяет визуализировать связи между ее компонентами.

Общие требования к подсистемам

1. Фронт-энд – серверное приложение, при разработке которого следует учесть следующие нюансы:

- Фронт-энд должен принимать запросы по протоколу HTTP и формировать ответ пользователю в формате HTML-страниц;
- Фронт-энд является посредником между пользователями, передавая их запросы последовательно на сервис агрегации запросов;

- В соответствии с выбранными технологиями реализации целесообразно использовать библиотеку Bootstrap, язык HTML и Razor для создания верстки HTML-страниц проекта.
2. К реализации бэк-эндов должны быть предъявлены следующие требования:
- Прием и возврат данных должен происходить в формате JSON по протоколу HTTP;
 - Если результаты работы сервиса необходимо сохранять в базе данных, то доступ к ней должен осуществляться по протоколу HTTP. Доступ к базе данных может осуществляться только из подсистем, работающих напрямую с данными ее таблиц.

Функциональные требования к сервисам

1. **Сервис агрегации запросов** - предоставляет пользовательский интерфейс и внешний API системы.

Сервис должен реализовывать следующий функционал:

- Проверка существования пользователя;
- Регистрация пользователя;
- Удаление пользователя;
- Изменение имени пользователя;
- Получение новостей для пользователя (статей тех авторов и категорий, на которые он подписан);
- Добавление статьи пользователя;
- Получение списка авторов и категорий, на которых пользователь подписан;
- Добавление подписки;
- Удаление подписки;
- Осуществление аутентификации пользователя;
- Получение списка доступных узлов системы (только для администратора);
- Добавление и удаление узлов системы (только для администратора);
- Конфигурация узлов системы (только для администратора);
- Добавление и удаление категории (только для администратора).

2. Сервис работы с публикациями - отвечает за добавление статей и хранение информации о них.

Хранимая в базе данных сущность, ассоциированная с сервисом, имеет следующие обязательные поля:

- Заголовок публикации;
- Тело публикации;
- Автор;
- Дата добавления;
- Категория;
- Счетчик просмотров.

Сервис должен реализовывать следующий функционал:

- Добавление автором статьи (при этом необходимо обеспечить проверку существования у данного автора статьи с таким названием);
- Группировка в соответствии с выбранным количеством статей для отображения на странице;
- Удаление всех статей автора;
- Изменение идентификатора автора во всех его статьях;
- Получение всех статей автора в категории;
- Получение всех статей в категории;
- Увеличение счетчика просмотров статьи при подробном просмотре статьи;
- Формирование топ-листа публикаций за месяц;
- Формирование топ-листа публикаций популярных авторов за месяц;
- Получение всех авторов, имеющих статьи в указанной категории.

3. Сервис работы с подписками - осуществляет формирование персонального информационного окружения пользователя.

Хранимая в базе данных сущность, ассоциированная с сервисом, имеет следующие обязательные поля:

- Идентификатор пользователя;
- Идентификатор автора или категории, на который подписывается читатель.

Сервис должен реализовывать следующий функционал:

- Получение идентификаторов авторов, на которых подписан пользователь;
- Добавление подписки;
- Удаление подписки;
- Получение всех подписок пользователя;
- Удаление подписок пользователя;
- Изменение идентификатора пользователя во всех ассоциированных с ним подписках.

4. Сервис работы с аккаунтами пользователей

Хранимая в базе данных сущность, ассоциированная с сервисом, имеет следующие обязательные поля:

- Идентификатор пользователя;
- Пароль.

Сервис должен реализовывать следующий функционал:

- Проверка существования пользователя;
- Аутентификация пользователя;
- Регистрация нового пользователя;
- Удаление пользователя;
- Изменение имени пользователя.

5. Сервис сбора статистики

Сервис должен реализовывать следующий функционал:

- Получение статистики добавления статей за последнюю неделю;
- Получение статистики операций за выбранный промежуток времени.

6. Сервис регистрации и авторизации пользователей

Сервис должен реализовывать следующий функционал:

- Возможность регистрации нового аккаунта в системе;
- Возможность авторизации пользователя как через аккаунт в системе, так и через аккаунт предлагаемых социальных сетей.

Требования к надежности и к документации

Система должна работать в соответствии с данным техническим заданием без перезапуска. Необходимо использовать «зеркалируемые серверы» для всех подсистем, которые будут держать нагрузку в случае сбоя до тех пор, пока основной сервер не восстановится.

Исполнитель должен подготовить и передать Заказчику следующие документы:

- руководство администратора Системы;
- руководство для пользователя по использованию Системы;

Конструкторский раздел

Концептуальный дизайн

Концептуальный дизайн системы содержит наиболее общие схемы описания функционала приложения с точки зрения пользователей [2]. Одной из таких схем является IDEF0-модель и графические модели, входящие в нее [4]. На рисунке 2 отображена контекстная диаграмма верхнего уровня, которая обеспечивает наиболее общее или абстрактное описание работы системы. Данный вид диаграммы позволяет формализовать описание запросов пользователя и ответов системы на данные запросы, отобразив систему в виде “черного” ящика.

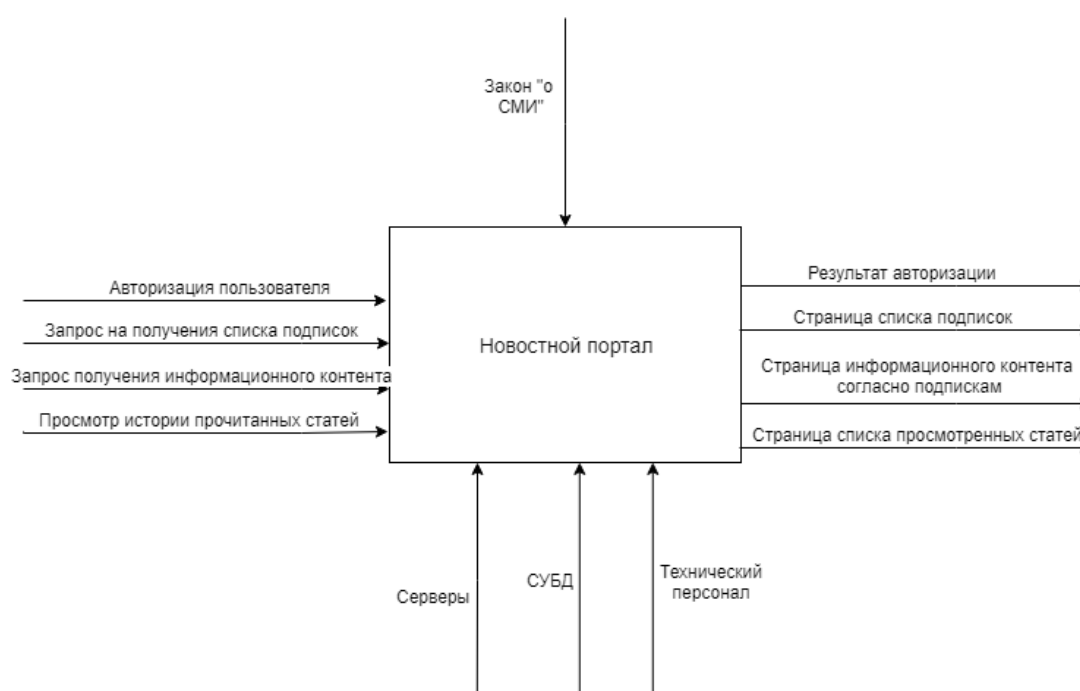


Рисунок 2. Концептуальная модель системы в нотации IDEF0.

Для уточнения деталей работы системы применяется декомпозиция функций, отображенных на диаграмме верхнего уровня, при помощи создания дочерних диаграмм. В качестве примера на рисунке 3 изображена дочерняя диаграмма, которая определяет последовательность выполнения операций в системе при обработке запроса пользователя на получение информационного контента.

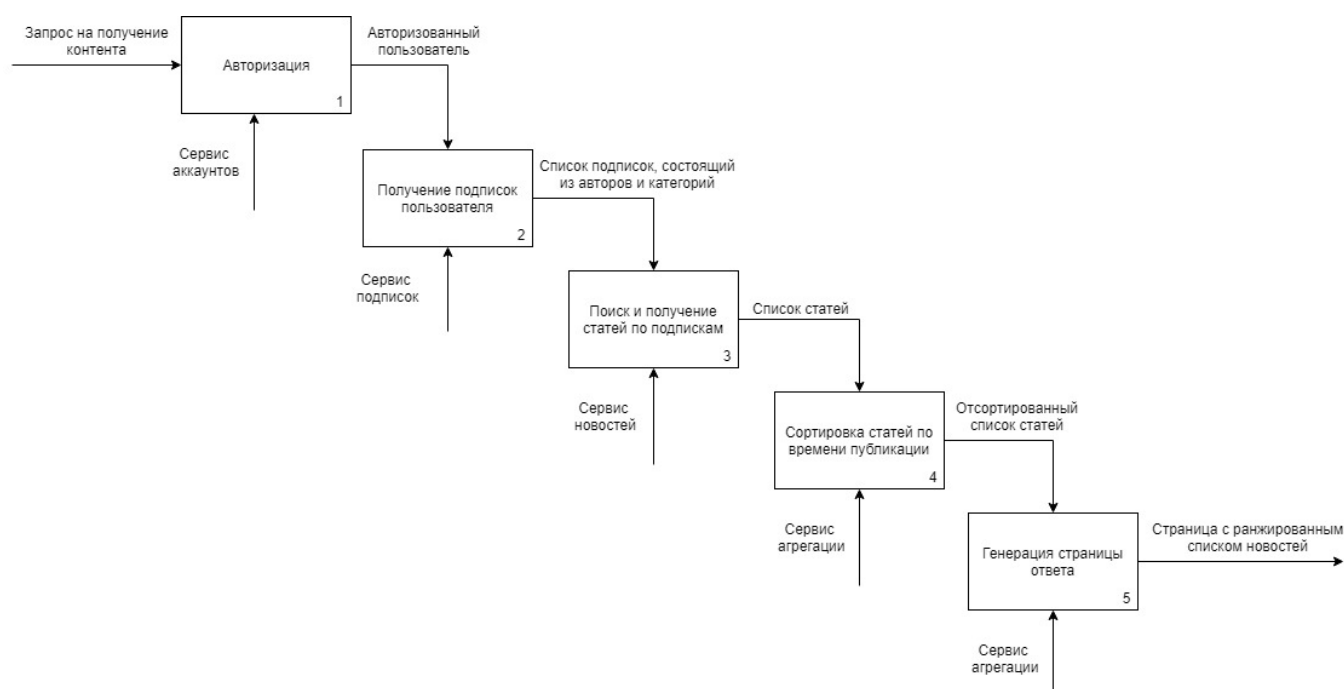


Рисунок 3. Детализированная концептуальная модель системы в нотации IDEF0.

Детализированная диаграмма, создаваемая при декомпозиции, охватывает ту же область, что и родительский блок, но описывает ее более подробно. Поэтому такая диаграмма может быть создана для любого из запросов, отображенных на диаграмме верхнего уровня. Каждый из блоков детализированной диаграммы может быть в свою очередь также описан при помощи дочерней диаграммы.

Сценарии функционирования системы

Сценарии функционирования или использования системы описывают конкретную последовательность действий, иллюстрирующую поведение пользователя при работе с приложением. Далее приведены подробные сценарии основных

возможных действий пользователя.

Регистрация пользователя:

1. Пользователь нажимает на кнопку «Войти» в интерфейсе приложения;
2. Пользователь перенаправляется на страницу авторизации, которая содержит поля для заполнения его данных;
3. Пользователь вводит данные в форму и для завершения регистрации нажимает на кнопку «Регистрация», тем самым подтверждая верность своих данных, а также согласие на их обработку и хранение.
4. Если пользователь с введенным для регистрации именем уже существует, то клиент перенаправляется на страницу ошибки. При успешной регистрации пользователь перенаправляется на страницу своего профиля в системе.

Авторизация пользователя:

1. Пользователь нажимает на кнопку «Войти» в интерфейсе приложения;
2. Пользователь перенаправляется на страницу авторизации, которая содержит поля для заполнения его логина и пароля;
3. Пользователь завершает работу с формой авторизации нажатием кнопки «Войти»;
4. При обнаружении ошибки в данных, пользователь перенаправляется на страницу ошибки; при совпадении данных с записью в базе данных аккаунтов пользователь получает доступ к системе.

Получение списка подписок:

1. Авторизованный пользователь нажимает на кнопку «Подписки»;
2. Пользователь перенаправляется на страницу, содержащую список пользователей и категорий, на которые он подписан.

Просмотр ранжированного списка новостей на основе подписок пользователя:

1. Авторизованный пользователь нажимает на кнопку «Новости».
2. Пользователь перенаправляется на страницу, которая содержит список новостей, ранжированных по дате публикации.

Получение списка просмотренных статей:

1. Авторизованный пользователь нажимает на кнопку «История просмотров».
2. Пользователь перенаправляется на страницу, содержащую список просмотренных пользователем статей, ранжированных по дате просмотра.

Получение статистики:

1. Пользователь с ролью администратор нажимает на кнопку “Панель администратора”.
2. Пользователь перенаправляется на страницу просмотра статистики среднего времени обработки запроса сервисами.

Диаграммы прецедентов

Графически сценарии функционирования системы можно представить при помощи диаграмм прецедентов. Они позволяют схематично отобразить типичные сценарии взаимодействия между клиентами и приложением. В системе выделены 2 основных роли: пользователь и администратор, диаграммы прецедентов для этих ролей изображены на рисунках 4 и 5.

