УДК 004.774:347.783

ОСОБЕННОСТИ ПОСТРОЕНИЯ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ СОЗДАНИЯ СОВМЕСТНЫХ ФОТОАЛЬБОМОВ

Глушко А.Ю.

Белорусский государственный университет информатики и радиоэлектроники, г. Минск, Республика Беларусь

Научный руководитель: Карпович Е.Б. – магистр техники и технологии, ст. преподаватель кафедры ИПиЭ

Аннотация. Исследованы подходы к построению микросервисной архитектуры веб-приложения для создания совместных фотоальбомов. Рассмотрены особенности применения гибридной модели, сочетающей реактивные и синхронные (блокирующие) компоненты для разделения ответственности между слоями системы. Предложена архитектурная схема с централизованной аутентификацией и изоляцией микросервисов, где доменная логика отделена от механизмов безопасности. Продемонстрированы преимущества предложенной архитектуры.

Ключевые слова: микросервисная архитектура, реактивное программирование, Spring WebFlux, Spring Cloud Gateway, Netflix Eureka, Service Discovery, JWT, объектное хранилище, SeaweedFS

Введение. Рост популярности сервисов для совместной работы с медиаконтентом требует систем, способных обрабатывать тысячи параллельных соединений при загрузке тяжёлых данных. Традиционные синхронные архитектуры (Thread-per-request) при таких нагрузках сталкиваются с проблемой исчерпания пула потоков [1]. Основная сложность заключается в поиске баланса: необходимо обеспечить быструю авторизацию каждого запроса и эффективную передачу данных, не перегружая внутреннюю бизнес-логику.

Целью данной работы является исследование гибридной микросервисной модели, где реактивный сервис системы берёт на себя функции безопасности и управления трафиком, а синхронные (блокирующие) сервисы отвечают за сложную доменную логику.

Основная часть. Выбор архитектуры серверной части веб-приложения для создания совместных фотоальбомов состоял из классической монолитной и современной микросервисной архитектуры. При проектировании систем, работающих с большим объемом бинарных данных и множеством одновременных пользовательских сессий, данные подходы демонстрируют принципиально разные показатели производительности и масштабируемости.

Монолитная архитектура предполагает объединение логики управления пользователями, обработки альбомов и модулей загрузки контента в рамках единого процесса. Несмотря на простоту разработки на начальных этапах, такая структура затрудняет независимое масштабирование наиболее нагруженных узлов. Микросервисная архитектура, напротив, позволяет выделить критически важные компоненты в изолированные сервисы, что обеспечивает гибкость в выборе стека технологий для каждой конкретной задачи [2].

Сравнив указанные подходы, выбор был сделан в пользу микросервисной архитектуры, реализованной в виде трех функциональных модулей: двух сервисов с блокирующим вызовом (управление пользователями и управление альбомами) и реактивного API-шлюза, который одновременно является центром аутентификации. Данная декомпозиция позволяет изолировать доменную логику от механизмов обеспечения безопасности и маршрутизации трафика.

Центральным звеном архитектуры является реактивный API-шлюз, совмещающий в себе функции единой точки входа и единого центра аутентификации. Реактивный API-шлюз реализован на базе Spring Cloud Gateway с использованием фреймворка Spring WebFlux, который построен на принципах реактивного программирования и использует неблокирующий I/O. Это позволяет обрабатывать большое количество одновременных

соединений с минимальным потреблением ресурсов потоков. Остальные микросервисы реализованы на базе Spring MVC, что соответствует их назначению: синхронная блокирующая модель упрощает разработку сложной бизнес-логики и естественно интегрируется с реляционными базами данных. Таким образом, реактивность сосредоточена на уровне шлюза, где она даёт наибольший эффект при обработке входящего потока запросов.

Сервисы с блокирующим вызовом настроены на доверие исключительно токенам, выпущенным централизованной подсистемой авторизации API-шлюза. Прямое взаимодействие между ними без участия этой подсистемы блокируется, что исключает риск компрометации данных при несанкционированном доступе к одному из внутренних узлов.

Важной особенностью спроектированной архитектуры является использование паттерна Service Discovery, реализованного на базе Netflix Eureka. В условиях динамического масштабирования, когда экземпляры сервисов могут менять свои сетевые адреса, Eureka выступает в роли центрального реестра. Это позволяет API-шлюзу автоматически обнаруживать доступные узлы сервисов пользователей и альбомов, обеспечивая балансировку нагрузки и высокую отказоустойчивость системы без необходимости жесткой привязки IP-адресов в конфигурации.

Для оптимизации работы с медиаконтентом в системе реализовано разделение потоков управления и потоков данных. Вместо ресурсоемкого хранения изображений в базе данных используется S3-совместимое объектное хранилище SeaweedFS.

Хотя само хранилище отдает файлы по прямым идентификаторам, эти идентификаторы представляют собой высокоэнтропийные непредсказуемые последовательности – UUID, а получение списка таких идентификаторов возможно только через API-шлюз после успешной верификации JWT-токена. Таким образом, даже при наличии публичного эндпоинта хранилища, злоумышленник не может получить доступ к медиаконтенту, поскольку база данных, где сопоставлены данные пользователя и ссылки на медиаконтент, надежно защищены шлюзом с JWT. Также необходимо упомянуть, что благодаря тому, что API-шлюз при обработке запроса возвращает фронтенд-приложению прямые идентификаторы на объекты в бакетах S3-совместимого хранилища, нагрузка по передаче бинарного трафика с серверной части значительно снижается: фронтенд самостоятельно подтягивает изображения из хранилища, используя полученные адреса.

