DOI 10.37882/2223-2966.2025.04-2.12

УСИЛЕНИЕ ЗАЩИТЫ ПОЛЬЗОВАТЕЛЕЙ ПРОГРАММНЫХ МЕССЕНДЖЕРОВ НА ОСНОВЕ ОРКЕСТРУЕМОЙ СИСТЕМЫ МНОГОУРОВНЕВОЙ ФИЛЬТРАЦИИ

ENHANCING THE SECURITY OF SOFTWARE MESSENGERS USERS BASED ON THE ORCHESTRATED MULTI-LEVEL FILTERING SYSTEM

I. Demin
M. Rysin

Summary. The article presents an architectural approach to creating a messenger with increased security based on multilevel filtering (MLFS). The need for additional processing of incoming requests in addition to traditional measures such as encryption and two-factor authentication is justified. The MLFS concept and ways of its optimization are described. A micro-service architecture is proposed that implements this solution, with an emphasis on adaptability and risk reduction.

Keywords: messenger, information security, cybersecurity, microservice, API Gateway, orchestrator, multilevel filtering, MLFS.

Демин Иван Александрович
МИРЭА — Российский технологический университет
demin.i.a2@edu.mirea.ru
Рысин Михаил Леонидович
кандидат педагогических наук, доцент,
МИРЭА — Российский технологический университет
rysin@mirea.ru

Аннотация. В статье представлен архитектурный подход к созданию мессенджера с повышенной безопасностью на основе многоуровневой фильтрации (MLFS). Обоснована необходимость дополнительной обработки входящих запросов помимо традиционных мер защиты, таких как шифрование и двухфакторная аутентификация. Описана концепция MLFS и пути её оптимизации. Предложена микросервисная архитектура, реализующая данное решение, с акцентом на адаптивность и снижение рисков.

Ключевые слова: мессенджер, информационная безопасность, кибербезопасность, микросервис, API Gateway, оркестратор, многоуровневая фильтрация, MLFS.

Введение

Мобильные мессенджеры являются одним из главных инструментов общения в цифровую эпоху. Они позволяют людям обмениваться сообщениями, файлами, совершать звонки и взаимодействовать в группах. С увеличением количества пользователей мессенджеров возрастает и число угроз, связанных не только с технологическими уязвимостями, но и с человеческим фактором.

Современные технологии безопасности, такие как шифрование сообщений и двухфакторная аутентификация, эффективно защищают данные на техническом уровне, но не предотвращают случаи мошенничества, социальной инженерии и других манипуляций, основанных на невнимательности или доверчивости пользователей. Эти проблемы требуют дополнительных механизмов защиты, ориентированных на анализ поведения и предотвращение угроз в реальном времени.

Проблемы современной безопасности в мессенджерах

Современные мессенджеры решают большое количество задач в плане безопасности — от шифрования сообщений и паролей, до систем автоматических блокировок аккаунтов на основе подозрительной активности [1, 2]. Несмотря на наличие указанных механизмов, остаются угрозы, которые либо не отслеживаются в реальном времени, либо обрабатываются только спустя некоторое время:

- мошенничество через мессенджеры — злоумышленники могут выдавать себя за доверенных лиц, не вызывая подозрений у системы;
- социальная инженерия — мошенники манипулируют пользователями, заставляя их передавать личные данные;
- аномальная активность пользователей — злоумышленники могут массово рассылать сообщения или добавлять пользователей в контакты для последующих атак.

Для решения этих проблем предлагается внедрение механизма фильтрации угроз на основе подхода многоуровневой системы фильтрации (Multi-Layer Filtering System, MLFS), которая может быть адаптирована разработчиками в зависимости от потребностей системы и уровня безопасности.

Концепция многоуровневой системы фильтрации (MLFS)

MLFS — система на основе построения цепочек фильтрации пользовательских запросов на уровне их обработки специализированными сервисами. Эта идея

реализуется антифрод-сервисами (от англ. anti-fraud «борьба с мошенничеством»), которые внедряются в систему и отслеживают пользовательскую активность в фоне. Цепочки фильтрации представляют собой прослойку между пользователем и слоем бизнес-логики сервиса, типовая схема ее использования изображена на рис. 1.