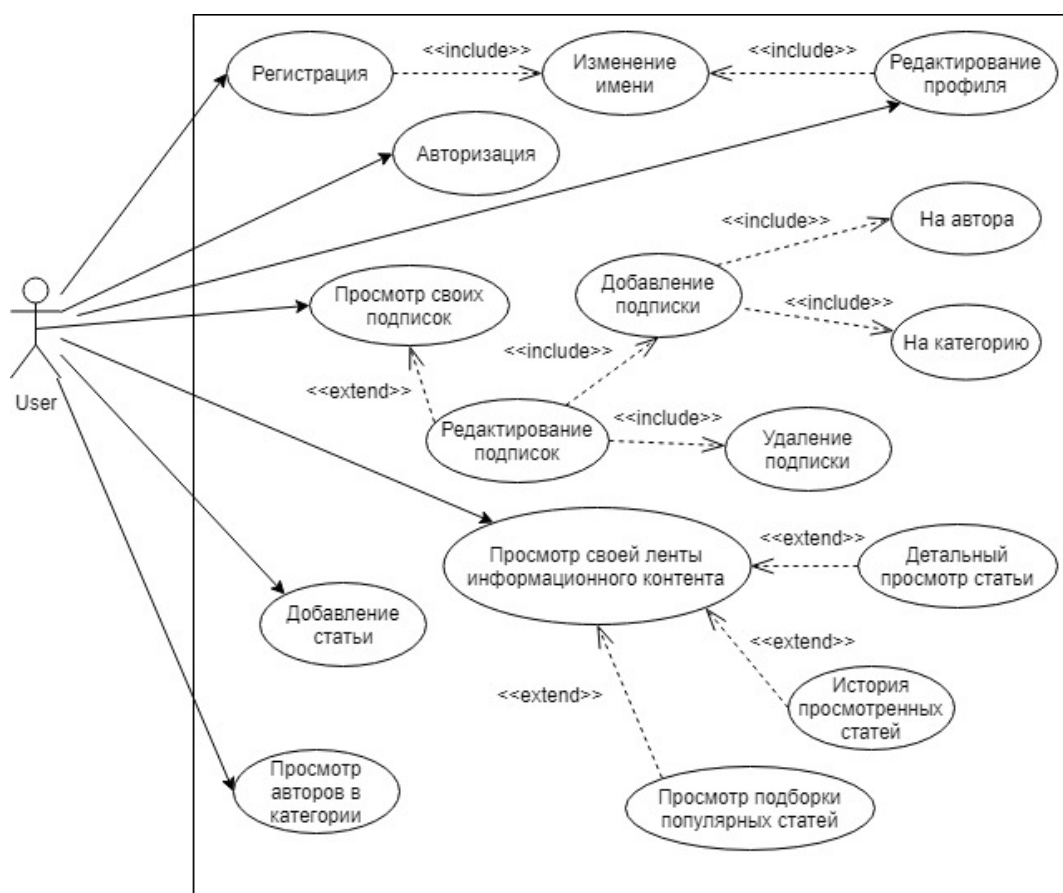


Рисунок 4. Диаграмма прецедентов с точки зрения пользователя.

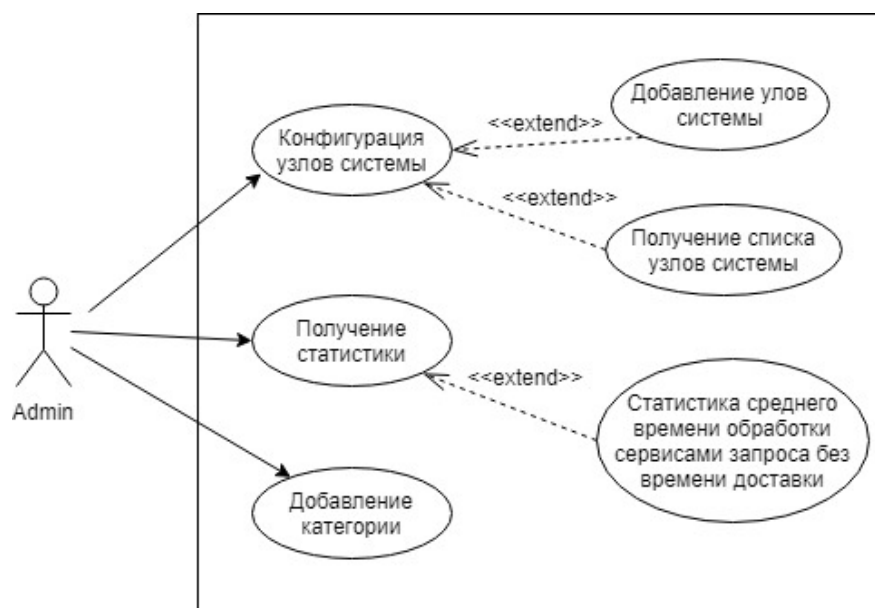


Рисунок 5. Диаграмма прецедентов с точки зрения администратора.

Спецификации сценариев

Приведенные сценарии могут иметь как основной поток выполнения, который выполняется чаще всего, так и альтернативные потоки, описывающие выполнение запроса при отклонении от основного хода сценария. Все возможные ходы выполнения сценария описываются при помощи спецификаций. Примеры спецификаций для описанных выше сценариев приведены в данном разделе.

Спецификация сценария «Регистрация»

Нормальный ход сценария	
Действие пользователя	Отклик системы
Пользователь нажимает кнопку “Войти”	Открывается страница для ввода данных
Пользователь вводит данные в поля и нажимает кнопку “Регистрация”	Открывается страница с профилем созданного пользователя

Альтернативный ход сценария	
Действие пользователя	Отклик системы
Пользователь нажимает кнопку “Войти”	Открывается страница для ввода данных
Пользователь вводит данные в поля и нажимает кнопку “Регистрация”	Открывается страница с сообщением об ошибке, что пользователь с такими данными существует

Спецификация сценария «Авторизация»

Нормальный ход сценария	
Действие пользователя	Отклик системы
Пользователь нажимает кнопку “Войти”	Открывается страница для ввода данных
Пользователь вводит данные в поля и нажимает кнопку “Войти”	Открывается страница профиля пользователя

Альтернативный ход сценария	
Действие пользователя	Отклик системы
Пользователь нажимает кнопку “Войти”	Открывается страница для ввода данных авторизации
Пользователь вводит данные в поля и нажимает кнопку “Войти”	Открывается страница ошибки с сообщением о неверно введенных данных

Спецификация сценария «Просмотр своих подписок»

Нормальный ход сценария

Действие пользователя	Отклик системы
Пользователь нажимает кнопку “Подписки”	Открывается страница со списком пользователей и категорий, на который он подписан

Альтернативный ход сценария	
Действие пользователя	Отклик системы
Пользователь нажимает кнопку “Подписки”	Открывается страница с сообщением об отсутствии подписок

Альтернативный ход сценария	
Действие пользователя	Отклик системы
Пользователь нажимает кнопку “Подписки”	Открывается страница с сообщением, что сервис подписок недоступен.

Спецификация сценария «Просмотр своей ленты информационного контента»

Нормальный ход сценария	
Действие пользователя	Отклик системы
Пользователь нажимает кнопку “Новости”	Открывается страница со списком новостей

Альтернативный ход сценария	
Действие пользователя	Отклик системы
Пользователь нажимает кнопку “Новости”	Открывается страница с сообщением, что у пользователя нет подписок, по которым можно смотреть новости

Альтернативный ход сценария	
Действие пользователя	Отклик системы
Пользователь нажимает кнопку “Новости”	Открывается страница с ошибкой, что по его подпискам нет новостей

Альтернативный ход сценария	
Действие пользователя	Отклик системы
Пользователь нажимает кнопку “Новости”	Открывается страница ошибки с сообщением, что сервис новостей недоступен

Спецификация сценария «История просмотренных статей»

Нормальный ход сценария	
Действие пользователя	Отклик системы
Пользователь нажимает кнопку “История просмотров”	Открывается страница со списком просмотренных статей

Альтернативный ход сценария	
Действие пользователя	Отклик системы
Пользователь нажимает кнопку “История просмотров”	Открывается страница с сообщением, что история пуста

Спецификация сценария «Получение статистики»

Нормальный ход сценария	
Действие пользователя	Отклик системы
Пользователь авторизуется как администратор	Открывается страница с профилем пользователя
Пользователь нажимает кнопку “Панель администратора”	Открывается страница, содержащая статистику среднего времени обработки запроса сервисами

Логический дизайн

В процессе создания концептуального дизайна системы были отражены основные сценарии взаимодействия пользователя и системы. В разделе логического дизайна представлена организация элементов системы и их взаимодействие между собой.

На основе функциональных требований к выделенным подсистемам, а также объектов, о которых необходимо хранить данные в системе, была разработана схема данных приложения. Результат ее проектирования отображен на условной ER-диаграмме, представленной на рисунке 6. На данной схеме овалами обозначены ключевые сущности, а ромбами - связи между ними. Участие сущности в отношении с другой сущностью отмечается линией, соединяющей их. Число, располагающееся около линии, означает тип связи между соединенными сущностями.

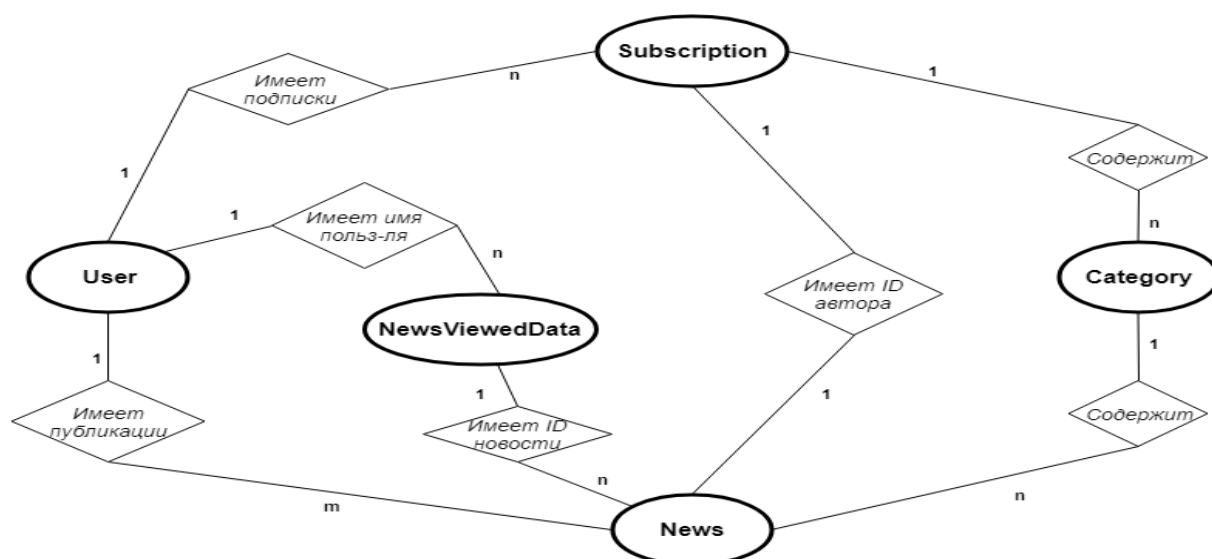


Рисунок 6. ER-диаграмма данных приложения.

На следующей стадии проектирования, добавив в схему данных атрибуты сущностей, получаем схему базы данных, которая изображена на рисунке 7.

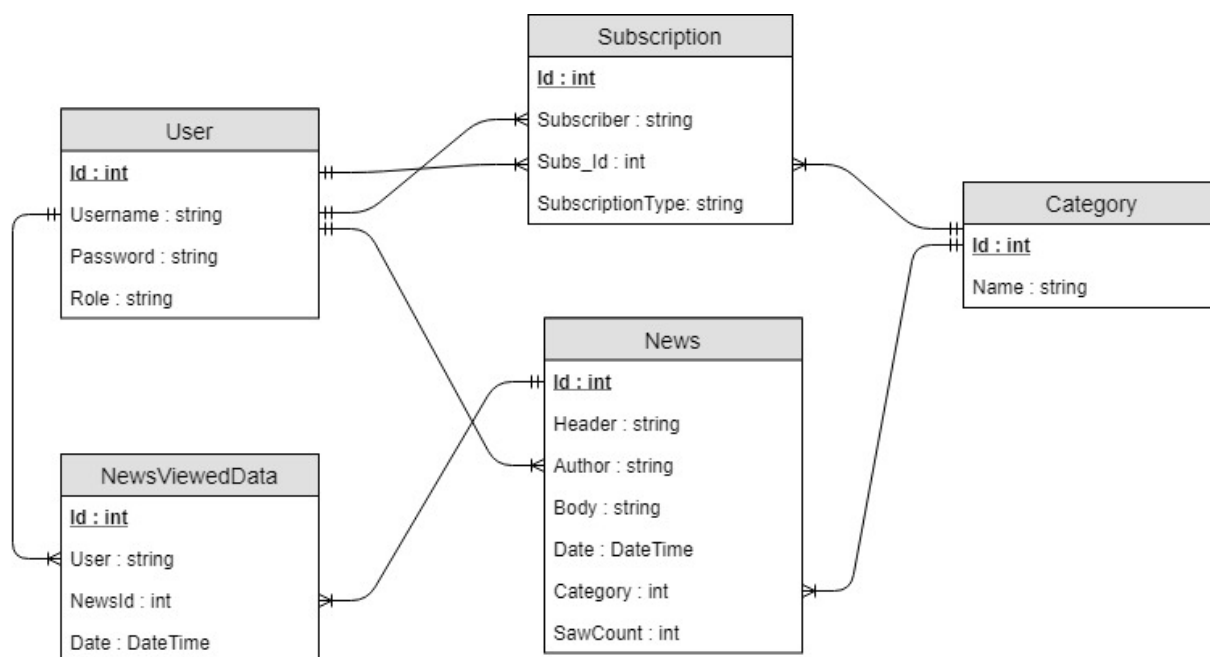


Рисунок 7. Схема базы данных системы.

Далее приводятся спецификации таблиц базы данных, приведенных на рисунке 7.

Спецификация таблицы User

Таблица User содержит данные о профиле пользователя.

Имя атрибута	Тип атрибута	Описание атрибута
Id	Public : int	Идентификатор публикации
UserName	Public : string	Имя пользователя
Password	Public : string	Пароль пользователя
Role	Public : string	Роль (клиент или администратор)

Спецификация таблицы Category

Таблица Category предназначена для работы с категориями публикаций.

Имя атрибута	Тип атрибута	Описание атрибута
Id	Public : int	Идентификатор категории
Name	Public : string	Название категории

Спецификация таблицы News

Таблица News является основной для сервиса публикаций и содержит в себе всю необходимую информацию о новостях.

