Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 24.М71-мм

Переход ECS от нерективного монолита к реактивному монолиту.

Юсуп Амин Турмуди

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
доцент кафедры системного программирования, к. Ф.-м. н.доцент Д. В. Луцив

Санкт-Петербург
2025

Оглавление

Введение

Вместе со стремительным развитием технологий появилось множество новых бизнес-моделей, которые обусловлены ростом числа пользователей смартфонов. Не стали исключением и старые бизнесмены, которые изначально делали ставку только на нетехнологичные виды бизнеса, а теперь стали развиваться на основе технологий или онлайн. Например, бизнес супермаркетов, в которых изначально продажи можно было осуществлять только придя непосредственно на место, теперь же продажи можно осуществлять через интернет или смартфон. Помимо того, что это облегчает жизнь покупателям, это также облегчает разработчикам бизнеса расширение рынка, только с помощью смартфона продавец может получить желаемый товар, не посещая супермаркет.

С появлением множества подобных предприятий, которые развивают свой бизнес с помощью технологий, руководители предприятий должны уметь конкурировать с подобными бизнес-конкурентами. Начиная с использования технологий, которые влияют на производительность, скорость, надежность системы в обеспечении удовлетворенности клиентов, чтобы бизнес мог конкурировать и хорошо работать. Если смотреть со стороны системы, то конкуренция проявляется в виде удовлетворенности пользователей услугами, предоставляемыми системой, с точки зрения скорости, точности и простоты использования услуги.

Возможно, система будет работать быстро, пока пользователей немного, но со временем их станет больше. С увеличением количества пользователей это повлияет на производительность системы. Вот что необходимо учитывать, если количество пользователей начнет расти, сможет ли система обрабатывать запросы, поступающие одновременно, и будет ли система хорошо справляться с запросами, если их будет много, и будет ли система обеспечивать ту же скорость. Это некоторые факторы, которые должны быть учтены при разработке архитектуры программного обеспечения, как сделать так, чтобы система по-прежнему предоставляла пользователям лучший сервис, чтобы пользователи не переключились на аналогичный бизнес.

Для разведения бизнеса приоритетом является запуск системы, а не ее производительность, она призвана быстро вывести бизнес на рынок. Но мы, как архитекторы программного обеспечения, должны думать наперед, что если пользователей системы уже много. Возможно, это еще можно контролировать, масштабируя систему по горизонтали. Но это будет сиюминутным решением, мы должны подумать о постоянном решении, чтобы преодолеть блокировку, которая возникает в традиционных системах, изменив код на реактивный.

«В традиционных системах многие пользователи могут обращаться к системе одновременно с помощью многопоточного метода» [5], но если в системе возникает блокировка, например, при запросе к базе данных или вызове внешнего API[1], то используемый поток замирает, и другие запросы ждут, пока поток снова освободится, что приводит к снижению производительности системы в плане скорости, задержке ответа. Поэтому появилось реактивное программирование, которое изменило синхронную систему на асинхронную (неблокирующую).

Реактивность — не новая концепция в мире программирования, но она становится все более популярной в сегодняшнюю эпоху из-за появления архитектуры микросервисов, которая позволяет разделить систему на несколько частей, чтобы система не зависела от других частей. Реактивное программирование играет важную роль в системах, которые несут архитектуру микросервисов, даже столь же важную, если реактивность реализована в монолитной архитектуре, потому что, реализуя реактивность, система получит преимущества в виде асинхронности, потока данных и событийного управления. Преимущества реактивного программирования заключаются в эффективности, максимальном использовании ресурсов за счет использования неблокируемого ввода-вывода (например, запросы к базе данных ввода-вывода, вызов внешних API). С точки зрения масштабируемости он способен обрабатывать множество параллельных задач, не блокируя потоки.

Здесь автор даст небольшое описание того, почему он обсуждает Reactive. Это связано с тем, что автор столкнулся с препятствиями в

[1]Application Programming Interface

системе ECS[2] в одной из компаний, где работает автор. Система была построена с монолитной архитектурой, в которой объединены ядро ECS и система для бэк-офиса. Проблема, которая часто возникает в текущей системе, заключается в том, что система часто испытывает простои, особенно когда к системе поступает много запросов. Даже несмотря на вертикальное масштабирование или дополнительную оперативную память на аппаратной стороне, простои все равно часто случаются, что приводит к финансовым потерям для компании.

