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

Кафедра информатики и программирования

Разработка приложения для персонализированной агрегации контента

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студентки 4 курса 441 группы
направления 02.03.03 Математическое обеспечение и администрирование информационных систем
факультета компьютерных наук и информационных технологий
Ключниковой Полины Алексеевны

Научный руководитель:
д. т. н., доцент _____ А.С. Фалькович

подпись, дата

Зав. кафедрой:
к. ф. -м. н, доцент _____ М.В. Огнева

подпись, дата

Саратов 2026

ВВЕДЕНИЕ

Актуальность темы.

В современном мире объем цифрового контента растет опережающими темпами, что приводит к информационной перегрузке пользователей. Особенно остро эта проблема стоит в мессенджерах и социальных сетях, где информация поступает непрерывным потоком из множества разрозненных источников. Пользователь оказывается не в состоянии вручную отслеживать все интересующие его каналы и отделять значимые сообщения от шума. Это делает актуальной задачу автоматического сбора, фильтрации и структурирования контента, что реализуется в агрегаторах контента. Качество такого сервиса определяется не только точностью рекомендаций, но и надежностью, масштабируемостью и безопасностью платформы, что требует продуманных инженерных решений [1].

Существующие решения, такие как Feedly и Flipboard, не всегда доступны на территории России и не поддерживают добавление произвольных источников, таких как Telegram. Платформа «Дзен» доступна в РФ, но не позволяет пользователю самостоятельно добавлять внешние источники. В связи с этим разработка собственного агрегатора контента, который бы сочетал в себе гибкость, надежность и современные архитектурные подходы, является актуальной задачей [2–4].

Цель бакалаврской работы – спроектировать и реализовать серверную и клиентскую части приложения-агрегатора контента на основе микросервисной архитектуры, обеспечивающие регистрацию и аутентификацию пользователей, управление профилями и подписками, сбор пользовательских событий и доставку персонализированной ленты с интеграцией внешней рекомендательной подсистемы.

Задачи выпускной квалификационной работы:

1. Проанализировать архитектурные подходы к построению серверной части, включая выбор между монолитной и микросервисной архитектурой, применение шлюза как единой точки входа и способы

событийной интеграции сервисов.

2. Изучить способы аутентификации и авторизации в распределённых системах, основанные на токенах и централизованном контроле доступа на уровне шлюза.

3. Рассмотреть подходы к проектированию схемы базы данных и управлению её изменениями с помощью миграций в условиях изоляции данных по сервисам.

4. Изучить подходы к разработке кроссплатформенного клиентского приложения и управлению его состоянием.

5. Спроектировать декомпозицию системы на сервисы и определить контракты их взаимодействия.

6. Спроектировать схему базы данных, модель пользовательских событий и модель безопасности платформы.

7. Реализовать серверные сервисы, включая шлюз, сервис аутентификации, сервис пользователей, сервис подписок и сервис событий взаимодействий.

8. Реализовать клиентское приложение и его интеграцию с серверной частью через шлюз.

9. Реализовать инфраструктуру платформы, включая контейнеризацию, брокер сообщений и автоматизированные миграции базы данных.

10. Провести апробацию разработанного решения и проверку его работоспособности.

Методологические основы построения распределенных систем и микросервисной архитектуры представлены в работах С. Ньюмена [5], К. Ричардсона [6], М. Клеппмана [7] и М. Фаулера [8]. Вопросы безопасности в распределенных системах рассмотрены в обзоре А. Ханноусса и С. Яхиуша [9] и стандартах IETF (RFC 7519, RFC 8725) [10], [11]. Для реализации были использованы такие технологии, как Spring Boot, Spring WebFlux, Apache Kafka, PostgreSQL, а для клиентской части – Flutter и Dart [12–15].

Теоретическая значимость работы заключается в систематизации знаний о проектировании и реализации масштабируемых серверных систем на основе микросервисной архитектуры, а также в обобщении подходов к построению безопасных распределенных приложений с использованием событийной модели.