Имя атрибута	Тип атрибута	Описание атрибута
Id	Public : int	Идентификатор статьи
Header	Public : string	Заголовок статьи
Body	Public : string	Текст статьи
Author	Public : string	Имя автора статьи
Date	Public : DateTime	Дата публикации
Category	Public : int	Идентификатор категории статьи
ViewCount	Public : int	Счетчик детального просмотра статьи

Спецификация таблицы NewsViewedData

Таблица NewsViewedData предназначена для работы с историей просмотров.

Имя атрибута	Тип атрибута	Описание атрибута
Id	Public : int	Идентификатор просмотра
NewsId	Public : int	Название категории
User	Public : string	Имя пользователя, просмотревшего публикацию

DateTime	Public : DateTime	Дата и время просмотра
----------	-------------------	------------------------

Спецификация таблицы Subscription

Таблица Subscription выделена для работы с подписками пользователей.

Имя атрибута	Тип атрибута	Описание атрибута
Id	Public : int	Идентификатор подписки
Subscriber	Public : string	Подписчик
Subs_Id	Public : int	Идентификатор объекта подписки (автор или категория)
SubscriptionType	Public : string	Тип подписки: на категорию или автора

Структура сервиса

На основе разработанной схемы данных можно установить соответствие сущностей и сервисов, описанных в предыдущем разделе. Например, для работы с сущностью News был выделен сервис публикаций. Рассмотрим его устройство более подробно. Так как приложение построено на основе модели MVC, каждый сервис имеет хотя бы один контроллер и хотя бы одну модель данных. Для связи модели и базы данных создается класс ApplicationDbContext, который наследует функционал от системного класса DbContext от .NET Core. Данный класс позволяет получать из хранилища набор данных для реализации дальнейших действий над ним. В качестве примера на рисунке 8 представлена диаграмма классов для разработки сервиса публикаций.

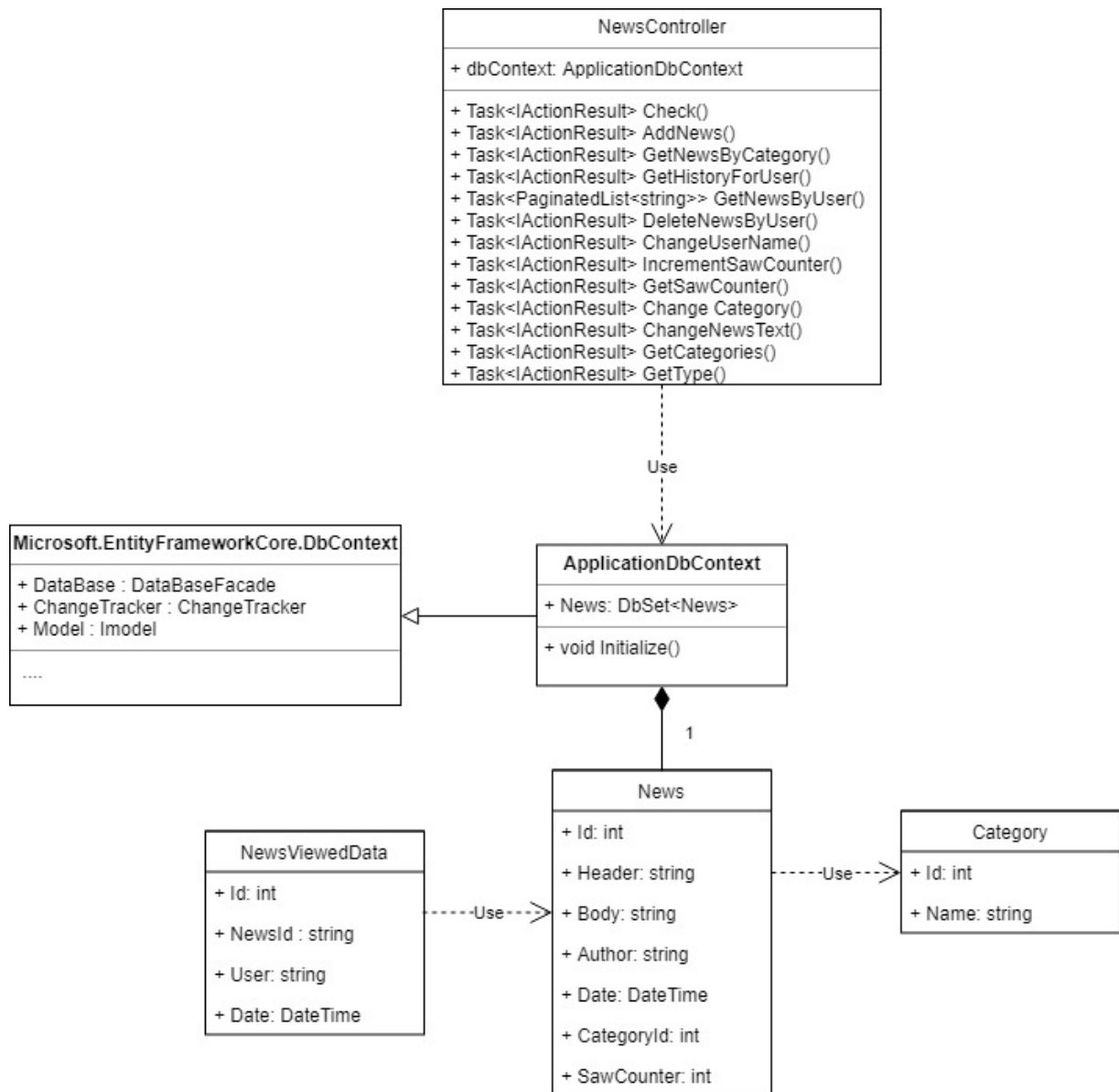


Рисунок 8. Диаграмма классов сервиса публикаций.

Функциональные требования, предъявляемые к сервису публикаций, реализуются при помощи методов контроллера NewsController. Далее приведено описание каждого метода данного контроллера.

Спецификация класса NewsController

Метод	Описание
Check	Проверка работоспособности сервиса новостей
AddNews	Добавление новости в базу

GetNewsByUser	Получение публикаций указанного пользователя
GetNewsByCategory	Получение новостей по указанной категории
GetHistoryForUser	Получить историю просмотренных статей для указанного пользователя
DeleteNewsByUser	Удалить публикации указанного автора
ChangeUserName	Изменить имя автора
IncrementSawCounter	Увеличить счетчик просмотров для указанной новости
GetSawCounter	Получить счетчик просмотров указанной публикации
GetCategories	Получить список категорий
ChangeCategory	Изменить категорию публикации
ChangeNewsText	Изменить текст публикации
GetType	Проверить, является ли указанная строка названием категории

Диаграммы последовательности действий

Для описания поведения компонентов системы на единой оси времени используются диаграммы последовательности действий, при помощи которых можно описать последовательность действий для каждого прецедента, необходимую для достижения цели. Например, на рисунке 9 изображен процесс получения списка новостей пользователя на основе списка его подписок.

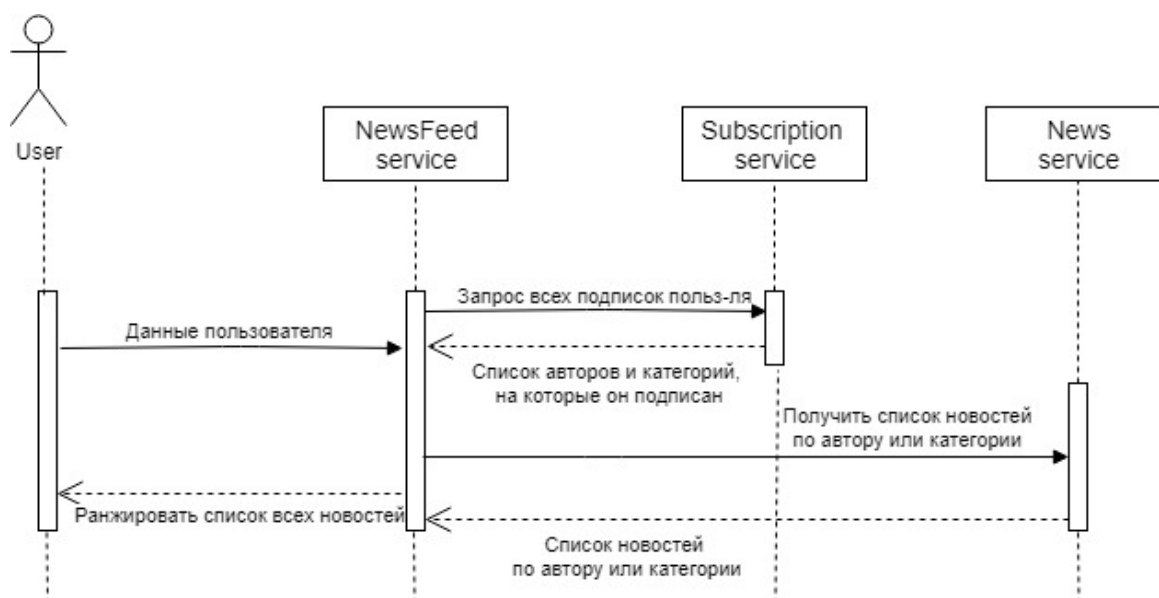


Рисунок 9. Диаграмма последовательности действий при запросе пользователем всех новостей на основе его подписок.

Главный сервис приложения отправляет запрос на получение списка авторов и категорий, на которые подписан пользователь. После получения данных главный сервис обращается к сервису новостей, чтобы для каждого автора или категории получить список новостей, которые были опубликованы. После окончания данной процедуры главный сервис ранжирует новости по времени публикации, формирует веб-страницу и возвращает ее пользователю.

Диаграмма потоков данных

Рассматриваемая система предполагает распределенное хранение данных. Все данные системы предполагают хранение в единой базе данных, хранилищами данных являются таблицы. Диаграмма потоков данных, представленная на рисунке 10, отображает модель информационной системы с точки зрения хранения, передачи и обработки данных во время обработки запроса пользователя на получение новостей.

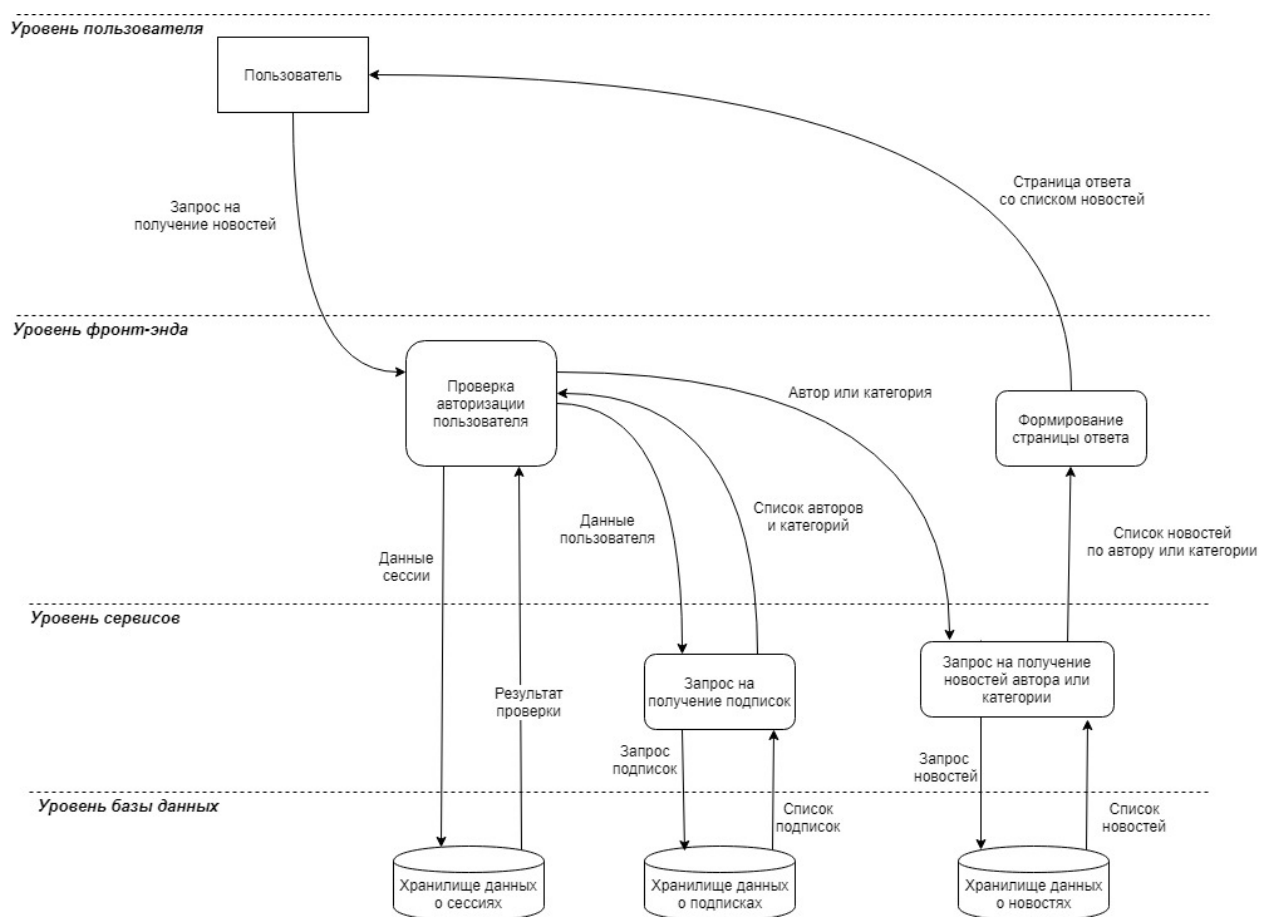


Рисунок 10. Диаграмма потоков данных при запросе пользователем всех новостей согласно подпискам.

Архитектура системы

Основополагающей идеей построения программной архитектуры является идея снижения сложности системы путём абстракции и разграничения полномочий. В данном проекте каждая функциональная область реализована посредством собственного микросервиса. Этот подход позволяет бороться со сложностью современных систем. Архитектура системы призвана показать способ развертывания системы во внешних средах. На рисунке 11 представлена архитектура системы, которая показывает размещение элементов системы на физических носителях и способах их взаимодействия, то есть, указаны протоколы, по которым происходит информационный обмен.