Архитектурная схема с централизованной аутентификацией и изоляцией микросервисов представлена на рисунке 1.

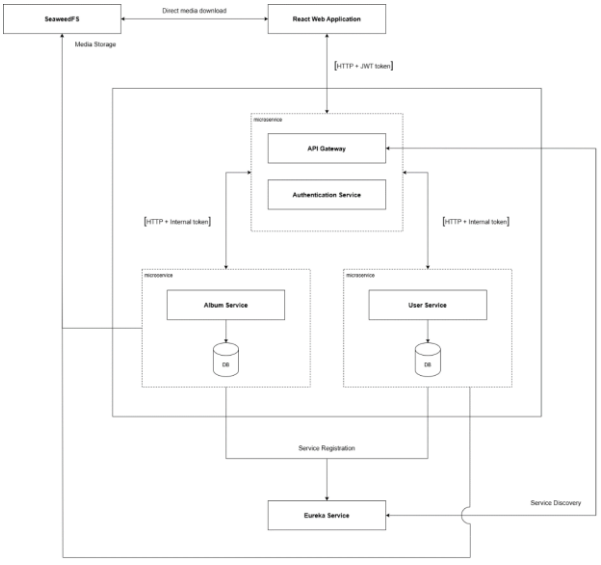

Рисунок 1 – Структурная схема веб-приложения для создания совместных фотоальбомов

Предложенная архитектура демонстрирует следующие преимущества:

1 повышенная безопасность: изоляция микросервисов с использованием внутренних токенов предотвращает цепочечную компрометацию системы при атаке на отдельный узел. По данным IBM, среднее время локализации инцидента на рынке составляет 75 дней – архитектурная изоляция позволяет сократить этот срок за счёт мгновенного отключения скомпрометированного сервиса без остановки всей системы [3].

2 гибкость масштабирования компонентов системы: каждый микросервис может развиваться независимо от других. По данным Global Market Insights, 86% организаций отмечают, что микросервисы действительно улучшают масштабируемость систем [4].

3 снижение деградации БД: вынос медиаконтента в S3-хранилище предотвращает раздувание страниц данных и индексов. Хранение объектов >1 МБ вне БД исключает загрязнение буферного пула крупными бинарными данными, сохраняя высокую скорость выполнения транзакционных запросов [5].

Заключение. В рамках данной работы исследована гибридная микросервисная архитектура веб-приложения для создания совместных фотоальбомов. Предложенная архитектура системы включает реактивный API-шлюз на базе Spring Cloud Gateway и два сервиса с блокирующим вызовом на Spring MVC, обеспечивающих управление пользователями и фотоальбомами.

Применение паттерна Service Discovery на базе Netflix Eureka обеспечивает динамическое обнаружение сервисов и автоматическую балансировку нагрузки. Централизация логики безопасности в API-шлюзе позволяет единообразно управлять аутентификацией для всех компонентов системы.

Гибридный подход подтверждает возможность оптимального сочетания реактивного программирования для задач маршрутизации и аутентификации с синхронной моделью для реализации сложной доменной логики.

Список литературы

1. *Why Reactive? Thread per Request vs. Reactive Programming Model (Eventloop) [Электронный ресурс]. – Режим доступа: https://medium.com/walmartglobaltech/thread-per-request-vs-7bf1f22f590. – Дата доступа: 09.02.2026.*

2. *Monolith to Microservices. Evolutionary Patterns to Transform Your Monolith / S. Newman – Sebastopol : O'Reilly Media, Inc., 2019. –255 p. – ISBN 978-1-492-04784-1.*

3. *Monoliths to Microservices: 4 Modernization Best Practices [Электронный ресурс]. – Режим доступа: https://vfunction.com/blog/monoliths-to-microservices-4-modernization-best-practices/. – Дата доступа: 20.02.2026.*

4. *Cloud Microservices Market Size & Share 2025 – 2034 [Электронный ресурс]. – Режим доступа: https://www.gminsights.com/industry-analysis/cloud-microservices-market. – Дата доступа: 20.02.2026.*

5. *To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem / Russell Sears / Catharine van Ingen1 / Jim Gray – Microsoft Corporation, 2006. – 11 p.*

UDC 004.774:347.783

FEATURES OF BUILDING A MICROSERVICE ARCHITECTURE FOR A COLLABORATIVE PHOTO ALBUMS WEB APPLICATION

Hlushko H.Y.

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Karpovich E.B. - master of engineering and technology, senior lecturer of the Department of EPE

Annotation. Approaches to building a microservice architecture for a collaborative photo albums web application have been studied. The features of applying a hybrid model that combines reactive and synchronous components to separate responsibilities between system layers are considered. An architectural scheme with centralized authentication and microservice isolation is proposed, where domain logic is separated from security mechanisms. The advantages of the proposed architecture are demonstrated.

Keywords: microservice architecture, reactive programming, Spring WebFlux, Spring Cloud Gateway, Netflix Eureka, Service Discovery, JWT, object storage, SeaweedFS