Практическая значимость работы заключается в создании работоспособного прототипа платформы для персонализированной агрегации контента. Приложение предоставляет пользователям возможности регистрации, управления подписками и сбора событий, а также может служить основой для дальнейшей разработки полноценного коммерческого продукта. Использованные архитектурные решения могут быть применены при построении аналогичных распределенных систем.

Структура и объём работы. Бакалаврская работа состоит из введения, четырех глав, заключения, списка использованных источников (35 наименований) и пяти приложений. Общий объем работы – 236 страниц, из них 86 страниц – основное содержание, включая 13 рисунков и 4 таблицы.

КРАТКОЕСОДЕРЖАНИЕ РАБОТЫ

Первый раздел «Анализ предметной области и существующих решений» посвящен изучению предметной области, анализу архитектурных подходов и выбору технологического стека для реализации проекта.

В разделе рассмотрены такие концепции, как агрегация контента и информационная перегрузка, а также существующие решения (Feedly, Flipboard, «Дзен»). Подробно анализируются монолитная и микросервисная архитектуры. Монолитная архитектура проста на начальном этапе, но с ростом системы становится трудно масштабируемой и сопровождаемой. Микросервисная архитектура, напротив, позволяет масштабировать отдельные компоненты и обеспечивает изоляцию отказов, что является решающим фактором для выбора. Для решения задачи выбрана микросервисная архитектура с декомпозицией по бизнес-возможностям [5].

В разделе рассматривается паттерн API Gateway и Backend for Frontend

(BFF). API Gateway служит единой точкой входа для клиентов, централизуя сквозные задачи, такие как аутентификация, ограничение частоты запросов и маршрутизация. Для реализации шлюза выбран реактивный стек Spring WebFlux, который позволяет эффективно обрабатывать большое количество соединений благодаря неблокирующей модели ввода-вывода [13].

Вопросы безопасности решаются с использованием JSON Web Tokens (JWT) с асимметричной подписью (RS256), что позволяет проверять подлинность токена без хранения состояния на сервере. Для надежной работы используется пара токенов: короткоживущий токен доступа и долгоживущий refresh-токен. Для защиты от подделки запросов внутри периметра применяется HMAC-подпись заголовков [10],[11],[16].

Для обеспечения слабой связанности сервисов выбрана асинхронная событийная интеграция через брокер Apache Kafka. События публикуются по паттерну transactional outbox, что гарантирует атомарность изменения состояния базы и публикации события. Такой подход повышает устойчивость системы к сбоям и позволяет масштабировать обработку событий [7], [17].

В области проектирования данных принят принцип «схема на сервис» (database-per-service) на едином экземпляре PostgreSQL, что обеспечивает логическую изоляцию данных при умеренных эксплуатационных затратах. Для управления схемами используется Liquibase. Таблица событий взаимодействий партиционируется по времени для повышения производительности и упрощения обслуживания [18–20].

Клиентская часть реализована как кроссплатформенное приложение на Flutter с использованием библиотеки Riverpod для управления состоянием и архитектурой, основанной на принципах чистой архитектуры. Это обеспечивает единую кодовую базу для нескольких платформ, высокую скорость разработки и удобство тестирования [12],[21],[22].

Второй раздел «Проектирование системы» посвящен детальной разработке проектных решений системы.

Выполнена декомпозиция системы на пять серверных компонентов:

API-шлюз, сервис аутентификации, сервис пользователей, сервис подписок и сервис взаимодействий. Такая декомпозиция позволяет независимо разрабатывать, развертывать и масштабировать каждый сервис.

Определены контракты взаимодействия через шлюз. Все внешние запросы проходят через единую точку входа, что скрывает внутреннюю топологию системы. Введена курсорная пагинация для коллекций, устойчивая к изменениям данных.

Спроектирована модель событий на основе Apache Kafka. Определены шесть основных топиков: регистрация пользователя, создание пользователя, пакет взаимодействий, изменение подписки, обновление рекомендаций и публикация контента. Ключом сообщений во всех топиках выбран идентификатор пользователя, что гарантирует порядок обработки событий. Для надежной публикации внедрен паттерн transactional outbox.