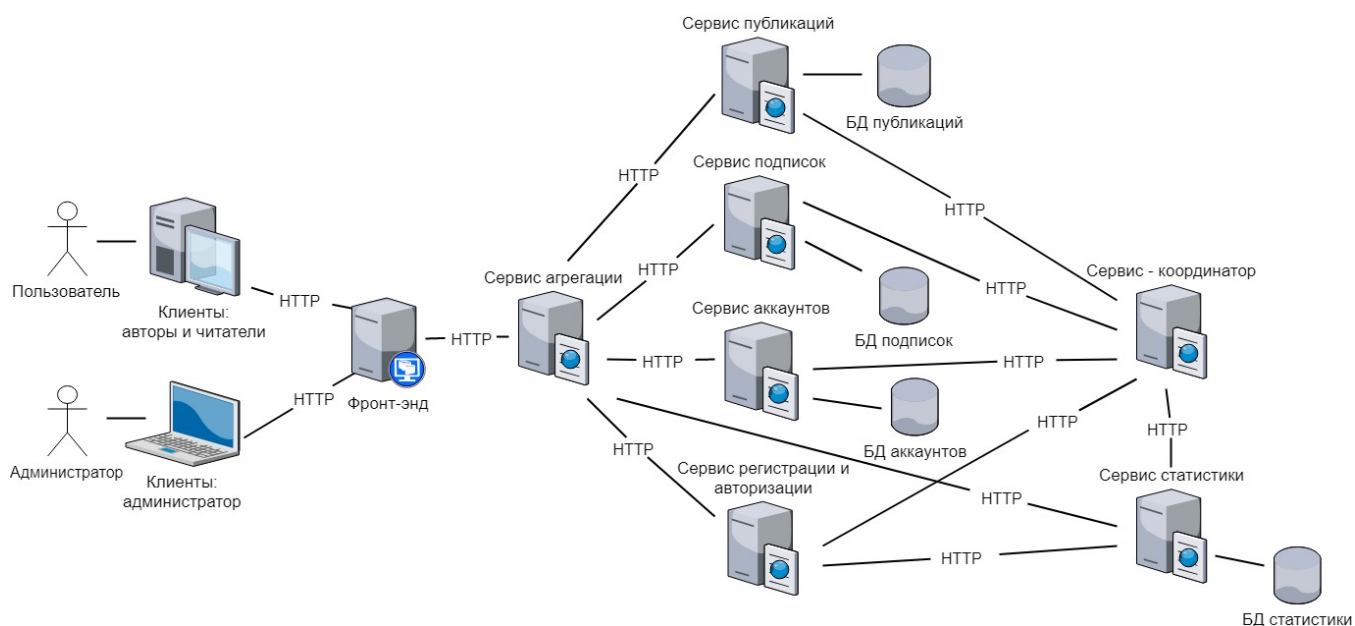


Рисунок 11. Архитектура разрабатываемой системы.

Технологический раздел

Выбор языка и фреймворка для разработки системы

Для реализации системы использовалась сервис-ориентированная архитектура (COA) [5]. Архитектурным стилем взаимодействия между сервисами выбран REST (Representational State Transfer — «передача состояния представления») [6].

В качестве языка реализации серверной части приложения выбран C#, так как он является одним из ведущих языков в разработке информационных систем на сегодняшний день, а также обладает всеми необходимыми особенностями для реализации COA и REST.

Для системы выбран фреймворк .Net Core 2.0 [7]. Он пришел на замену .Net Framework, и, в отличие от последнего, обладает возможностью запуска на разных платформах, использует только необходимые библиотеки и быстро развивается.

В рамках фреймворка .Net Core 2.0 доступна технология ASP (Active Server Pages — «активные серверные страницы»). Она позволяет динамически создавать страницы на стороне сервера и отправлять их пользователю. Такой способ создания веб-страниц для работы с пользователем называется серверной шаблонизацией или «server-rendering». Серверная шаблонизация выполняется на одном из сер-

висов системы, который работает с пользователем, обращаясь при необходимости к другим сервисам.

Сама технология ASP содержит в себе фреймворк ASP.NET MVC (Model-View-Controller, «Модель-Представление-Контроллер») [8]. Он, как понятно из названия, основан на архитектуре приложения MVC, и содержит три основных компонента:

- Модель – данные и работа с ними;
- Представление – отображение данных модели;
- Контроллер – реакция на действия пользователя и оповещение модели о необходимости изменений.

Данная архитектура позволяет четко разделять ответственность между компонентами, отделять бизнес-логику от её визуализации. Например, для добавления поддержки мобильных устройств достаточно добавить только соответствующее представление, не меняя модель и контроллер.

Выбор СУБД

Фреймворк .Net Core включает в себя фреймворк работы с реляционными базами данных Entity Framework Core [9]. Данный фреймворк позволяет обращаться к широкому спектру реляционных баз данных при помощи унифицированного интерфейса. Благодаря этому возможно менять СУБД в любой момент при минимальных изменениях в системе. Так как, согласно ТЗ, в базе данных не требуется хранить сложные объекты, например, файлы, для проекта подходит реляционная база данных. Поэтому для хранения данных была выбрана СУБД Microsoft SQL, как оптимальный вариант в связи с широким знакомством с ней в рамках курса «Базы данных».

Обеспечение масштабируемости

Как было отмечено выше, COA позволяет масштабировать систему горизонтально с использованием сервисов-балансировщиков. Совместное использование COA и REST-стиля, запрещающего сервисам хранить состояние, предостав-

ляет возможность с легкостью добавлять и удалять сервисы, динамически распределяя нагрузку между существующими. Однако можно воспользоваться глобальным сервисом-координатором. Технология ASP позволяет выстраивать конвейер по обработке каждого запроса к сервису [8]. Добавив в этот конвейер этап для сбора статистики запросов, можно отслеживать нагрузку, не прибегая к добавлению сервиса-балансировщика. На основании полученной информации, сервис-координатор будет определять нагрузку на каждый из сервисов и, основываясь на этой информации, динамически определять связи между сервисами с целью минимизации задержек и снижения нагрузки. Пример работы координатора показан на Рис.12.

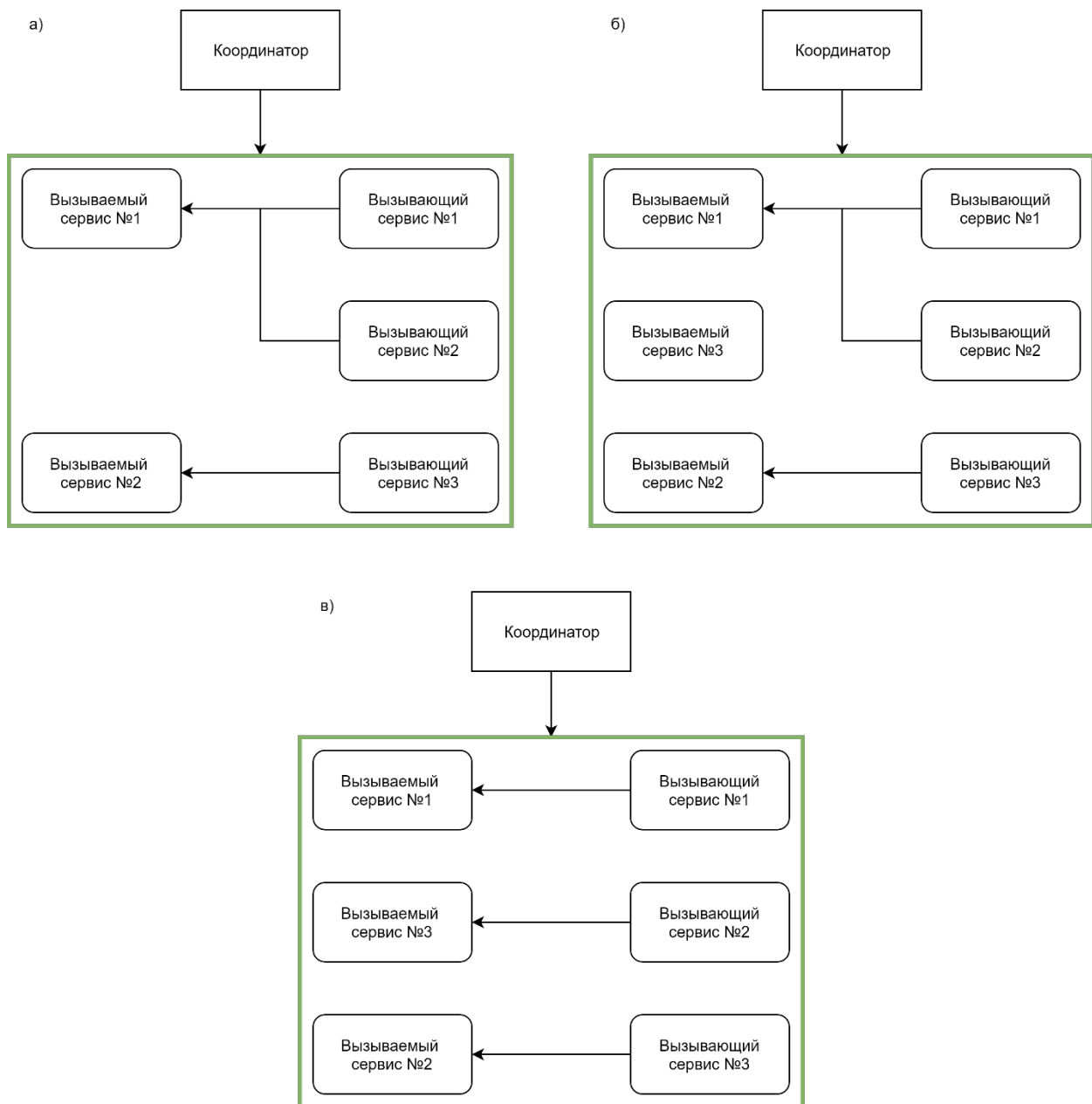


Рис. 12. Пример работы координатора. а) начальное состояние системы, состоящей из 3 однотипных вызывающих сервисов и 2 вызываемых; б) добавление нового вызываемого сервиса; в) балансировка нагрузки координатором путем переноса запросов вызывающего сервиса №2 на вызываемый сервис №3.

Реализация сервисной архитектуры

Общая архитектура сервиса

Для соответствия стилям COA и REST необходимо разбить систему на отдельные сервисы и организовать взаимодействие между ними через интерфейсы,

без прямого обращения. Каждый сервис создается как веб-приложение ASP.Net Core. Данный шаблон проекта имеет несколько дочерних шаблонов, включающих в себя веб-API и веб-приложение. Основным отличием между этими двумя типами являются создаваемые по умолчанию файлы: в веб-API присутствует минимальный набор файлов, ориентированный на прием и отправку HTTP-запросов, тогда как веб-приложение сразу обладает возможностями для создания веб-страниц путем серверной шаблонизации (которые можно добавить и в веб-API). Для всех сервисов, не взаимодействующих напрямую с пользователем, достаточно веб-API, а для взаимодействующих – веб-приложения.

Основным элементом сервисов являются контроллеры – классы, методы которых вызываются в ответ на запросы извне. Важно понимать, что объекты контроллеров не хранятся в приложении, а создаются при поступлении запроса и удаляются после его обработки. Таким образом, каждый контроллер имеет изолированные от других переменные, что, с одной стороны, делает систему менее гибкой, но с другой позволяет избежать большого числа ошибок. Контроллеры не создаются напрямую, за этим следит основная система веб-приложения совместно с системой внедрения зависимостей. Пример простого контроллера приведен в листинге 1.

```
[Route("home")]
public class HomeController : Controller
{
    private DbContext dbContext;
    public HomeController(DbContext dbContext)
    {
        this.dbContext = dbContext;
    }

    [HttpGet("about")]
    public IActionResult About()
    {
        return StatusCode(200, dbContext.TestCollection().First());
    }
}
```

Листинг 1 – Простой контроллер.

В листинге 1 показан простейший контроллер под названием Home (часть имени класса «Controller» принято опускать), получающий в конструкторе объект контекста базы данных (об этом далее). Единственным методом данного контроллера является метод About(), который будет вызван при обращении по адресу “/home/about”, который складывается из адреса контроллера, указанного в атрибуте Route и адреса метода, указанного в атрибуте HttpGet. Вторым атрибутом также определяет метод HTTP, который должен быть использован для обращения. В ответ на данный запрос будет получен код 200 (Ok в словаре кодов HTTP) и первый элемент коллекции объектов TestCollection.