MLFS может быть построена на архитектуре с использованием паттерна «API Gateway» с основным контролирующим элементом (единой точкой входа), через который проходят все входящие запросы. Прежде чем запрос перейдет в сервис бизнес-логики, он проходят через набор фильтров, которые могут быть настроены как последовательными (для поэтапной проверки), так и параллельными (например, логирование, сбор аналитики).

Фильтры могут быть сгруппированы как монолитом, так и разбиты на отдельные микросервисы [3]. Второй вариант предпочтителен, так как обеспечивает четкое разграничение функциональности, но допускается и монолитный подход, если это необходимо по архитектурным причинам.

Предлагаемые уровни фильтрации:

1. Аналитика активности пользователей — отслеживание действий пользователей для выявления аномального поведения.
2. Система черных меток — механизм репутации пользователей на основе жалоб и автоматических проверок, позволяющий выявлять мошенников и злоумышленников.
3. Гибкие настройки доступа к данным другого пользователя — возможность ограничения взаимодействий в зависимости настроек, установленных другим пользователем.

Определение требований к фильтрам

Усложнение механизма защиты может сказаться на скорости работы системы: при использовании рекомендуемого подхода по разбиению фильтров возможны существенные затраты времени на обращение в сервис, поиск необходимой информации и обработку данных. Поэтому необходимо использовать следующие средства:

- Выбор подходящих баз данных:
  - Elasticsearch — для логирования и аналитики;
  - Redis — для кэширования временных данных и быстрого доступа к информации (должен использоваться всегда для улучшения быстродействия системы).
- Кэширование на всех уровнях:
  - Redis — как основной способ кэширования.
  - CDN — для ускорения загрузки профилей и статических данных.
- Оптимизация межсервисного взаимодействия:
  - gRPC — для уменьшения скорости общения между оркестратором и фильтром.

Определим базовые требования, которым должны следовать наши фильтры:

1. Каждый фильтр должен отвечать принципу Single Responsibility — это значит, что фильтр должен реализовывать только необходимую для его работы логику [4].
2. Каждый фильтр должен реализовывать кэширование — используется для ускорения получения данных, в качестве используемых технологий предлагается использовать Redis или Memcached [5].
3. Фильтр не должен обращаться к другим сервисам для получения данных — чтобы сделать фильтр максимально легковесным.
4. Персистентные сущности должны содержать минимум информации, для наибольшей скорости работы базы данных.
5. Межсервисное взаимодействие должно реализовываться с использованием gRPC [6, 7].
6. Использование паттерна API Gateway, для упрощения получения и обработки пользовательских запросов [8, 9].

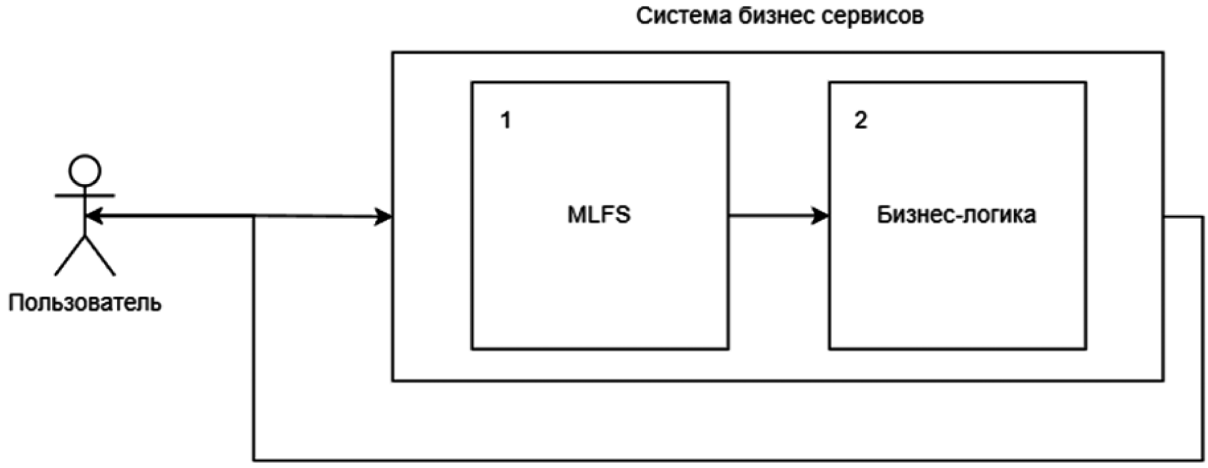

Рис. 1. Схема использования MLSF

7. Последовательность работы фильтров должна координироваться паттерном «Оркестратор» [10].