Почему бы не провести разделение системы или не перестроить систему в микросервисы, потому что это занимает много времени и ресурсов, что истощит финансы компании. Поэтому для решения вышеуказанной проблемы автор ищет другое решение, а именно пытается создать приложение с реактивной концепцией с минимальными ресурсами.

После внедрения будет проведено тестирование с точки зрения производительности, эффективности и масштабируемости. И сравнение результатов с традиционной нереактивной системой, при этом ожидается, что результаты предоставят обзор в качестве соображения для перехода от системы, которая все еще нереактивна, к реактивной системе.

[2]E-Money core system

1. Постановка задачи

Целью данной работы является сравнение реактивных и нереактивных приложений, реализованных с использованием языка программирования Java и фреймворка Spring. Для достижения этой цели были поставлены следующие задачи:

- Изучите существующие подходы к решению проблемы одновременных запросов.
- Внедрите фреймворк, обеспечивающий одновременные запросы.
- Проанализируйте и сравните время отклика, использование ЦП и памяти и измерьте, сколько запросов система может обработать в секунду.

2. Обзор

Новая система, как правило, быстро реагирует на наши запросы, тем более что пользователей в ней пока немного. Проблемы в системе возникнут, когда к ней обратится много пользователей. Примером проблемы, с которой мы часто сталкиваемся, является блокировка, которая заставляет следующий запрос пользователя ждать, пока завершится блокирующий запрос [3]. Чрезмерная блокировка может снизить общую производительность системы, так как потоки не могут продолжать работу и эффективно расходовать циклы процессора. Это особенно актуально для высокопроизводительных систем. Для наглядности ниже показано, как происходит многопоточная обработка запросов в нереактивной системе.