Все приложения ASP.Net Core сразу снабжаются системой DI – Dependency Injection, внедрение зависимостей. Она отвечает за передачу нужных аргументов во все конструкторы классов, которые ей известны, а также за их инициализацию. Конфигурация данной системы происходит в файле Startup.cs. В нем, в методе ConfigureServices(...) происходит подключение допустимых классов к общей системе DI. По умолчанию, к ней также подключаются все контроллеры. Поэтому для получения объектов достаточно просто указать их как аргументы конструктора контроллера, как например, DbContext. Пример простой конфигурации DI приведен в листинге 2.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddDbContext<DbContext>();
    services.AddTransient<TransientService>();
    services.AddSingleton<SingletonService>();
}
```

Листинг 2 – Простая конфигурация DI.

Каждый подключаемый к системе DI сервис может быть объявлен либо как временный (Transient), либо как постоянный (Singleton). Временные сервисы создаются каждый раз, когда в них возникает необходимость. Примером временных

сервисов служат контроллеры. Постоянные сервисы остаются в памяти между запросами на их получение. Такие сервисы используются для хранения больших объемов данных, повторное создание которых является дорогостоящей операцией.

Базы данных, поддерживаемые Entity Framework, в свою очередь, особым образом подключаются к системе DI при помощи функции `AddDbContext<>()`. Контекст базы данных является временным сервисом, имеющим ограниченную область видимости. Это важное свойство, так как оно не позволяет получать контекст базы данных в постоянные сервисы. Это обусловлено тем, что поддержание постоянного соединения с базой данных (поддерживаемой Entity Framework) является плохим тоном (из-за возникающей нагрузки на СУБД). Однако, так как контроллеры являются временными объектами DI, получения контекста в них допустимо.

Сам контекст содержит в себе список коллекций объектов, хранящихся в базе данных. Он не привязан к конкретному представлению или СУБД, поэтому один контекст может быть использован для различных баз данных. Пример простого контекста приведен в листинге 3:

```
public class ApplicationDbContext : DbContext
{
    public DbSet<SimpleClass> SimpleCollection { get; set; }

    public ApplicationDbContext(DbContextOptions options) : base(options)
    {
    }

    protected ApplicationDbContext()
    {
    }
}
```

Листинг 3 – Простой контекст базы данных.

Приведенный в листинге 3 класс контекста базы данных содержит одну коллекцию объектов класса `SimpleClass`. В каждом классе, наследующем класс `DbContext` необходимо создавать конструктор от аргумента `DbContextOptions`. Со-

здание второго конструктора (без аргументов) не обязательно, однако может потребоваться для тестирования.

Взаимодействие между сервисами

Для взаимодействия между сервисами используется протокол HTTP. Для общения между сервисами на главном (здесь и далее главным сервисом будет называться сервис, взаимодействующий с пользователем) сервисе создаются классы – интерфейсы сервисов. Централизация расположения этих интерфейсов облегчает дальнейшую модификацию системы, например, добавление общих заголовков к отправляемым запросам [10].

Базовым классом всех сервисов является *Service*, в котором определены функции-обертки над низкоуровневыми операциями по отправке HTTP-запросов и анализе ответов. Пример простейшей обертки над отправкой GET-запроса приведен в листинге 4:

```
protected async Task<HttpResponseMessage> Get(string addr)
{
    using (var client = new HttpClient())
    {
        try
        {
            return await client.GetAsync(GetAddress(addr));
        }
        catch { return null; }
    }
}
```

Листинг 4 – Обертка над GET-запросом.

Запрос в листинге 4 возвращает либо ответ от сервиса (как объект класса *HttpResponseMessage*), либо *null* в случае ошибки. Функция поддерживает асинхронное выполнение, о чем свидетельствует тип возвращаемого значения *Task<>* и наличие ключевого слова *async* в заголовке метода. Механика работы системы *async/await* сложна, в отличие от её использования. Для асинхронной работы достаточно пометить метод словом *async*, обернуть возвращаемый тип в *Task<>* и вызвать метод, применяя ключевое слово *await* (как вызывается *GetAsync()* у объекта *client*). Разработанные таким образом методы будут выполняться параллельно, если это возможно без нарушения логики работы.

Классы, унаследованные от класса `Service`, реализуют более высокоуровневую логику. Одним из таких классов является сервис учетных записей. Пример его метода приведен в листинге 5:

```
public async Task<List<string>> GetUserInfo(string username)
{
    var addr = $"user/{username}";
    var httpResponseMessage = await Get(addr);
    if (httpResponseMessage == null || httpResponseMessage.Content == null)
        throw new ServiceNotAvailableException(addr);
    return JsonConvert.DeserializeObject<List<string>>(await httpResponseMessage.Content.ReadAsStringAsync());
}
```

Листинг 5 – Метод получения информации о пользователе.

В листинге 5 в методе описана логика получения ответа от сервера и десериализации ответа в список объектов (в данном случае – строку). Переменная `addr` задается как адрес метода контроллера на целевом сервисе. В данном случае будет произведен поиск метода, отвечающего адресу `user/%строка%`. По умолчанию все ответы сервисов ASP.NET Core сериализуются в JSON (JavaScript Object Notation, система обозначений объектов JavaScript). JSON легко читается людьми, что облегчает отладку сервисов.

Пользовательский интерфейс

В приложениях ASP.NET Core используется технология серверной шаблонизации, при которой страница генерируется на сервере в соответствии с текущими данными, а затем отправляется пользователю. В ответ на его действия генерируются новые страницы. Для описания шаблона в формате `.cshtml` используется HTML вместе с языком Razor, который позволяет с легкостью использовать объекты языка C# внутри шаблона страницы. Пример простой страницы приведен в листинге 6.

```
@{
    ViewData["Title"] = "Ошибка";
```

```
}  
@model NewsFeed.Models.Shared.ErrorModel  
<h2>Ошибка</h2>  
Код ошибки: @Model.Code  
<hr />  
Текст ошибки: @Model.Message
```

Листинг 6 – Шаблон страницы.

В листинге 6 приведен простой шаблон страницы с сообщением об ошибке. Он начинается с указания свойства Title словаря ViewData, которое обычно используется в качестве заголовка страницы. Далее указывается используемая на странице модель, то есть набор данных, для которых данная страница служит представлением (в соответствии со схемой Модель-Представление-Контроллер, MVC [8]). Сами страницы генерируются из контроллеров в ответ на обращения к ним. После указания модели идет обычный HTML код. Части, написанные с использованием Razor, начинаются с символа @. Например, такой частью является @Model.Code, который соответствует коду ошибки из модели. При отправке данной страницы пользователю все элементы Razor будут заменены на HTML код или текст.

Шаблоны по умолчанию должны располагаться в папке Views в корневой директории проекта, в подпапках, соответствующих контроллерам. Например, шаблоны, относящиеся к контроллеру HomeController, должны располагаться в папке Views/Home. Исключение составляют два служебных шаблона, _ViewImports.cshtml и _ViewStart.cshtml. Шаблон _ViewImports.cshtml содержит в себе импортируемые во все шаблоны пространства имен, а также вспомогательные элементы Razor. В _ViewStart.cshtml находятся общие для всех шаблонов директивы Razor. Например, в файле _ViewStart.cshtml обычно указывают используемый макет страниц (мастер-страницу), который будет применен к каждому шаблону страницы.

Макет обычно содержит в себе большую часть разметки HTML, используемую

в обычных страницах, а также навигационные элементы. Располагается он как правило в папке Views/Shared, под названием _Layout.cshtml. Пример простого макета приведен в листинге 7.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>@ViewData["Title"]</title>
  <environment exclude="Development">
    <link rel="stylesheet"
href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />
  </environment>
</head>
<body>
  <nav class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
      <div class="navbar-header">
        <a asp-controller="Home" asp-action="Index" class="navbar-brand">Домашняя
страница</a>
      </div>
    </div>
  </nav>
  <div class="container body-content">
    @RenderBody()
  <hr />
  <footer>
    <p>&copy; 2018</p>
  </footer>
</div>
</body>
</html>
```

Листинг 7 – Макет страницы.

В листинге 7 большая часть является обычным кодом HTML страницы, хотя и с некоторыми добавлениями. В частности, здесь всё также используются элементы языка Razor, например, при обозначении заголовка страницы. Также есть вспомогательные теги HTML, такие как `<environment>`. Содержимое данного тега будет включаться (include) или исключаться (exclude) в зависимости от среды выполнения приложения. Это может быть полезно для отладки.

В макете также показан пример использования дополнительных атрибутов для указания ссылок на действия контроллеров на примере домашней страницы. Для создания ссылки на неё используется не относительный путь, а атрибуты `asp-controller=имя_контроллера` и `asp-action=имя_функции`. При их указании финальная ссылка на действие будет генерироваться динамически при отправке страницы пользователю, что позволяет избежать проблем при смене адресов доступа в приложении.

Наиболее важным элементом макета является вызов функции `RenderBody()`. Именно на её место в дальнейшем будет вставлена страница, которую запросил пользователь.

Сбор статистики

Для сбора статистики используется брокер очередей RabbitMQ. Его использование обуславливается тем, что сообщения статистики необходимо отправлять быстро и не дожидаясь ответа от сервера. Брокер при этом хранит сообщения и выдает их по требованию сервису сбора статистики, который, в свою очередь, сохраняет необходимую статистику в базу данных.

Сервис статистики оформляется как веб-API приложение, так как от него не требуется серверной шаблонизации. Также на данном сервисе хранятся классы для взаимодействия с брокером RabbitMQ. RabbitMQ — платформа, реализующая систему обмена сообщениями между компонентами программной системы на основе стандарта AMQP (от англ. Advanced Message Queuing Protocol - открытый протокол для передачи сообщений между компонентами системы)[11]. Другими

словами, в разрабатываемой системе Rabbit MQ - это программное обеспечение, при подключении к которому сервисы могут отправлять и получать сообщения. Необходимость отправки сообщения сервисом задается разработчиком. В свою очередь менеджер очередей Rabbit MQ сохраняет отправленные сервисами сообщения до тех пор, пока другие сервисы, которым было адресовано сообщение, не подключатся и не получат его. Для хранения сообщений брокер использует очереди, при подключении к которым каждый из компонентов системы определяет сообщения, на которые он будет подписан. Такой обмен позволяет упростить взаимодействие компонентов и достигнуть экономии ресурсов, а также обеспечить независимость компонентов и гарантировать последовательную обработку данных.

Взаимодействие источников сообщений с брокером является полностью асинхронным, а взаимодействие получателя – сервиса статистики – асинхронным, основанным на событиях. Метод отправки сообщений в очередь приведен в листинге 8:

```
public void Publish(Event @event)
{
    new Thread(o =>
    {
        try
        {
            @event = o as Event;
            if (!connection.IsConnected)
                connection.TryConnect();

            @event.OccurenceTime = DateTime.Now;
            var name = @event.GetType().FullName;

            using (var channel = connection.CreateModel())
            {
                channel.ExchangeDeclare(exchange: exchangeName, type: "direct");
```

```

        var body = Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(@event));
        channel.BasicPublish(exchange: exchangeName,
            routingKey: name,
            basicProperties: null,
            body: body);
    }
}
catch (Exception e)
{
    logger.LogCritical($"Failed to publish event, {e}");
}
}).Start(@event);
}

```

Листинг 8 – Метод отправки сообщений в очередь.

В методе в листинге 8 создается отдельный поток для отправки сообщения, после чего он запускается с аргументов `@event`. Такой способ запуска гарантирует моментальный возврат в вызвавшую отправку функцию, что позволяет не прерывать работу для отправки статистики. Внутри потока выполняется небольшая функция, которая проверяет соединение, записывает в объект события время его возникновения (которое считается как время попадания в функцию, после чего отправляет объект брокеру. При поступлении нового сообщения в очередь, брокер уведомляет подписавшихся на получение этих событий получателей. Сервис статистики, получив уведомление, вызывает метод обработки событий, приведенный в листинге 9:

```

private async Task ProcessEvent(string eventName, string message)
{
    if (handlers.ContainsKey(eventName))
    {
        var type = Type.GetType(eventName);
        var template = typeof(EventHandlers.EventHandler<>);
        var @event = JsonConvert.DeserializeObject(message, type);
    }
}

```

```

        foreach (var handler in handlers[eventName])
        {
            try
            {
                string id = (@event as Event).Id;
                {
                    var genericType = template.MakeGenericType(type);
                    await
                    (Task)genericType.GetMethod("Handle").Invoke(handler, new object[] { @event });
                }
            }
            catch (Exception e)
            {
                logger.LogCritical($"Exception while trying to execute event han-
dler: {e}");
            }
        }
    }
}

```

Листинг 9 – Метод обработки событий.

В качестве аргументов метод в листинге 9 принимает имя класса события и само событие в формате JSON. Далее, при наличии обработчика такого класса событий, этот обработчик извлекается из коллекции обработчиков и для него вызывается метод Handle (Event event). В приведенном коде для этого применяется рефлексия.

Реализация масштабируемости

Несмотря на то, что сервисная архитектура и стиль REST позволяют легко горизонтально масштабировать систему, для автоматизации процесса необходимо

приложить усилия. В разрабатываемой системе для этого выделен отдельный сервис-координатор, отвечающий за балансировку нагрузки между сервисами. Принцип работы координатора прост: анализируя полученные от сервиса статистики данные о нагрузке на сервисы, а также время доступа одного сервиса к другому, координатор меняет связи между сервисами для равномерного распределения нагрузки на систему.

Для сбора статистики запросов, а также изменения связей в каждом сервисе, используются объекты промежуточного слоя (англ. Middleware). Они располагаются в конвейере обработки запросов ASP.Net Core, позволяя анализировать полученные данные до их попадания в основное приложение. Пример промежуточного слоя для сбора статистики запросов приведен в листинге 10.

```
public class RequestStatisticsMiddleware
{
    private readonly RequestDelegate next;
    private readonly IEventBus eventBus;

    public RequestStatisticsMiddleware(RequestDelegate next, IEventBus eventBus)
    {
        this.next = next;
        this.eventBus = eventBus;
    }

    public virtual async Task Invoke(HttpContext context)
    {
        var connection = context.Connection;
        eventBus.SendRequestEvent(
            $"{connection.LocalIpAddress}:{connection.LocalPort}",
            $"{connection.RemoteIpAddress}:{connection.RemotePort}",
            context.Request.Path.ToString(),
            RequestType.Gateway);
    }
}
```

```
        await next(context);  
    }  
    }.
```

Листинг 10 – Промежуточный слой сбора статистики.

Как показано в листинге 10, классы, относящиеся к промежуточному слою, не наследуются от каких-либо специализированных классов. Требованиями к ним являются наличие объекта класса `RequestDelegate` в данном классе и метод `Invoke(HttpContext)`. Делегат `RequestDelegate` отвечает за метод `Invoke` промежуточного слоя, стоящего следующим в конвейере обработки. Таким образом, если какой-либо слой не собирается препятствовать запросу и пропускает его дальше, он вызывает метод делегата `RequestDelegate`. В методе `Invoke` данного промежуточного слоя отправляется сообщение в очередь `RabbitMQ`, откуда оно будет извлечено сервисом статистики. Важно отметить, что методы `Invoke` напрямую влияют на время отклика, и потому нежелательно размещать в них большие синхронные операции.

Аналогичным образом реализуется промежуточный слой координатора, встраиваемый во все сервисы. Он анализирует пакеты на предмет наличия в них заранее определенного заголовка и, при его наличии, интерпретирует значение заголовка как одну из допустимых команд. Достаточный для реализации координатора набор состоит из команды установки нового адреса и команды замера времени доступа от одного сервиса к другому. Данные о нагрузке на сервисы поступают от сервиса статистики, собирающего информацию о запросах при помощи промежуточного слоя из листинга 5.1. Анализируя эти данные, а также обладая информацией о времени отправки запросов между сервисами, координатор динамически меняет связи сервисов для снижения нагрузки и времени отклика системы в целом.