Такой подход позволяет:

- обеспечить четкую логику обработки запросов, передавая их через CMLFS в строго определенном порядке;
- контролировать цепочку вызовов сервисов и управлять передачей данных между ними.

Практические аспекты реализации системы

На рис. 2 представлен пример того, как может быть реализована схема системы фильтров в мессенджере, с учётом описанных выше принципов и требований.

Здесь API Gateway выполняет роль оркестратора и управляет маршрутизацией запросов, направляя их в систему фильтрации перед передачей в микросервисы мессенджера.

Компонент CMLFS контроля отправки сообщений проверяет, может ли пользователь отправить сообщение, его диаграмма классов представлена на рис. 3. Данные о правилах доступа загружаются из реляционной СУБД PostgreSQL и кэшируются в NoSQL СУБД Redis.

Компонент CMLFS контроля отправки файлов проверяет возможность отправки и скачивания файлов с учетом установленных ограничений, его диаграмма классов представлена на рис. 4.

CMLFS черных меток управляет базой данных пользователей с плохой репутацией, его диаграмма классов представлена на рисунке 4. Если пользователь имеет большое количество жалоб, его функционал может быть ограничен.

Сервис управления аккаунтами обрабатывает изменения в настройках пользователей и обновляет правила в базе данных CMLFS.

Сервисы сообщений, файлов и аутентификации обеспечивают базовую логику работы мессенджера после прохождения всех фильтров безопасности.