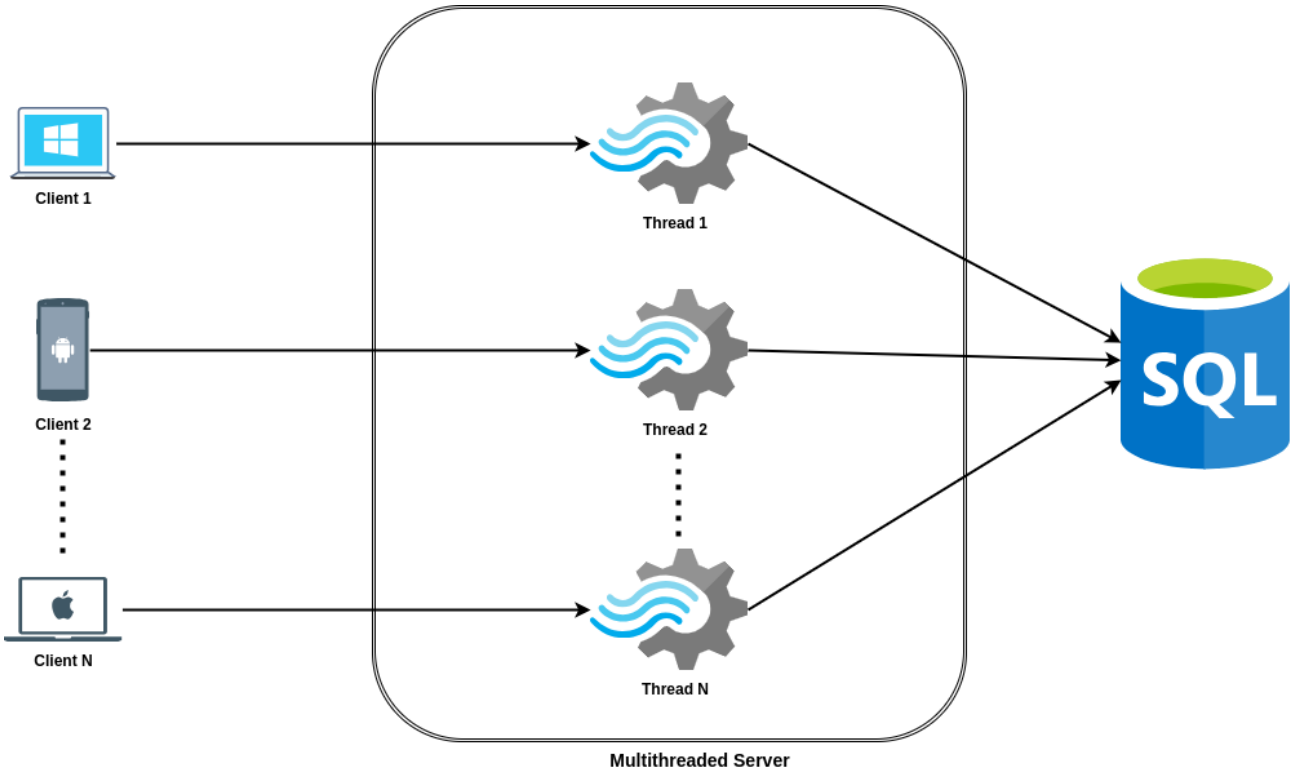

Рис. 1: Многопоточный сервер

Исходя из приведенной иллюстрации, можно сделать вывод, что каждый запрос будет обрабатываться потоком, если используемый поток достиг предела, то возникает блокировка (пример: запрос к базе данных), тогда новый запрос не может быть обработан и должен ждать,

пока не появится свободный поток. Однако, исходя из приведенного выше примера, существует несколько способов уменьшить блокировку в нереактивных системах, включая вертикальное масштабирование и горизонтальное масштабирование.

2.1. Текущая архитектура

Как автор уже писал во введении. Архитектура приложения emoney, которое в настоящее время работает, представляет собой монолитную архитектуру. где все входящие запросы от многих клиентов будут обрабатываться в одной системе. Более подробную информацию см. на изображении ниже.

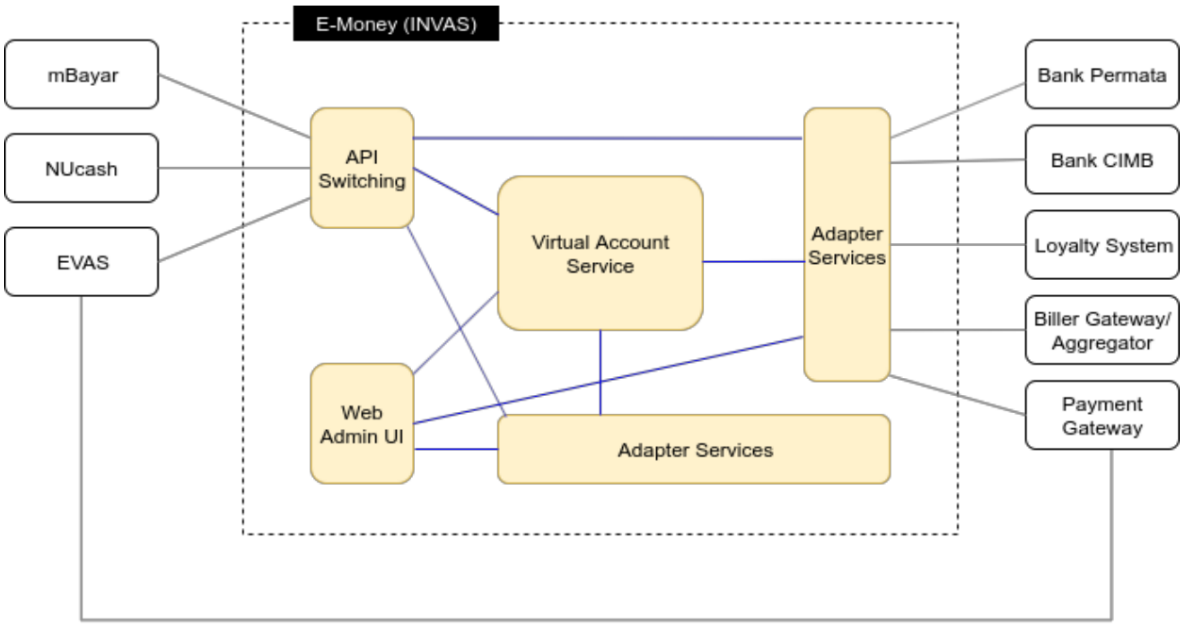

Рис. 2: Текущая архитектура

Здесь система состоит из:

1. . API-коммутации, которая используется для обработки входящих запросов от внешних служб, точнее, этот микросервис функционирует как маршрутизатор, сервер OAuth и шлюз между конечной точкой API клиента и службой назначения.

2. Служба виртуальных счетов: этот микросервис функционирует как основная система. Его задача — управлять счетами и транзакциями клиентов и продавцов, а также конфигурациями комиссий.

3. Веб-интерфейс администратора: это веб-приложение предоставляет административные или бэк-офисные функции, такие как управление конфигурациями микросервисов, данными клиентов, выставление счетов за продукты, выполнение ручных транзакций, создание отчетов и мониторинг действий.

4. Мобильные приложения: мобильные приложения предоставляют интерфейс для функций электронных денег для клиентов. Например, существующие мобильные приложения mBayar и NUcash.

5. Адаптеры сторонних поставщиков Этот адаптер представляет собой набор микросервисов, каждый из которых отвечает за обработку бизнес-сервисов и интеграцию с внешними системами, такими как банки, выставление счетов, SMS-шлюзы, службы электронной почты, службы push-уведомлений и т. д.

Из рисунка выше совершенно ясно, что основная система emoney использует монолитную архитектуру, где транзакции, управление клиентами, управление клиентами и отчетность обрабатываются в одной системе, а именно в службе виртуальных счетов. Из-за этого часто происходит простой системы, когда много пользователей совершают транзакции, а бэк-офис выполняет такие действия, как создание ежемесячных отчетов, которые требуют большого количества ресурсов. Из вышеприведенной проблемы предлагается несколько решений, включая горизонтальное масштабирование и вертикальное масштабирование.

2.2. Горизонтальное масштабирование

Горизонтальное масштабирование (также известное как масштабирование наружу) означает добавление в систему большего количества машин (серверов, узлов или экземпляров) для распределения рабочей

нагрузки [4]. Вместо того чтобы увеличивать ресурсы (процессор, оперативную память и т. д.) одной машины, вы распределяете задачи по нескольким машинам, чтобы повысить производительность, доступность и отказоустойчивость. Обычно горизонтальное масштабирование осуществляется путем добавления физических машин, виртуальных машин или облачных экземпляров. Горизонтальное масштабирование реализует распределенную систему, в которой рабочая нагрузка распределяется между узлами.

Для распределения рабочей нагрузки между системами обычно используют балансировщик нагрузки, который управляет трафиком и распределяет нагрузку между другими системами. Это снижает нагрузку на одну систему при обработке множества запросов. Горизонтальное масштабирование очень подходит для непредсказуемых систем, таких как социальные сети, электронная коммерция и электронные деньги, кроме того, если в одной системе возникает проблема, то система может продолжать работать, обеспечивая более высокую доступность и надежность системы.Вот простой пример того, как выполняется горизонтальное масштабирование.