Использование координатора позволяет отказаться от записи необходимых ад-

ресов в конфигурационный файл каждого сервиса. Вместо этого при запуске координатора происходит инициализация всех известных координатору сервисов. В соответствии с заданными зависимостями каждому сервису отправляются необходимые для его работы адреса. При добавлении администратором новых сервисов они инициализируются, а периодически проходит балансировка на основании ранее описанных данных.

Реализация отказоустойчивости

Отказоустойчивость гарантирует работоспособность системы, несмотря на внутренние неполадки в системе. Существуют разные способы сохранения функционирования, например, деградация функциональности, полный откат операции, очередь сообщений и другие.

Деградация функциональности

Деградация функциональности - принцип сохранения работоспособности всего приложения при частичной потере функциональности.

В данном приложении отказоустойчивость распространяется на недоступность сервисов. То есть в том случае, когда один из сервисов не способен дать ответ на запрос из-за недоступности, необходимо вывести пользователю информацию, которую удалось обработать без участия недоступного сервиса, и сообщение об ошибке.

Деградацию, как правило, используют в GET-запросах, так как это не влечет никакой потери пользовательских данных.

Откат

Деградация функциональности не решает проблему в том случае, когда запрос влечет изменение данных (например, PUT, DELETE). Потому что операция изменения данных является единой транзакцией, а значит, частичная работа над ними невозможна.

Откат операции означает, что метод, который агрегирует запрос пользователя в запросы к сервисам, перед внесением изменений должен сохранять снимок исходного состояния изменяемых данных. Это необходимо для того, чтобы в про-

цессе выполнения запроса при возникновении ошибки система могла стереть только что внесенные изменения, вернуть исходное состояние и выдать пользователю ошибку о невозможности обработки запроса.

Очередь сообщений

Суть данного метода отказоустойчивости заключается в том, что в случае, если сервис недоступен и невозможно выполнить запрос, изменяющий данные, искомый запрос становится в очередь на выполнение.

Когда задача выделяется из очереди, происходит попытка ее выполнения. Если попытка оказалась успешной, задача считается выполненной, и из очереди обрабатывается следующую задачу. При неудаче задача снова ставится в очередь.

Очередь организована классом, имеющим методы постановки задачи в очередь и обработки элемента в очереди. Реализация такого процесса приведена в листинге 11:

```
private static void TimerCallback(object o)
{
    Func<Task<bool>> task = null;
    while (allTasks.Count > 0)
        if (allTasks.TryDequeue(out task) && !task.Invoke().Result)
            allTasks.Enqueue(task);
}
```

Листинг 11. Обработка элемента очереди.

Способ постановки запроса в очередь приведен в листинге 12.

```
MyQueue.Retry(async () =>
{
    using (var client = new HttpClient())
    {
        response = await userService.ChangeName(oldName, newName);
        if (response != null)
```

```
        return true;

        return false;

    } });
```

Листинг 12. Пример постановки запроса в очередь

Реализация авторизации в системе

Для реализации регистрации и авторизации в приложении прежде всего необходимо создать сервис работы с хранилищем данных пользователя. В разрабатываемой системе за данный функционал отвечает сервис аккаунтов. В основе его реализации лежит модель под названием User, которая содержит следующие поля:

- Имя
- Пароль
- Роль

В контроллере сервиса необходимо реализовать методы работы с данными пользователя, например, сохранения информации в базу данных при регистрации или проверки данных пользователя при попытке авторизации и другие сопутствующие методы.

После реализации сервиса аккаунтов необходимо внедрить функционал регистрации и авторизации в главный сервис, во-первых, чтобы в системе мог работать только авторизованный пользователь, во-вторых, для проверки актуальности сессии клиента, работающего с приложением. В связи с тем, что проверка авторизации пользователя должна производиться перед выполнением любого запроса, требуется встроить данную проверку в процесс обработки запросов при помощи добавления компонентов промежуточного слоя.

Компонент промежуточного слоя - программное обеспечение, выстраиваемое в виде конвейера приложения ASP.NET для обработки запросов и откликов. Конвейер запросов ASP.NET Core состоит из последовательности делегатов запроса, вызываемых один за другим. Каждый из делегатов может выполнять операции до и после следующего делегата. Делегат также может принять решение не передавать запрос следующему делегату, что называется замыканием конвейера запро-

сов. Добавление компонентов промежуточного слоя происходит в методе `Configure` файла `Startup.cs` проекта, при этом порядок, в котором компоненты добавляются в метод, определяет порядок их вызова при запросах и обратный порядок для отклика.

Обычно программное обеспечение промежуточного слоя принято инкапсулировать в класс, который содержит метод `Invoke`. В данном методе прописывается основной функционал компонента. В качестве базового компонента можно создать класс `AuthorizationMiddleware`, в котором описать основные функции при авторизации, а остальные компоненты для реализации деталей авторизации сделать наследуемыми от него. Конструктор данного класса принимает на вход следующий делегат для вызова и экземпляр класса, работающего с токенами. Метод `Invoke` класса `AuthorizationMiddleware` приведен в листинге 13. В качестве параметра данному методу передается объект типа `HttpContext`, который используется для хранения специальной HTTP-информации о запросе.

```
public virtual async Task Invoke(HttpContext context)
{
    //проверка наличия ключевого слова "Authorization" в контексте
    if (context.Request.Cookies.Keys.Contains(AuthorizationWord))
    {
        var auth = context.Request.Cookies[AuthorizationWord];
        await CheckAuthorization(context, auth);
    }
    //разделение пути на составляющие и проверка на соответствие выделенным в методе
    GetAnonymousPaths путям
    else if (context.Request.Path.Value.Split('/').Intersect(GetAnonymousPaths()).Any())
    {
        await this._next(context);
    }
    else
    {
        var message = "No authorization header provided";
        await ReturnForbidden(context, message);
    }
}
```

```
}  
}
```

Листинг 13. Код метода `Invoke` класса `AuthorizationMiddleware`.

Представленный метод осуществляет проверку сессии пользователя, работающего с приложением. Если переданный контекст не содержит информации о данных для авторизации, то необходимо разделить путь запроса на составляющие и посмотреть, является ли запрос анонимным и передать управление следующему компоненту промежуточного слоя, в противном случае вернуть ответ, содержащий отказ в работе с приложением. Анонимный запрос – это запрос, позволяющий работать с приложением без авторизации. К таким запросам относятся запросы на авторизацию и регистрацию. Код метода `CheckAuthorization`, который проверяет актуальность данных для авторизации пользователя, переданных в контексте представлен на листинге 14.

```
protected async Task CheckAuthorization(HttpContext context, string auth)  
{  
    //проверка формата токена  
    var match = Regex.Match(auth, @"Bearer (\S+)");  
    if (match.Groups.Count == 1)  
    {  
        await ReturnForbidden(context, "Invalid token format");  
    }  
    else  
    {  
        var token = match.Groups[1].Value;  
        //проверка токена на актуальность с данными в хранилище токенов  
        var result = tokensStore.CheckToken(token);  
        //если токен валиден  
        if (result == CheckTokenResult.Valid)  
        {  
            //создание объекта для хранения данных пользователя  
            ClaimsIdentity identity = new ClaimsIdentity(new List<Claim>  
            {
```

```

        new Claim(UserWord, tokensStore.GetNameByToken(token)),
        new Claim(RoleWord, tokensStore.GetRoleByToken(token))
    }, "CustomAuthenticationType");
    context.User.AddIdentity(identity);
    await this._next(context);
}
//если токен закончил свое действие
else if (result == CheckTokenResult.Expired)
    await ReturnForbidden(context, "Token has expired");
else
    await ReturnForbidden(context, "Token not valid");
}
}

```

Листинг 14 – Код метода Check Authorization.

Стоит отметить, что формат токенов, выдаваемых пользователям, можно выбрать самостоятельно, а затем проверять соответствие информации из контекста выбранному формату для исключения внедрения злоумышленников в процесс обработки запросов. Еще один важный момент – создание объекта типа `ClaimsIdentity` для хранения данных пользователя. Он позволяет хранить информацию о текущем пользователе, тем самым уменьшая количество обращений к базе данных. Константы `AuthorizationWord`, `UserWord` и `RoleWord` имеют значения, которые приведены в листинге 15:

```

public static string AuthorizationWord = "Authorization";
public static string UserWord =
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name";
public static string RoleWord =
"http://schemas.microsoft.com/ws/2008/06/identity/claims/role".

```

Листинг 15 – Значения констант для авторизации

Значения констант выбраны не случайно, они позволяют интегрироваться с системой ASP.NET Identity.

После реализации базового компонента промежуточного слоя `AuthorizationMiddleware` требуется описать компонент `NewsFeedAuthorizationMiddleware`, наследуемый от него и непосредственно отвечающий за авторизацию пользователя при работе с приложением. Код его основного метода `Invoke` приведен в листинге 16.

```
public override async Task Invoke(HttpContext context)
{
    if (context.Request.Headers.Keys.Contains(AuthorizationWord))
    {
        await this._next(context);
    }
    else
        await base.Invoke(context);
}
```

Листинг 16 – Код метода `Invoke` класса `NewsFeedAuthorizationMiddleware`.

Данный метод проверяет, содержится ли в заголовках переданного объекта `HttpContext` информация об авторизации. Если содержится, то вызывается следующий компонент конвейера обработки запросов, если не содержится, то вызывается метод `Invoke` базового класса `AuthorizationMiddleware`. Класс `TokenStore` осуществляет взаимодействие с хранилищем токенов и работу с ним. Здесь токен – это объект типа `TokenInfo`. Он содержит в себе не только сам токен, но и данные пользователя, дату и время его последнего обращения, а также дату и время истечения токена.

Класс `TokenStore` осуществляет следующие функции:

- выдача токена;
- проверка токенов;
- удаление истекших токенов;
- получение данных пользователя по токenu.

После подготовки и реализации описанных выше компонентов можно приступить к внедрению авторизации в основной проект. Для этого необходимо моди-

фицировать методы `ConfigureServices` и `Configure` в файле `Startup.cs` основного проекта. В метод `ConfigureServices` необходимо при помощи команды `services.AddSingleton<...>` добавить сервис аккаунтов и класс работы с токенами. А в метод `Configure` – команды, связанные с добавлением компонентов промежуточного слоя и авторизации, приведенные в листинге 17.

```
app.UseMiddleware<NewsFeedAuthorizationMiddleware>();  
app.UseAuthentication();
```

Листинг 17 – Код добавления компонентов промежуточного слоя к основному проекту.

Очень важно вставить данные команды до добавления статических файлов, чтобы проверка авторизации проводилась до начала основной обработки запросов пользователя.

Реализация авторизации в системе через социальные сети

На сегодняшний день многие пользователи имеют множество аккаунтов в различных приложениях, в связи с этим иногда гораздо удобнее авторизоваться в новом приложении при помощи уже существующего аккаунта другого сервиса, нежели заново регистрироваться. Возможность такой авторизации предоставляет фреймворк OAuth 2, который позволяет приложениям осуществлять ограниченный доступ к пользовательским аккаунтам на HTTP-сервисах, например, аккаунтам в популярных социальных сетях. Он работает по принципу делегирования аутентификации пользователя сервису, на котором находится аккаунт пользователя, позволяя стороннему приложению получать доступ к аккаунту пользователя. В данном разделе подробно описана реализация авторизация пользователя через Google в разрабатываемом приложении.

Для аутентификации через любую социальную сеть, необходимо зарегистрировать свое приложение в консоли разработчика выбранного провайдера. Для рассматриваемого приложения была выбрана платформа Google. Как зарегистрировать свое приложение подробно описано в источнике [11]. В консоли раз-

работчика настроен адрес обратного вызова локального IdentityServer путем добавления пути sign-google к базовому адресу. После регистрации выдается AppId и AppSecret для разрабатываемого приложения.

Использование токенов безопасности

Чтобы предоставлять доступ только авторизованным пользователям на некоторый период времени, используются токены. Для реализации авторизации через OAuth рассматриваются 2 вида токенов: access-токен (для доступа пользователя к функционалу приложения) и refresh-токен (передается серверу по истечению сессии для выдачи свежего access-токена).

При авторизации клиентское приложение Client обращается к серверу аутентификации IdentityServer и получает access-токен, который затем использует в качестве Bearer-токена для вызова главного сервиса приложения.

Bearer-токен обладает свойством, что любой, кто им владеет, может использовать токен любым способом. Доказательства того, что предъявитель токена является владельцем - не требуется. Создаваемый access-токен имеет три поля: идентификатор, владелец и время истечения. Хранилище токенов организовано так же, как и другие таблицы базы данных приложения. Работа с токенами происходит в контроллере сервиса авторизации.

Вся работа с токенами проходит на стороне главного сервиса в классе AuthorizationMiddleware, работа которого описана в разделе реализации авторизации пользователя по логину и паролю.

Работа с Google.

Работа с авторизацией через социальную сеть происходит полностью на стороне главного сервиса.

После регистрации приложения в консоли разработчика, сначала в классе Startup в методе ConfigureServices нужно прописать обработчик аутентификации Google, представленный на листинге 18.

```
services.AddAuthentication(v =>
```

```
{
```



```

v.DefaultAuthenticateScheme = GoogleDefaults.AuthenticationScheme;

v.DefaultChallengeScheme = GoogleDefaults.AuthenticationScheme;

}).AddGoogle(ops => {

    ops.ClientId = "863755605836-hbil....apps.googleusercontent.com";

    ops.ClientSecret = "5vzEp-5...JSEi5f";

});

```

Листинг 18. Обработчик аутентификации Google

Здесь ops.ClientId и ops.ClientSecret - это те Id и Secret, который выдал Google при регистрации приложения.