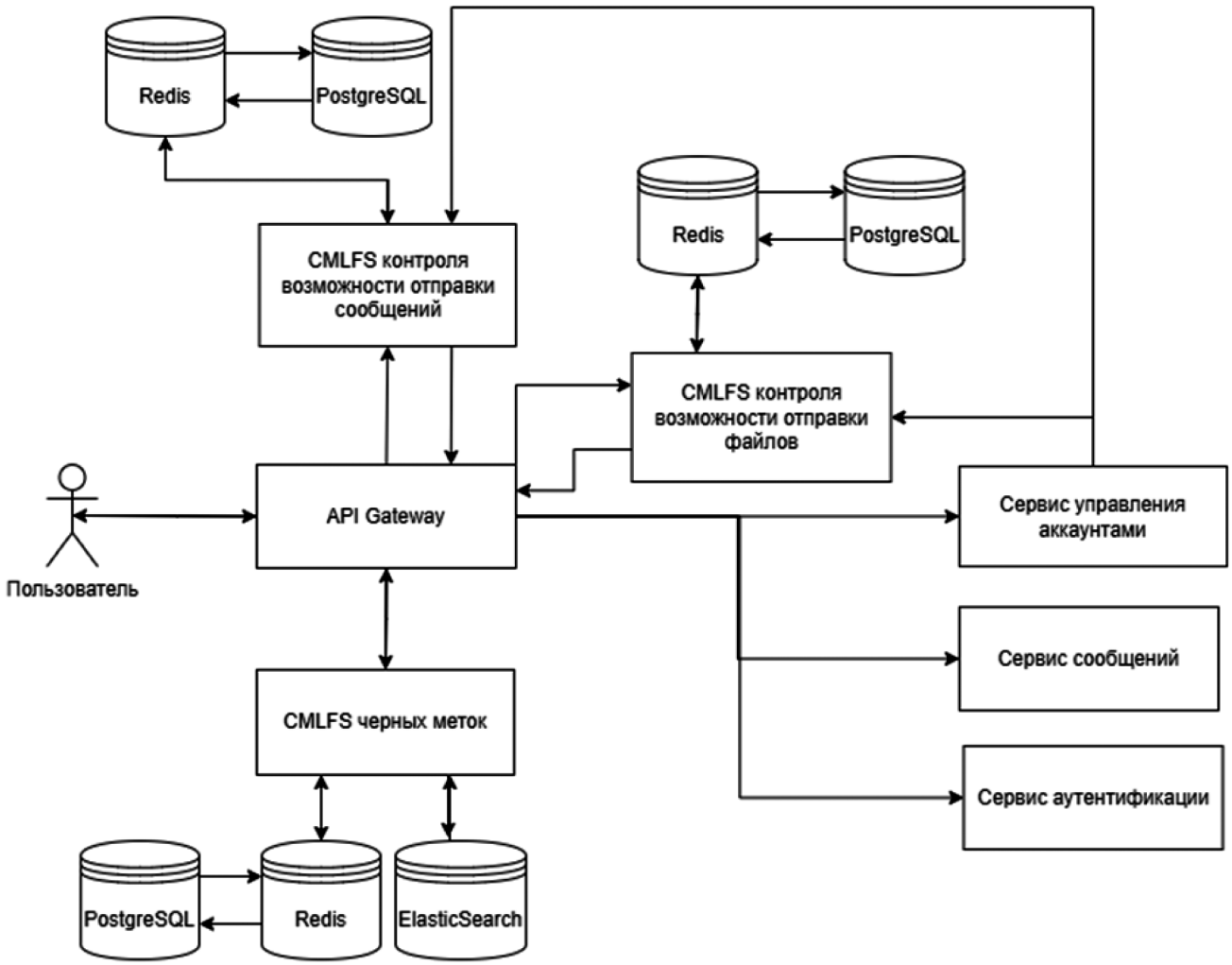

Рис. 2. Схема использования системы фильтров

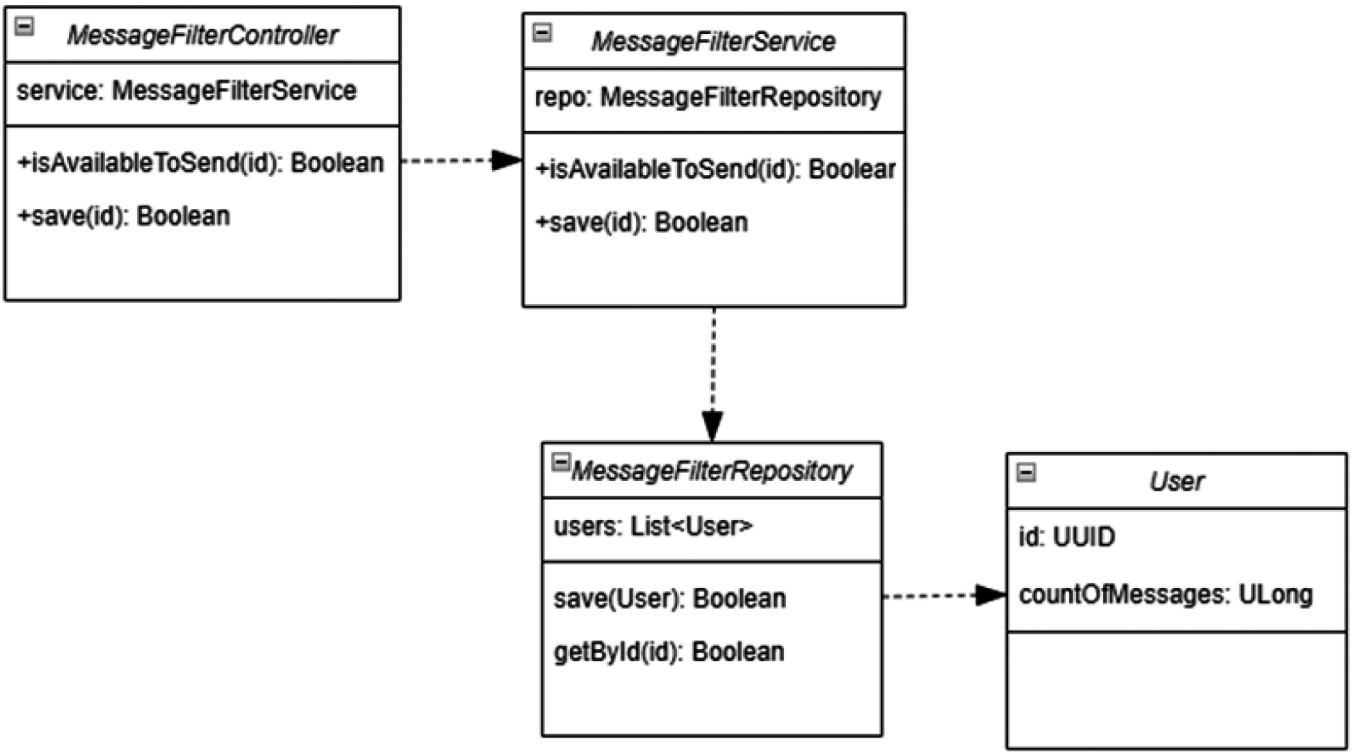

Рис. 3. Диаграмма классов сервиса MessageFilter

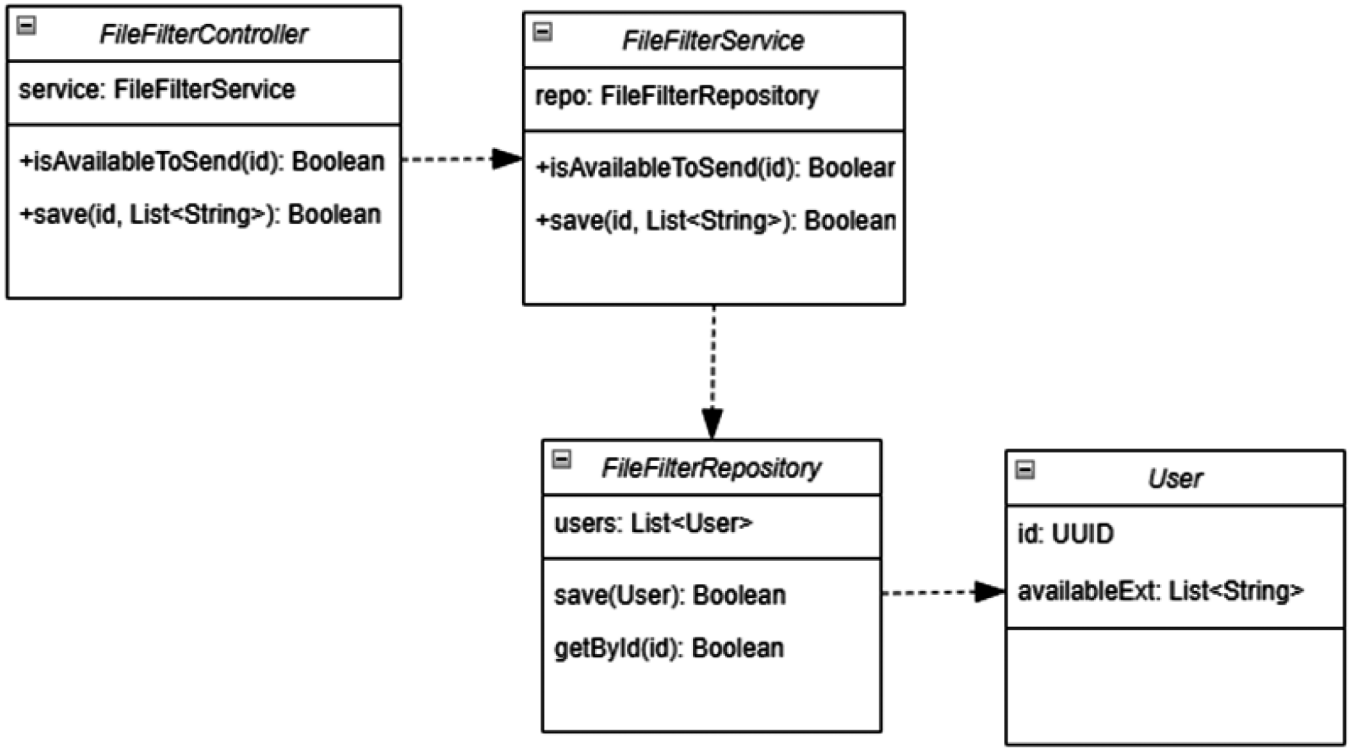

Рис. 4. Диаграмма классов сервиса FileFilter

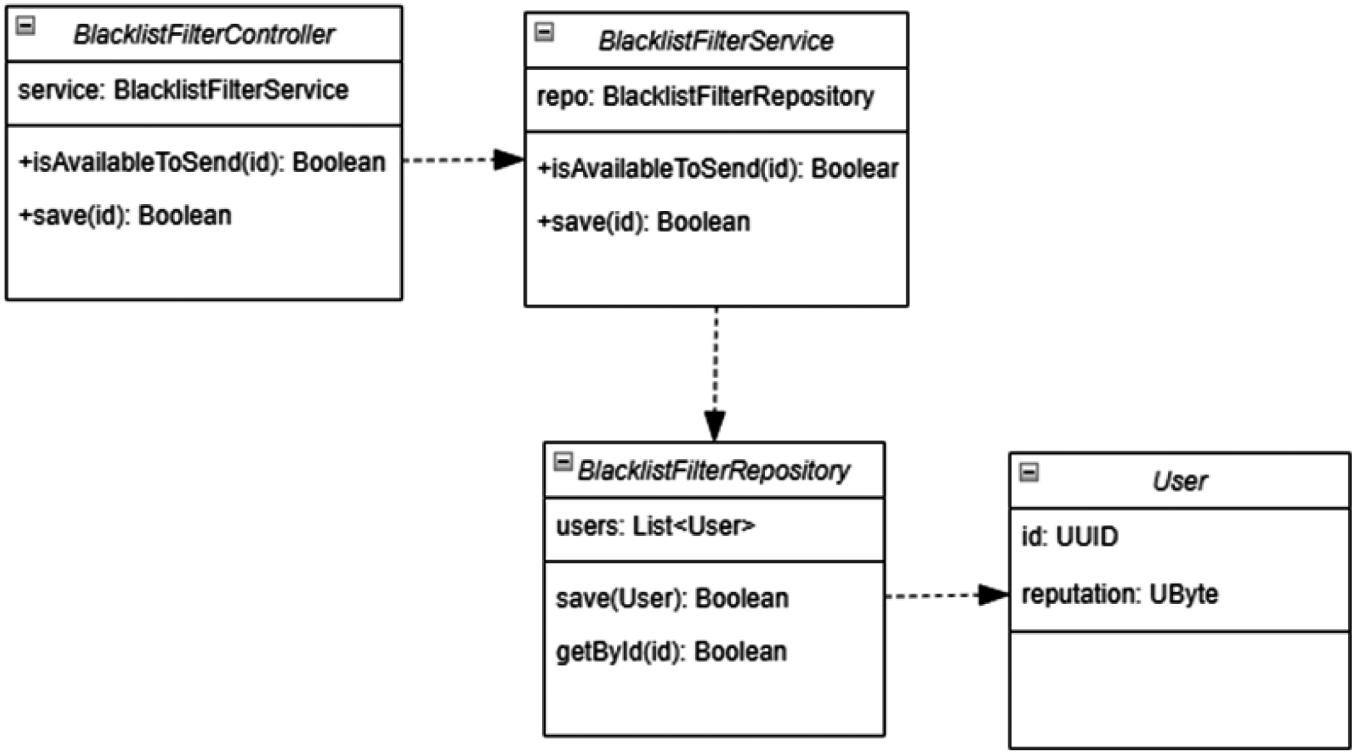

Рис. 5. Диаграмма классов сервиса BlacklistFilter

Листинг 1.

Пример реализации паттерна «оркестратор»

```
@RestController
@RequestMapping(«/api/messages»)
class MessageOrchestratorController(
 private val messageFilterService: MessageFilterService,
 private val fileFilterService: FileFilterService,
 private val blackListFilterService: BlackListFilterService,
 private val businessLogicService: BusinessLogicService,
) {
 @PostMapping(«/send»)
 @CircuitBreaker(name = «messageService», fallbackMethod = «fallbackSendMessage»)
 fun sendMessage(@RequestBody request: MessageRequest): Response<String> {
 if (!messageFilterService.isAvailableToSend(request.senderId, request.receiverId)) {
 return ErrorResponse(HttpStatus.FORBIDDEN, «The user has disabled the ability to send him messages from unknown
users»)}
 if (request.file != null && !fileFilterService.isAvailableToSend(request.senderId, request.file)) {
 return ErrorResponse(
 HttpStatus.BAD_REQUEST,
 «The user disabled the ability to send files with this permission.»
 )
 }
 if (!blackListFilterService.isAvailableToSend(request.senderId)) {
 return ErrorResponse(HttpStatus.TOO_MANY_REQUESTS, «Too many requests»)
 }
 businessLogicService.sendMessage(request.message)
 return SuccessResponse(«success»)
 }
}
```

Оркестрация реализована нами во фреймворке Spring на языке Kotlin. Код представлен в листинге 1.