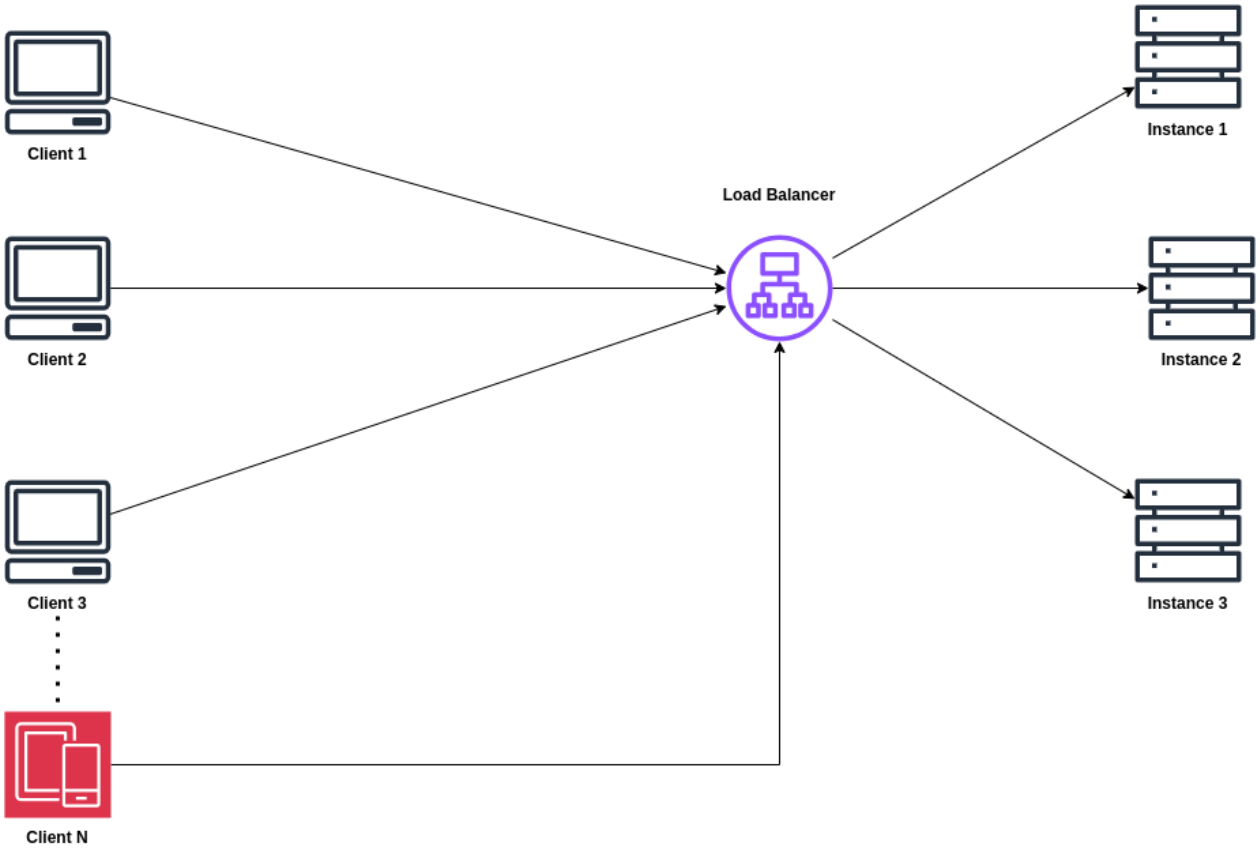

Рис. 3: Горизонтальное масштабирование

Исходя из вышеизложенного, это решение потребует больших затрат денег и человеческих ресурсов на разработку системы.

2.3. вертикальное масштабирование

Вертикальное масштабирование - это процесс добавления ресурсов, таких как оперативная память и хранилище, к одной машине для увеличения мощности [4]. Вместо того чтобы добавлять больше машин, как при горизонтальном масштабировании, вертикальное масштабирование направлено на модернизацию машин, чтобы сделать их более мощными. Модернизация включает в себя добавление большего количества процессоров, увеличение объема оперативной памяти, переход на более быстрое хранилище (SSD)[3] или увеличение пропускной способности сети. При вертикальном масштабировании все рабочие нагрузки системы обрабатываются одной машиной, что упрощает управление.

[3]Solid-state drive

Кроме того, есть несколько преимуществ, которые можно получить при вертикальном масштабировании, а не горизонтальном, в том числе один системный узел получит прямую выгоду от увеличения ЦП, памяти и хранилища. Нет необходимости во многих изменениях архитектуры системы и нет необходимости управлять множеством машин и балансировщиков нагрузки. Однако вертикальное масштабирование имеет недостатки: добавление дополнительных ресурсов не всегда пропорционально увеличивает производительность из-за программных или архитектурных ограничений, а когда машина выходит из строя, все операции прекращаются, и это может привести к потерям в бизнесе. Из двух предложенных решений есть решение, которое было внедрено в систему Emoney, а именно вертикальное масштабирование, но это дополнение все еще не оказало существенного влияния на систему.