В контроллере необходим метод-обработчик кнопки авторизации через Google. Его ключевым атрибутом является указание в “шапке” схемы Google-аутентификации, которая задана в Startup.cs. Пример приведен на листинге 19.

```

[HttpGet("googleauth")]

[Authorize(AuthenticationSchemes = GoogleDefaults.AuthenticationScheme)]

public async Task<IActionResult> GoogleAuth()

{

    return RedirectToAction(nameof(Index));

}

```

Листинг 19. Метод контроллера по обработке Google-авторизации

Реализация основных функциональных требований к системе

В этом разделе приведена подробная реализация ответа на запрос о получении истории просмотров, как пример одной из основных функциональных требований. В ответ на внешний запрос, происходящий, например, по нажатию на кнопку в интерфейсе, должна приходить страница, показывающая историю просмотров пользователя. Для хранения истории используются записи NewsViewedData. Каждая запись содержит в себе указание статьи, кем она была прочитана, а также время чтения (данное время является временем отправки текста статьи с сервера).

Эти записи хранятся на сервисе статей.

Для того, чтобы не сильно нагружать страницу истории, на неё выводятся только записи за последний час. При желании пользователь может получить историю и за больший период. Метод получения истории просмотров из сервиса статей приведен в листинге 20.

```
[HttpGet("history/{name}")]
public async Task<List<(string str, int id)>> GetHistoryForUser(string name, TimeSpan
timeSpan)
{
    var date = DateTime.Now - timeSpan;
    var views = db.NewsViewedData.Where(d => d.User == name && d.Date > date);
    var result = new List<(string str, int id)>();
    if (views.Any())
    {
        var news = views.AsEnumerable().Select(d => (date: d.Date, news:
db.News.FirstOrDefault(n => n.Id == d.NewsId)))
        .Where(t => t.news != null);
        if (news.Any())
            result = news.Select(t => ($"{t.date} - {t.news.Header}", t.news.Id)).ToList();
    }
    return result;
}
```

Листинг 20 – Метод получения истории просмотров с сервиса новостей.

В листинге 18 сначала происходит вычисление минимальной даты, затем получение всех записей истории для пользователя, дата просмотра которых не превосходит минимальной, а затем – поиск и создание списка новостей, соответствующих данным просмотрам. Один из аргументов метода передается в строке запроса, второй – как аргумент GET-запроса, то есть через `&timeSpan=<интервал>`. Например, строка запроса может иметь вид `“http://localhost:8888/history/user1×pan=0”`. Результатом работы метода является список, состоящий из дат просмотра и заголовков новостей, а также иденти-

фикаторов новостей в базе. Метод получения страницы с историей из основного сервиса приведен в листинге 21.

```
[HttpGet("history")]
public async Task<IActionResult> History(string username = null)
{
    if (User.Identities.Any(i => i.IsAuthenticated))
        username = User.Identities.First(i => i.IsAuthenticated).Name;
    if (string.IsNullOrEmpty(username))
        return View("Error", new ErrorModel(new ObjectResult("Пользователь не найден")));
    var content = (await apiController.GetViewsHistory(username,
        TimeSpan.FromHours(1))).Value as List<(string, int)>;
    return View(content);
}
```

Листинг 21 – Метод получения страницы с историей просмотров.

В методе из листинга 21 сначала идет проверка, авторизован ли текущий пользователь. Если пользователь не авторизован – будет возвращена страница с именем шаблона `Error`, на которой отобразится сообщение о том, что пользователь не найден. Если же всё в порядке – произойдет запрос истории просмотров с сервиса новостей, после чего на основании полученного ответа будет сгенерирована страница. Код шаблона страницы приведен в листинге 22.

```
@{
    ViewData["Title"] = "History";
}
@model List<(string title, int id)>

@foreach (var (title, id) in Model)
{
    <div class="row">
        <div class="col-md-8">
            @title
        </div>
        <div class="col-md-4">
```

```

        <a asp-action="Details" asp-route-Id="@id">Подробнее</a>
    </div>
</div>
}

```

Листинг 22 – Шаблон страницы истории просмотров.

Как видно из листинга 22, шаблон страницы просмотров достаточно прост: он не использует особого класса модели, только список, и включает в себя инструкции на языке Razor, создающие для каждого элемента списка строку в таблице. Таблица организуется при помощи элементов `<div>` с соответствующими классами Bootstrap. Это позволяет гибко настраивать внешний вид таблицы при помощи CSS.

Руководство пользователя

Данное руководство описывает варианты работы пользователя в разработанной системе.

1. Авторизация и регистрация

В связи с тем, что в приложении может работать только авторизованный пользователь, единственная страница, которая доступна в приложении до авторизации и является стартовой - страница авторизации.

Войти

Имя пользователя:

Пароль:

Регистрация Войти

Google

Воспользуйтесь аккаунтом Google для входа

Войти

© 2018

[О проекте](#)

Рисунок 13. Страница авторизации приложения.

Для начала работы в системе в открывшейся форме необходимо заполнить данные пользователя: логин и пароль. Предлагаемая форма может использоваться как для регистрации нового пользователя, так и для входа уже существующего в зависимости от выбранной опции после заполнения данных.

При успешном завершении операции будет осуществлен переход на страницу профиля авторизованного пользователя.

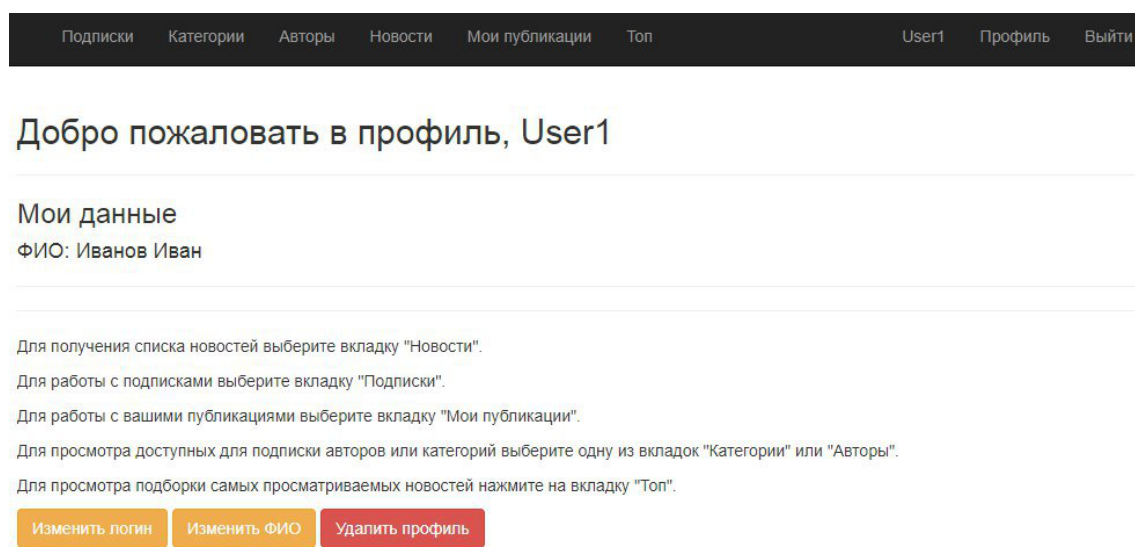
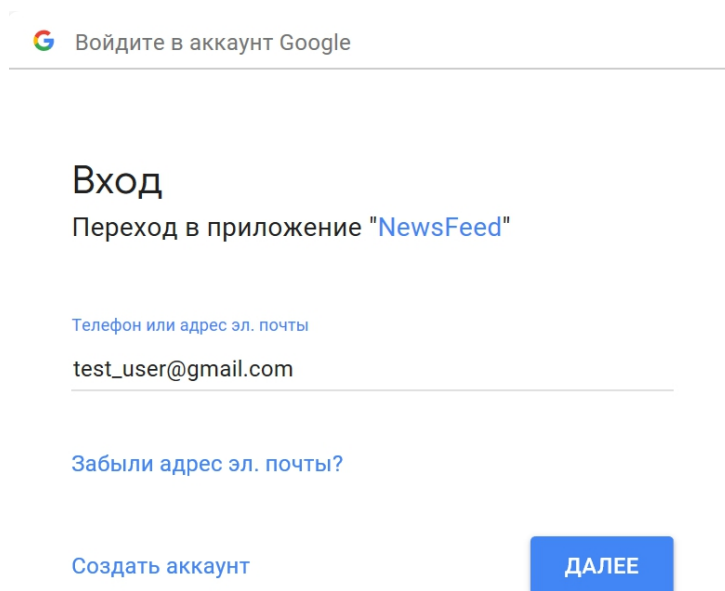


Рисунок 14. Переход на страницу профиля после успешной авторизации.

Вход в систему также может быть выполнен через социальную сеть. Для этого необходимо нажать на зеленую кнопку «Google» и ввести во всплывающее окно данные своего Google-аккаунта.



Войдите в аккаунт Google

Вход

Переход в приложение "NewsFeed"

Телефон или адрес эл. почты

test_user@gmail.com

Забыли адрес эл. почты?

Создать аккаунт

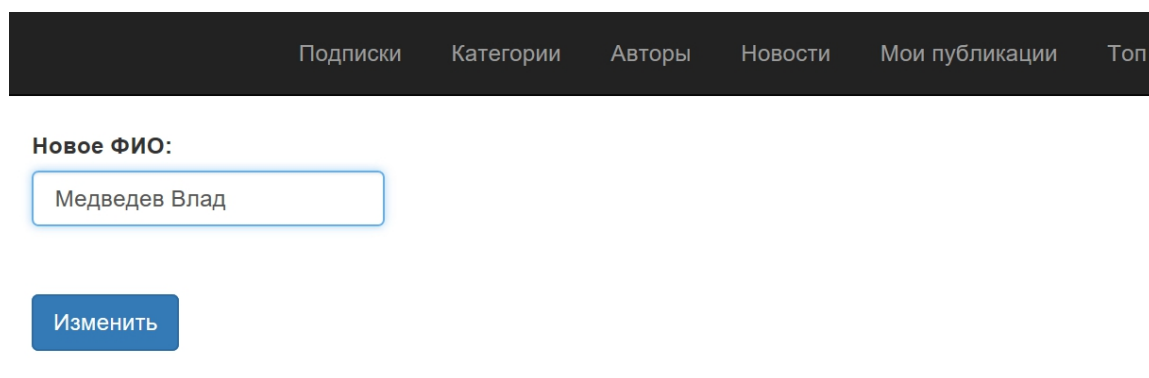
ДАЛЕЕ

Рисунок 15. Ввод данных во всплывающее окно Google.

При успешном завершении операции будет также осуществлен переход на страницу профиля авторизованного пользователя.

2. Вкладка «Профиль»

На данной странице отображается общая информация о работе в системе, а также есть возможность изменить данные профиля пользователя либо удалить профиль из системы. При выборе опции изменения данных откроется страница с формой для ввода новых данных пользователя, например, имени.



Подписки Категории Авторы Новости Мои публикации Топ

Новое ФИО:

Медведев Влад

Изменить

© 2018

[О проекте](#)

Рисунок 16. Страница изменения данных пользователя.

После окончания редактирования данных необходимо завершить операцию нажатием кнопки «Изменить». После чего пользователь будет перенаправлен на страницу своего профиля. При выборе опции удаления профиля пользователь будет переведен на страницу подтверждения удаления. На данной странице можно как подтвердить окончательное удаление профиля, так и отменить удаление.

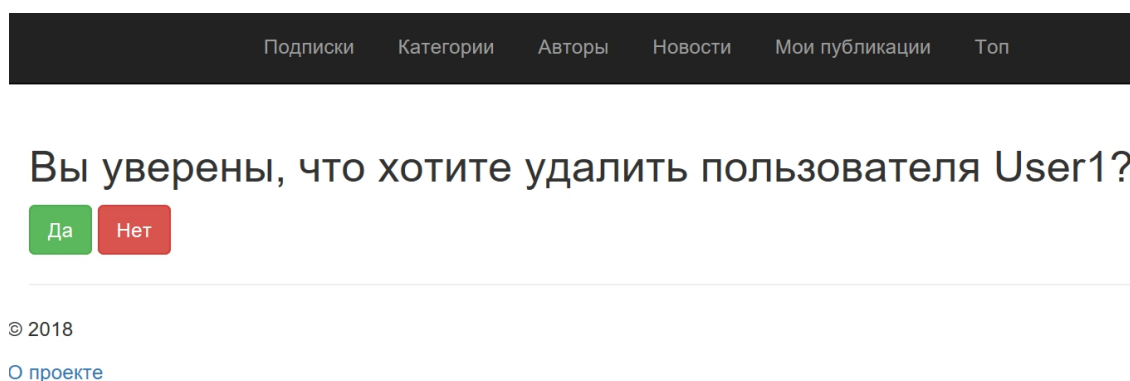


Рисунок 17. Страница подтверждения удаления профиля.

3. Вкладка «Подписки»

На данной странице осуществляется просмотр и управление подписками пользователя. Подписка может осуществляться на категории публикаций и конкретных авторов.

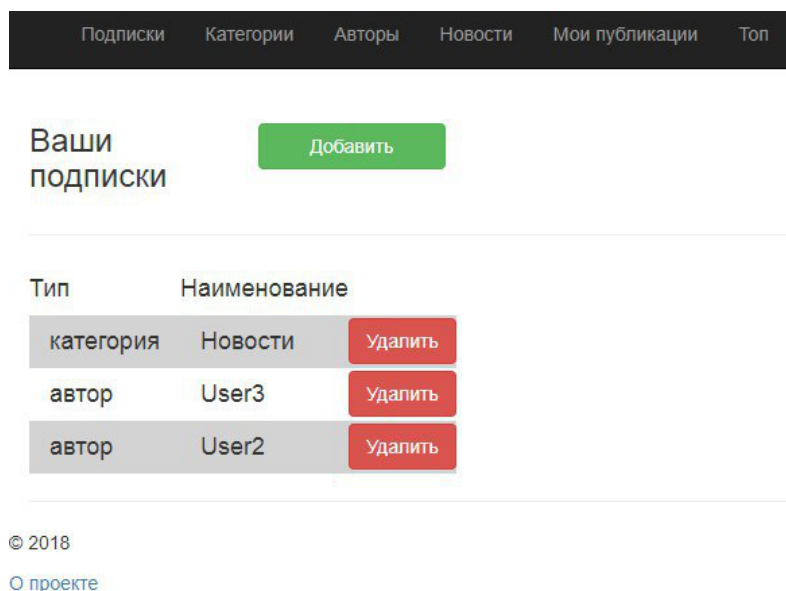


Рисунок 18. Страница управления и просмотра подписок пользователя.