Здесь класс MessageOrchestratorController — это контроллер, реализующий паттерн «оркестратор» для обработки отправки сообщений. Он координирует последовательную работу фильтров, проверяя каждое входящее сообщение перед передачей в основной бизнес-сервис.

Контроллер обрабатывает POST-запрос на эндпоинт /api/messages/send, получая от клиента объект MessageRequest.

1. Далее выполняются три ключевые проверки:
2. MessageFilterService — проверяет, может ли отправитель отправить сообщение получателю, в том числе с учётом настроек приватности (например, запрет на сообщения от незнакомых пользователей).
3. FileFilterService — если сообщение содержит файл, фильтр проверяет его допустимость по типу, размеру или настройкам отправителя.
4. BlackListFilterService — проверяет, не заблокирован ли пользователь системой (например, за подозрительную активность или превышение лимита).

В ходе последующего тестирования отключим возможность пользователю отправлять сообщения от неизвестных. Результат виден на рис. 6, время запроса составило 22 мс (по кэшируемым значениям в Redis).

Затем проведем тестирование отправки файлов с запрещенным расширением. На рис. 7 видно, что запрос выполнился за 37 мс, причём половина времени ушла на запрос в сервис MessageFilter, а оставшееся время было потрачено на сервис FileFilter.

Для проверки работы последнего фильтра необходимо совершить несколько сотен запросов от лица одного пользователя, чтобы система запретила отправку сообщений на несколько минут. В результате увидим ответ от сервиса по примеру рис. 8. Половина времени работы из 54 мс тратится на поход в сервисы фильтрации сообщений и файлов.