Разработана схема базы данных по принципу «схема на сервис». Каждый сервис владеет собственными таблицами. Для таблицы событий взаимодействий применено помесячное партиционирование. Управление изменениями схемы выполняется с помощью Liquibase.

Модель безопасности построена на JWT с асимметричной подписью, где сервис аутентификации выступает удостоверяющим центром, а шлюз — точкой принудительного применения политики. Доверие к шлюзу обеспечивается HMAC-подписью внутренних заголовков. Дополнительно реализован механизм черного списка отозванных токенов на основе Redis.

Клиентское приложение построено на Flutter с разделением на три слоя: представление, предметная область и данные, что позволяет изолировать бизнес-логику от деталей реализации. Управление состоянием возложено на Riverpod. Навигация реализована декларативно с централизованным перенаправлением.

Третий раздел «Реализация» описывает техническую реализацию всех компонентов системы.

Реализован реактивный API-шлюз на Spring WebFlux, который

выполняет маршрутизацию, аутентификацию, ограничение частоты запросов и обогащение запросов заголовками. В шлюзе реализован фильтр для проверки отзыва токенов и ограничения частоты запросов по фиксированному окну.

Сервис аутентификации реализован на синхронном стеке Spring Web и отвечает за регистрацию, подтверждение адреса электронной почты, вход и управление токенами. Пароли хешируются с помощью BCrypt, а токены JWT подписываются асимметрично. Реализована ротация refresh-токенов.

Сервис пользователей создает профили, реагируя на события регистрации, и обеспечивает прохождение онбординга. Сервис подписок интегрируется с платёжным провайдером YooKassa и реализует согласование статуса платежей.

Сервис взаимодействий принимает и буферизирует события от клиента для отправки в Kafka. Реализован механизм валидации пакетов и пакетной публикации.

Все сервисы используют transactional outbox для надежной публикации событий. Сквозная идентификация запросов обеспечивается передачей уникального идентификатора через все компоненты.

Клиентское приложение взаимодействует с сервером через HTTP-клиент Dio с перехватчиками для автоматической подстановки токенов и их прозрачного обновления. Реализована очередь запросов при обновлении токена для предотвращения множественных попыток.

Инфраструктура оркеструется с помощью Docker Compose. Используются PostgreSQL, Valkey (кеш), Kafka (брокер) и SeaweedFS (объектное хранилище). Миграции базы данных вынесены в отдельные одноразовые задачи.

Четвёртый раздел «Апробация и развертывание» описывает процесс тестирования, локального развертывания и проверки готовности системы к масштабированию.

В работе использовалась методология разработки через тестирование

(TDD). Модульные тесты проверяют отдельные компоненты в изоляции, а интеграционные тесты с использованием Testcontainers проверяют работу с реальной базой данных и брокером сообщений.

Проверены два ключевых сквозных сценария: регистрация, онбординг и взаимодействие с лентой, а также оплата подписки и активация премиум-доступа. Тесты подтвердили корректность работы механизмов идемпотентности и согласованности данных.

Локальное развертывание осуществляется с помощью docker-compose, что позволяет поднять все компоненты одной командой. Встроенные средства наблюдаемости (health-проверки, метрики и структурированное логирование) обеспечивают мониторинг состояния системы.

Архитектура спроектирована с учетом горизонтального масштабирования: сервисы не хранят состояние локально, а используют общие базы данных и кеш. Событийная модель с ключом по идентификатору пользователя позволяет параллельно обрабатывать события разных пользователей.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы спроектированы и реализованы серверная и клиентская части приложения-агрегатора контента.

В рамках исследования были проанализированы архитектурные подходы к построению распределенных систем, изучены способы аутентификации, проектирования баз данных и разработки кроссплатформенных приложений. Проведенный анализ позволил обосновать выбор микросервисной архитектуры с декомпозицией по бизнес-возможностям, что обеспечивает масштабируемость, изоляцию отказов и независимость разработки.