Для добавления подписки необходимо нажать на кнопку «Добавить» и вписать имя автора или название категории в выделенное поле. Для удаления существующих подписок необходимо использовать кнопку удаления рядом с выбранной подпиской.

4. Вкладки «Категории» и «Авторы»

Данные разделы сайта могут быть использованы для ознакомления с существующими категориями и авторами публикаций. На странице «Категории» отображаются все доступные категории для подписки. При нажатии на название категории, открывается страница с подборкой новостей по выбранной категории.

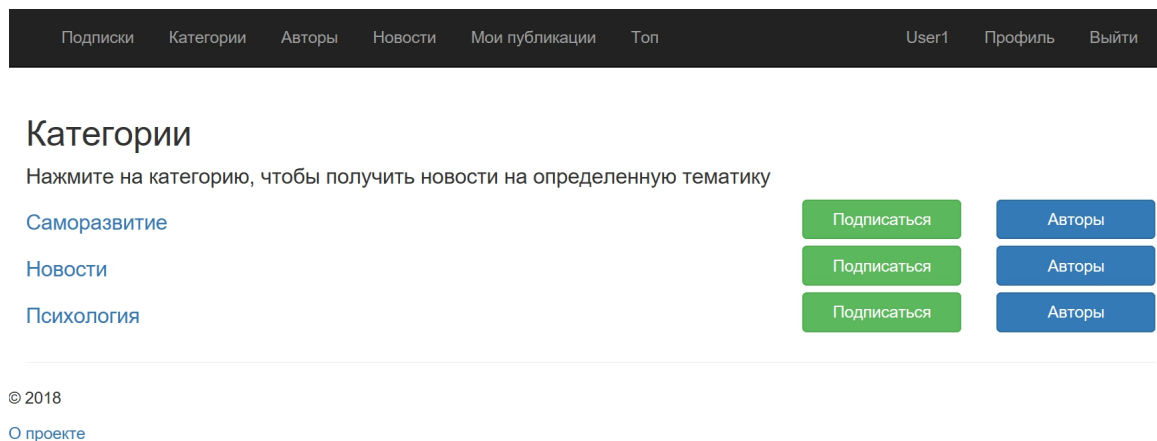


Рисунок 19. Страница категорий публикаций.

Раздел авторов предоставляет список всех пользователей, которые являются авторами публикаций.

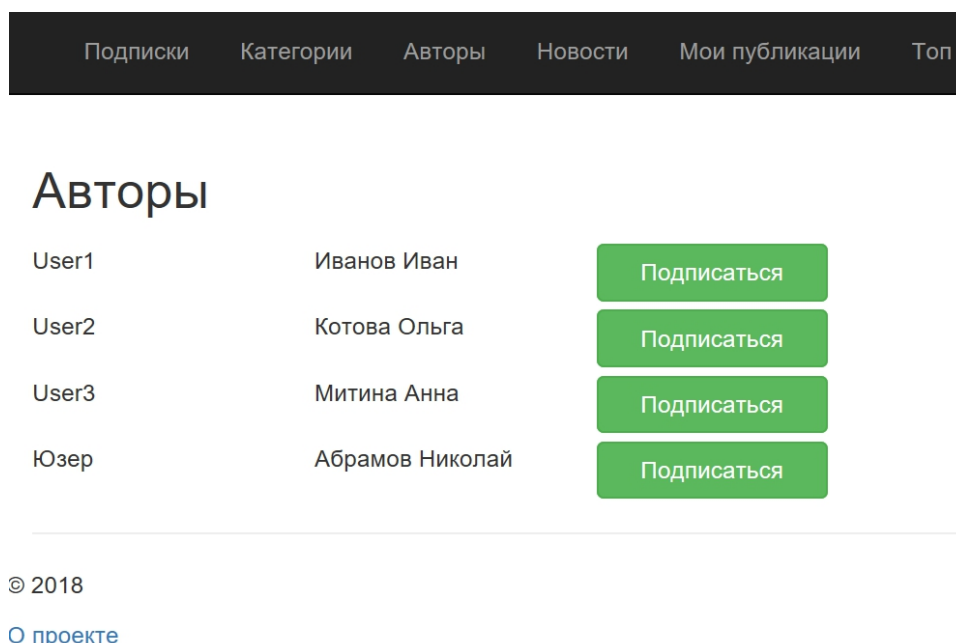


Рисунок 20. Страница со списком авторов.

5. Вкладка “Новости”

На данной странице осуществляется просмотр ленты новостей, сформированной на основе подписок пользователя, и переход к подробному просмотру выбранной публикации. Также пользователь может добавить свою новость и перейти на страницу просмотров публикаций.

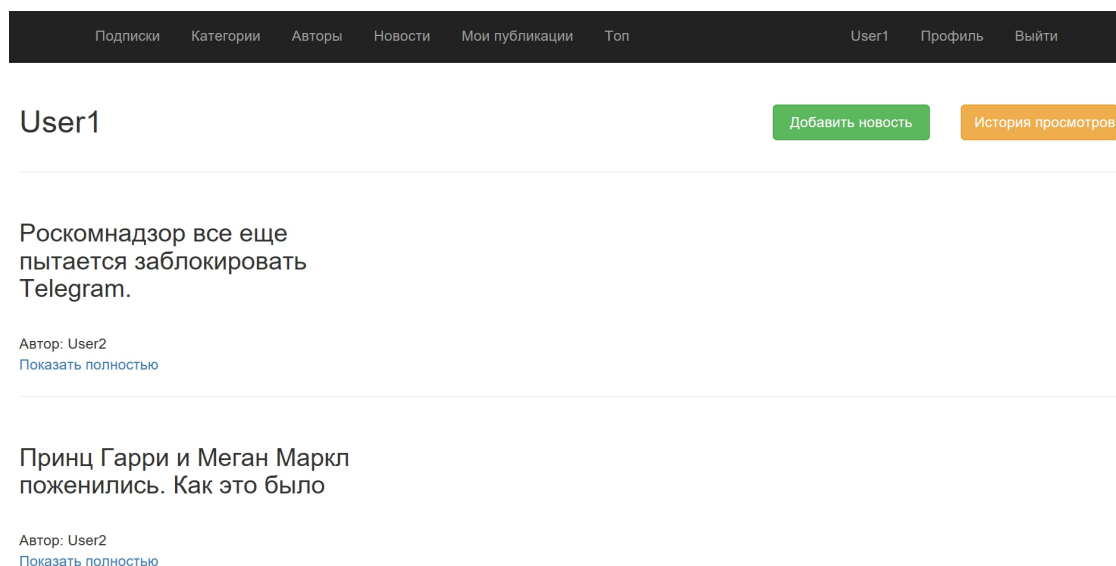


Рисунок 21. Страница с лентой новостей.

На странице добавления новости необходимо ввести заголовок и тело новости, а также выбрать категорию, к которой она будет отнесена. Завершение операции добавления происходит при нажатии кнопки «Добавить», после чего осуществляется переход на вкладку «Новости».

На странице истории просмотров отображается информация о дате и времени посещения прочитанных пользователем статей.

6. Вкладка “Мои публикации”

Данная страница содержит список публикаций авторизованного пользователя. Также в разделе доступен функционал удаления и добавления публикаций и доступа к истории просмотра новостей.

Ваши публикации, User1

7 ошибок, из-за которых вашему выступлению никто не поверит

[Удалить публикацию](#)

Автор: User1

[Показать полностью](#)

Искусство безделья. Как лень делает нас продуктивнее

[Удалить публикацию](#)

Автор: User1

[Показать полностью](#)

Рисунок 22. Список публикаций пользователя.

7. Вкладка «Топ»

Данный раздел сайта содержит подборку самых просматриваемых публикаций в приложении.

8. Выход из системы

Для выхода из системы необходимо нажать кнопку “Выйти” в правом углу интерфейса приложения.

Руководство администратора

В руководстве пользователя описаны основные варианты работы в системе при авторизации через аккаунт, которому присвоена роль «Пользователь». В данном руководстве приводится описание функционала для аккаунта с ролью «Администратор». Администратор портала может использовать как стандартный функционал приложения, доступный всем зарегистрированным пользователям, так и функционал администрирования приложения. Основные функции администратора – распределение нагрузки между сервисами системы и управление кате-

гориями публикаций.

Управление узлами системы

В разделе «Панель администратора» отображаются статистические данные о медиане времени отклика каждого из сервисов системы.

Подписки Категории Авторы Новости Мои публикации Топ User3 Профиль Панель администратора Выйти			
Панель администратора			
Доступные сервисы			
Тип сервиса	Адрес	Медиана времени отклика	
Accounts	http://localhost:58490/	0,1 sec	-
Authentication	http://localhost:58489/	0 sec	-
Gateway	http://localhost:58492/	0,19 sec	-
News	http://localhost:58493/	0,28 sec	-
Statistics	http://localhost:58494/	0,06 sec	-
Subscriptions	http://localhost:58495/	0,07 sec	-
			+

Рисунок 23. Информация о нагрузке сервисов и управление ими.

Согласно требованиям к функциональным характеристикам системы данный параметр для каждого из сервисов не должен превышать 800мс. Таким образом администратор, заметив значительные задержки во времени отклика сервиса, должен воспользоваться опцией добавления сервиса к системе. Для этого необходимо нажать зеленую кнопку с иконкой «+», после чего будет осуществлен переход на страницу добавления сервиса.

На данной странице необходимо выбрать тип сервиса для добавления, а также вписать адрес расположения сервиса в установленном формате. После нажатия кнопки «Добавить» новый сервис появится в списке, а время отклика у других сервисов такого же типа уменьшится. При отсутствии необходимости использования сервисы могут быть удалены.

Подписки

Категории

Авторы

Новости

Мои публикации

Топ

Тип сервиса:

News

▼

Адрес:

http://localhost:58999

Добавить

© 2018

[О проекте](#)

Рисунок 24. Страница добавления сервиса администратором.

1. Добавление категорий публикаций

Помимо распределения нагрузки между сервисами администратор также занимается управлением категориями публикаций. Данный функционал также доступен в разделе «Панель администратора».

Категории	
Название	
Саморазвитие	-
Новости	-
Психология	-
	+

Рисунок 25. Раздел управления категориями публикаций.

Для удаления и создания категорий используются кнопки «+» и «-». После нажатия зеленой кнопки «+» откроется страница добавления категории. В форму необходимо ввести название создаваемой категории сохранить изменения, нажав кнопку «Добавить».

Итоги практики

В ходе выполнения практикума по РСОИ был разработан портал просмотра ленты новостей, формирующейся на основе подписок пользователя. Перед началом разработки сформулировано техническое задание с общими и функциональными требованиями, а также требованиями к надежности и документации. Помимо требований к системе в техническом задании приведена топология системы, на основе которой разработаны требования к подсистемам. В конструкторском разделе описано проектирование сервисов по отдельности и системы в целом, проработаны сценарии функционирования и представлена архитектура сервисов. В технологическом разделе описаны основные принципы реализации ключевого функционала сервисов. Данная работа также включает в себя реализацию описанного проекта согласно разработанному техническому заданию. Реализованный функционал отлажен и протестирован.

Список литературы

1. Техническое задание. Требования к содержанию и оформлению (ГОСТ 19.201-78). [Электронный ресурс] Режим доступа: URL: <http://internet-law.ru/gosts/gost/31884/>
2. Вишневская Т.И., Романова Т.Н. Методология программной инженерии: Мет. указания к выполнению лабораторных работ М.: Изд-во МГТУ им. Н.Э. Баумана, 2017.-58 с.
3. Научно-исследовательский Центр CALS - Методология функционального моделирования IDEF0: ИПК Издательство стандартов, 2000 – 75 с.
4. Богданов А., Корхов В., Мареев В. Архитектуры и топологии многопроцессорных вычислительных систем. Издательство Интуит.Ру, 2013, -176 с.
5. Ньюмэн С. Создание микросервисов - Издательство Питер, 2016, - 392 с.
6. Amindsen M., Rubi S. RESTful Web APIs: Services for a Changing World, O'Reilly, 2013,- 300 p.
7. Announcing .NET Core 2.0 [Электронный ресурс] Режим доступа: URL: <https://blogs.msdn.microsoft.com/dotnet/2017/08/14/announcing-net-core-2-0>.

8. Develop ASP.NET Core MVC apps [Электронный ресурс] Режим доступа: URL: <https://docs.microsoft.com/ru-ru/dotnet/standard/modern-web-apps-azure-architecture/develop-asp-net-core-mvc-apps>
9. Entity Framework Core Quick Overview [Электронный ресурс] Режим доступа: URL: <https://docs.microsoft.com/en-us/ef/core/>
10. Усачев В., Рагулин П. Концептуальная модель масштабируемого сервиса социальной сети, Молодой учёный №12 (116) июнь-2 2016 г. [Электронный ресурс] Режим доступа: <https://moluch.ru/archive/116/>
11. Authentication Using Google In ASP.NET Core 2.0 [Электронный ресурс] Режим доступа: URL: <https://www.c-sharpcorner.com/article/authentication-using-google-in-asp-net-core-2-0/>

ЛИТЕРАТУРА

1. Ньюмэн С. Создание микросервисов - СПб.: Питер, 2016, - 392 с
2. Техническое задание. Требования к содержанию и оформлению (ГОСТ 19.201-78). [Электронный ресурс] Режим доступа: URL: <http://internet-law.ru/gosts/gost/31884/>
3. Вишневская Т.И., Романова Т.Н. Методология программной инженерии: Мет. указания к выполнению лабораторных работ М.: Изд-во МГТУ им. Н.Э. Баумана, 2017.-58 с.
4. Научно-исследовательский Центр CALS - Методология функционального моделирования IDEF0: ИПК Издательство стандартов, 2000 – 75 с.
5. Богданов А., Корхов В., Мареев В. Архитектуры и топологии многопроцессорных вычислительных систем. Изд-во Интуит.Ру, 2013, -176 с.
6. Мартин Клеппман. Высоконагруженные приложения. Программирование, масштабирование, поддержка - СПб.: Питер, 2018.- 640с.

Вишневская Татьяна Ивановна,
Романова Татьяна Николаевна