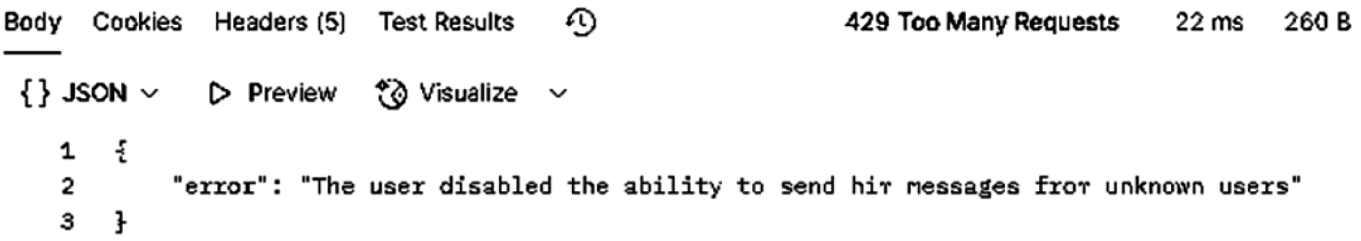

Рис. 6. Ошибка отправки от неизвестного пользователя

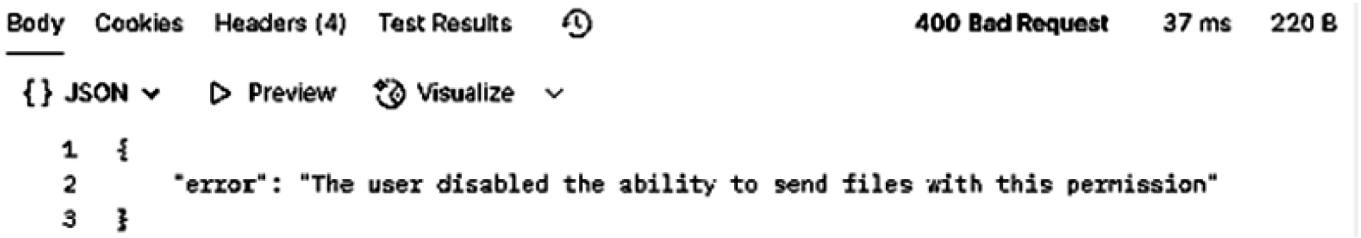

Рис. 7. Ошибка отправки файла с запрещенным разрешением

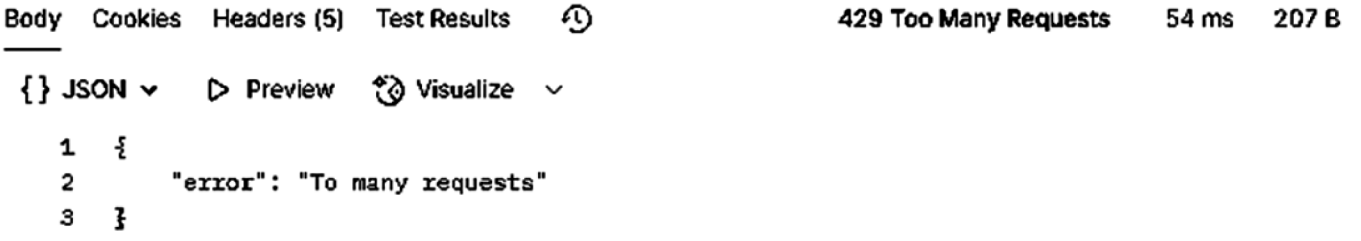

Рис. 8. Ошибка отправки сообщения из-за спама

Рис. 9. Запрос, прошедший всю цепочку фильтров

В итоге, если запрос проходит всю цепочку фильтров, то за время порядка 240 мс он попадает в бизнес-слой программного мессенджера (рис. 9).

Заключение

В результате применения концепции многоуровневой системы фильтрации реализована микросервисная архитектура мессенджера, позволяющая существенно усилить уровень защиты пользователя: ограничить взаимодействие с неизвестными пользователями (для предотвращения угроз мошенничества и социальной инженерии) в сочетании фильтром передаваемого содержимого и контролем аномальной активности (от угрозы рассылок спама).

При этом несмотря на внедрение цепочки новых фильтров и, как следствие, усложнения структуры программной системы, удалось достичь приемлемых показателей временных задержек.

ЛИТЕРАТУРА

1. Анализ механизмов защиты информации в микросервисных архитектурах [Электронный ресурс]. URL: https://habr.com/ru/articles/669816/ (дата обращения 15.02.2025).
2. DNS и NGFW: многоуровневая система защиты [Электронный ресурс]. URL: https://www.it-world.ru/security/ttzq8vsqi3kgs0cg4w0o0kokw8kowwg.html (дата обращения 15.02.2025).
3. Микросервисы vs монолит: разница архитектур и руководство по переходу [Электронный ресурс]. URL: https://worksolutions.ru/blog/mikroservisy-vs-monolit-raznicza-arhitektur-i-rukovodstvo-po-perehodu/ (дата обращения 03.03.2025).
4. Основные принципы разработки [Электронный ресурс]. URL: https://habr.com/ru/articles/810941/ (дата обращения 01.03.2025).
5. Memcached vs Redis [Электронный ресурс]. URL: https://dev.to/faangmaster/memcached-vs-redis-3b9m (дата обращения 27.02.2025).
6. Микросервисы: почему именно gRPC? [Электронный ресурс]. URL: https://habr.com/ru/articles/853392/ (дата обращения 28.02.2025).
7. Проектирование архитектуры для микросервисов с использованием gRPC [Электронный ресурс]. URL: https://habr.com/ru/companies/otus/articles/767724/ (дата обращения 19.02.2025).
8. Что такое Gateway API [Электронный ресурс]. URL: https://community.exolve.ru/blog/chto-takoe-gateway-api/ (дата обращения 25.02.2025).
9. Миграция микросервисной архитектуры на API Gateway [Электронный ресурс]. URL: https://habr.com/ru/articles/765944/ (дата обращения 03.03.2025).
10. Что такое оркестрация и хореография и почему это так важно для микросервисов [Электронный ресурс]. URL: https://babok-school.ru/blog/orchestration-and-choreography-of-microservices-in-eda/?utm_source=chatgpt.com (дата обращения 05.03.2025).

Журнал «Современная наука: актуальные проблемы теории и практики»