На этапе проектирования были определены состав и границы микросервисов, разработаны их контракты и модель событий. Спроектирована схема базы данных по принципу «схема на сервис» с партиционированием таблицы событий. Разработана модель безопасности на основе JWT и HMAC-подписи.

В результате разработки реализованы пять серверных компонентов: реактивный API-шлюз, сервис аутентификации, сервис пользователей, сервис подписок и сервис событий взаимодействий. Все сервисы используют событийную интеграцию через Kafka с надежной публикацией через transactional outbox.

Разработано кроссплатформенное клиентское приложение на Flutter, которое обеспечивает регистрацию, управление профилем, просмотр персонализированной ленты и взаимодействие с сервером.

Подготовлена инфраструктура локального развертывания, объединяющая все компоненты в воспроизводимое окружение с помощью Docker Compose.

Проведенная апробация подтвердила работоспособность всех сквозных сценариев, а спроектированная архитектура готова к горизонтальному масштабированию.

Поставленная цель работы достигнута, все задачи выполнены. Разработанное решение представляет собой рабочий прототип платформы для агрегации контента и может служить основой для создания полноценного коммерческого продукта.

Основные источники информации:

1. Feedly News Reader : [сайт] // Feedly Inc. 2026. URL: https://feedly.com/ (дата обращения: 06.06.2026).
2. How It Works // Flipboard. 2026. URL: https://about.flipboard.com/how-it-works/ (дата обращения: 06.06.2026).
3. Дзен : персональная лента рекомендаций : [сайт] // Яндекс. 2026. URL: https://dzen.ru/ (дата обращения: 06.06.2026).
4. Ньюмен С. Создание микросервисов / пер. с англ. С. Черникова. — 2-е изд. — СПб. : Питер, 2023. — 624 с. — (Библиотека программиста). — ISBN 978-5-4461-1145-9.
5. Ричардсон К. Микросервисы. Паттерны разработки и рефакторинга / пер. с англ. — СПб. : Питер, 2019. — 544 с. — (Библиотека программиста). — ISBN 978-5-4461-0996-8.
6. Fowler M. Microservices : a definition of this new architectural term : [сайт]. 2014. URL: https://martinfowler.com/articles/microservices.html (дата обращения: 27.10.2025).
7. Hannousse A., Yahiouche S. Securing microservices and microservice architectures : a systematic mapping study // Computer Science Review. 2021. Vol. 41. P. 100415. DOI: 10.1016/j.cosrev.2021.100415. URL: https://doi.org/10.1016/j.cosrev.2021.100415 (дата обращения: 27.10.2025).
8. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) : RFC 7519. — IETF, 2015. — URL: https://www.rfc-editor.org/rfc/rfc7519 (дата обращения: 30.01.2026).
9. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка / пер. с англ. — СПб. : Питер, 2018. — 640 с. — (Бестселлеры O’Reilly). — ISBN 978-5-4461-0512-0.
10. Fowler M. Patterns of Enterprise Application Architecture : [каталог пат-тернов]. 2002. URL: https://martinfowler.com/eaaCatalog/ (дата обраще-ния: 05.03.2026).