3. Описание решения

Исходя из предложенных выше решений, можно выделить несколько трудностей. Для начала горизонтального масштабирования требуется физический сервер, или виртуальный сервер, или аренда IaaS[4], компетентные специалисты для управления и финансирование, которое должно быть тщательно рассчитано. Если финансирование будет рассчитано неверно, это приведет к потерям, не говоря уже о том, что система должна быть способна адаптироваться к новой среде. А это сильно истощит финансы компаний, которые только начинают осваивать ИТ-сферу.

Поэтому автор предлагает решение, которое не обойдется компании слишком дорого и позволит снизить нагрузку на сервер за счет минимизации использования потоков, но при этом сможет обрабатывать множество пользовательских запросов, а именно с помощью реактивного программирования. Реактивное программирование - не новая концепция, но она вновь стала популярной после появления архитектуры микросервисов. Это делает реактивное программирование еще более популярным, поскольку оно предлагает асинхронные процессы. «Ниже показана концепция реактивного программирования на языке программирования Java» от Виктор Кланг [2].

[4]Infrastructure as a Service

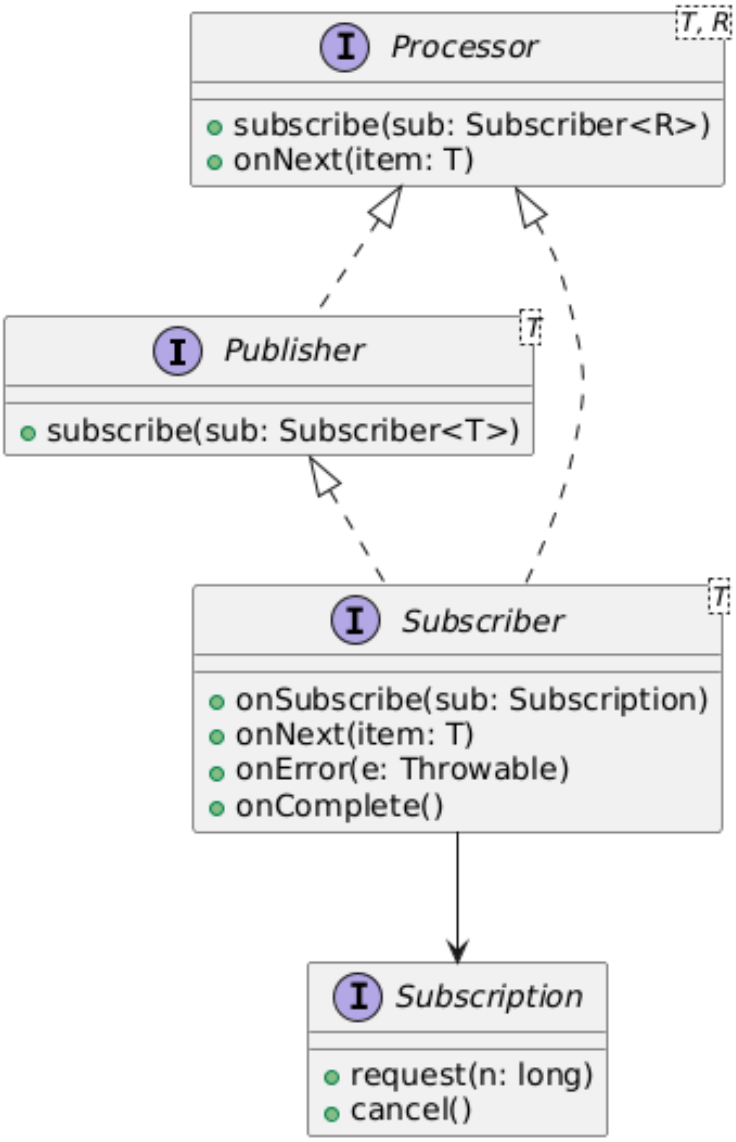

Рис. 4: Reactive Programming Java

- Издатель: Выдает данные (по запросу) и Передает данные подписчику через подписку.
- Подписчик: Подписывается на Издателя и Сигнализирует о запросе с помощью request() и потребляет выданные элементы.
- Процессор: Выступает в качестве посредника (необязательно) и Получает данные от издателя, обрабатывает их и пересылает другому подписчику.
- Подписка: Управляет потоком данных и Обрабатывает обратное давление, регулируя количество выдаваемых элементов.