11. Richardson C. Pattern : API Gateway / Backends for Frontends // Microservices.io. 2024. URL: https://microservices.io/patterns/apigateway.html (дата обращения: 12.03.2026).
12. Spring Cloud Gateway Documentation // VMware (Broadcom). 2025. URL: https://docs.spring.io/spring-cloud-gateway/reference/ (дата обращения: 13.03.2026).
13. Newman S. Backends For Frontends : [сайт]. 2015. URL: https://samnewman.io/patterns/architectural/bff/ (дата обращения: 19.03.2026).
14. Web on Reactive Stack : Spring Framework Reference (WebFlux) // VMware (Broadcom). 2025. URL: https://docs.spring.io/spring-framework/reference/web/webflux.html (дата обращения: 20.03.2026).
15. Reactor Core Reference Guide // Project Reactor. 2025. URL: https://projectreactor.io/docs/core/release/reference/ (дата обращения: 23.03.2026).
16. RateLimiter : Resilience4j Documentation // Resilience4j. 2024. URL: https://resilience4j.readme.io/docs/ratelimiter (дата обращения: 23.03.2026).
17. JSON Web Token Best Current Practices : RFC 8725 / Sheffer Y., Hardt D., Jones M. — IETF, 2020. — URL: https://www.rfc-editor.org/rfc/rfc8725 (дата обращения: 03.04.2026).
18. Hardt D. The OAuth 2.0 Authorization Framework : RFC 6749. — IETF, 2012. — URL: https://www.rfc-editor.org/rfc/rfc6749 (дата обращения: 03.04.2026).
19. Lodderstedt T., Bradley J., Labunets A., Fett D. Best Current Practice for OAuth 2.0 Security : RFC 9700. — IETF, 2025. — URL: https://www.rfc-editor.org/rfc/rfc9700 (дата обращения: 03.04.2026).
20. Krawczyk H., Bellare M., Canetti R. HMAC : Keyed-Hashing for Message

Authentication : RFC 2104. — IETF, 1997. — URL: https://www.rfc-editor.org/rfc/rfc2104 (дата обращения: 03.04.2026).
21. Apache Kafka Documentation // Apache Software Foundation. 2025. URL: https://kafka.apache.org/documentation/ (дата обращения: 16.02.2026).
22. Bonér J., Klang V. Uncovering the Hidden Potential of Event-Driven Architecture : a Research Agenda // arXiv preprint. 2023. arXiv:2308.05270. URL: https://arxiv.org/abs/2308.05270 (дата обращения: 03.04.2026).
23. Richardson C. Pattern : Transactional outbox // Microservices.io. 2024. URL: https://microservices.io/patterns/data/transactional-outbox.html (дата обращения: 10.04.2026).
24. Richardson C. Pattern : Saga // Microservices.io. 2024. URL: https://microservices.io/patterns/data/saga.html (дата обращения: 10.04.2026).
25. Richardson C. Pattern : Database per service // Microservices.io. 2024. URL: https://microservices.io/patterns/data/database-per-service.html (дата обращения: 10.04.2026).
26. Liquibase Documentation // Liquibase Inc. 2025. URL: https://docs.liquibase.com/ (дата обращения: 10.04.2026).
27. PostgreSQL 17 Documentation. Chapter 5.12. Table Partitioning // PostgreSQL Global Development Group. 2024. URL: https://www.postgresql.org/docs/current/ddl-partitioning.html (дата обра-щения: 10.04.2026).
28. Flutter Documentation // Google. 2025. URL: https://docs.flutter.dev/ (дата обращения: 28.02.2026).
29. Dart Language Documentation // Google. 2025. URL: https://dart.dev/guides (дата обращения: 10.04.2026).
30. State Management Approaches : Flutter Documentation // Google. 2025. URL: https://docs.flutter.dev/data-and-backend/state-mgmt/options (дата обращения: 10.04.2026).
31. Riverpod Documentation // Remi Rousselet. 2025. URL:

https://riverpod.dev/docs/introduction/why_riverpod (дата обращения: 10.04.2026).

32. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения / пер. с англ. — СПб. : Питер, 2018. — 352 с. — (Библиотека программиста). — ISBN 978-5-4461-0772-8.
33. Docker Compose Documentation // Docker Inc. 2025. URL: https://docs.docker.com/compose/ (дата обращения: 30.05.2026).
34. Testcontainers Documentation // AtomicJar (Docker Inc.). 2025. URL: https://java.testcontainers.org/ (дата обращения: 30.05.2026).
35. Spring Boot Reference Documentation // VMware (Broadcom). 2025. URL: https://docs.spring.io/spring-boot/index.html (дата обращения: 30.05.2026).