3.1. Определите основные метрики для измерения

Чтобы проверить, окажет ли переход от нереактивного к реактивному программированию существенное влияние, в обеих системах будут измеряться несколько метрик:

1. Задержка относится к задержке между запросом и ответом в системе. Задержка измеряет время, необходимое для перемещения данных из одной точки в другую.

2. Пропускная способность относится к объему работы, которую система может обработать за определенный период времени. Пропускная способность обычно измеряется в запросах в секунду (RPS), транзакциях в секунду (TPS) или сообщениях в секунду, в зависимости от контекста.

3. Частота ошибок — это процент запросов или транзакций, которые не были выполнены, по сравнению с общим числом запросов, обработанных системой. Частота ошибок показывает, как часто возникают ошибки, и является ключевым показателем для измерения стабильности и устойчивости системы.

4. Использование ресурсов относится к объему системных ресурсов (ЦП, память, диск, сеть и т. д.), потребляемых приложением или системой во время ее работы. Это важный показатель для мониторинга производительности, масштабируемости и экономической эффективности.

3.2. Используемые инструменты

Apache JMeter — это инструмент с открытым исходным кодом, используемый для тестирования производительности, нагрузочного тестирования и функционального тестирования приложений, особенно веб-приложений, API и серверов [1]. Этот инструмент помогает телескопировать реальный пользовательский трафик, чтобы измерить, насколько хорошо система работает в различных условиях. Apache JMeter

можно использовать для тестирования производительности на статистических и динамических ресурсах, динамических веб-приложениях. Этот инструмент можно использовать для хранения больших нагрузок на сервере, группе серверов, сети или объекте для проверки прочности или анализа общей производительности при различных типах нагрузок. Apache Jmeter способен выполнять нагрузочные тесты и тесты производительности множества различных приложений, серверов и протоколов. Jmeter похож на браузер или, точнее, на множество браузеров, которые оценивают множество пользователей, получающих доступ к определенной системе. Из результатов этого теста будут видны ключевые показатели, которые можно использовать для рассмотрения перехода системы из нереактивного состояния в реактивное. Результирующие показатели состоят из задержки, пропускной способности, частоты ошибок, использования ресурсов.

Заключение

В результате работы над учебными практиками были решены следующие задачи:

- внедрение вертикального шкалирования.
- горизонтальное шкалирование не проводилось, в связи с отсутствием средств на него.

Планирование дальнейшей работы:

- реализация системы с использованием spring framewrok.
- сравнение реактивных и нереактивных систем с точки зрения производительности, гибкости и масштабируемости.

Список литературы

[1] Community Apache. Apache Jmeter. — URL: https://jmeter.apache.org/index.html (дата обращения: 12 августа 2022 г.).

[2] Klang Viktor. Reactive Streams JVM. — URL: https://github.com/reactive-streams/reactive-streams-jvm (дата обращения: 21 января 2022 г.).

[3] Maldini Stephane, Baslé Simon. Introduction to Reactive Programming. — URL: https://projectreactor.io/docs/core/release/reference/reactiveProgramming.html (дата обращения: 7 марта 2025 г.).

[4] Slingerland Cody. Horizontal Vs. Vertical Scaling: Which Should You Choose? — URL: https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling (дата обращения: 20 января 2024 г.).

[5] Toerktumlare. Reactive vs non reactive. — URL: https://stackoverflow.com/questions/70937652/should-we-use-a-reactive-stack-web-framework-like-spring-webflux (дата обращения: 30 декабря 2023 г.).