МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

ПРИМЕНЕНИЕ АРХИТЕКТУРНЫХ ПАТТЕРНОВ CQRS И EVENT SOURCING В РАЗРАБОТКЕ СИСТЕМЫ С ИСПОЛЬЗОВАНИЕМ AXON FRAMEWORK.

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 5 курса 551 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Рудописова Николая Владимировича

Научный руководитель
зав.каф., к.ф.-м.н., доцент _____ И. А. Батраева

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4
1 Описание 5
1.1 Модель 5
1.2 Используемые технологии 5
1.2.1 Spring Framework 6
1.2.2 Event Sourcing 6
1.2.3 CQRS 7
1.2.4 AXON Framework 7
1.2.5 PostgreSQL 8
1.2.6 JMeter 8
1.3 Теоретические основы разработки проектов с использованием объектно-ориентированного подхода 8
1.4 Теоретические основы разработки проектов с использованием событийно-ориентированного подхода 9
2 Реализация 10
2.1 Реализация демо-проекта с объектно-ориентированным подходом 10
2.1.1 Модели объектов 10
2.1.2 Репозитории 10
2.1.3 Сервисный уровень 10
2.1.4 API 10
2.1.5 Работа приложения 11
2.2 Реализация демо-проекта событийно-ориентированным подходом 11
2.2.1 Модели команд 11
2.2.2 Модели событий 11
2.2.3 Агрегат 12
2.2.4 Модели данных для запросов (DTO) 12
2.2.5 Модели запросов 12
2.2.6 Проекция 12
2.2.7 API 13
2.2.8 Работа приложения 13
2.3 Сравнение 13
2.3.1 Замеры объектно-ориентированного приложения 14
2.3.2 Замеры событийно-ориентированного приложения 14

2.3.3 Общее 14
ЗАКЛЮЧЕНИЕ 15
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 16

ВВЕДЕНИЕ

При проектировании современного программного обеспечения, предназначенного для обработки больших данных существенными являются такие характеристики как скорость записи и чтения данных с устройств хранения, эффективность организации постоянного аудита хранящихся данных и действий, производимых над ними. Поэтому разработчики вынуждены искать новые более производительные средства и архитектурные подходы к разработке систем обработки данных.

Тема данной дипломной работы связана с проектами выполняющимися в компании Neoflex, в рамках которых потребовалось внедрение новых архитектурных подходов в разработке программного обеспечения. В качестве объектов исследования предложены перспективные архитектурные паттерны *Event Sourcing* [1] и *CQRS* (Command Query Responsibility Segregation) [2].

Целью работы является разработка демо-проекта и демонстрация преимуществ подхода с событийно-ориентированной архитектурной для выполнения ряда задач. Были поставлены следующие задачи:

- Рассмотрение принципы событийно-ориентированной архитектуры, патерны *Event Sourcing* и *CQRS*;
- Разработка демо-проекта биллинговой системы с использованием Axon Framework [3];
- Разработка аналогичного проекта с использованием Spring Framework [4] с объектно-ориентированной архитектурой для сравнения;
- Сравнение реализаций, демонстрация преимуществ.

1 Описание

1.1 Модель

Модель (рис. 1) состоит из следующих элементов:

— Аккаунт, содержащий информацию о номере, балансе и статусе;
— Информацию о произведенных переводах, содержащая информацию о номерах аккаунтов и сумме;
— Информацию о всех движениях по аккаунту (пополнения/списания);
— Аудит всех операций переводов, пополнений и списаний по аккаунту.

В случае *Event Sourcing* модель отличается. Аудит хранится в виде списка действий пользователей (событий).

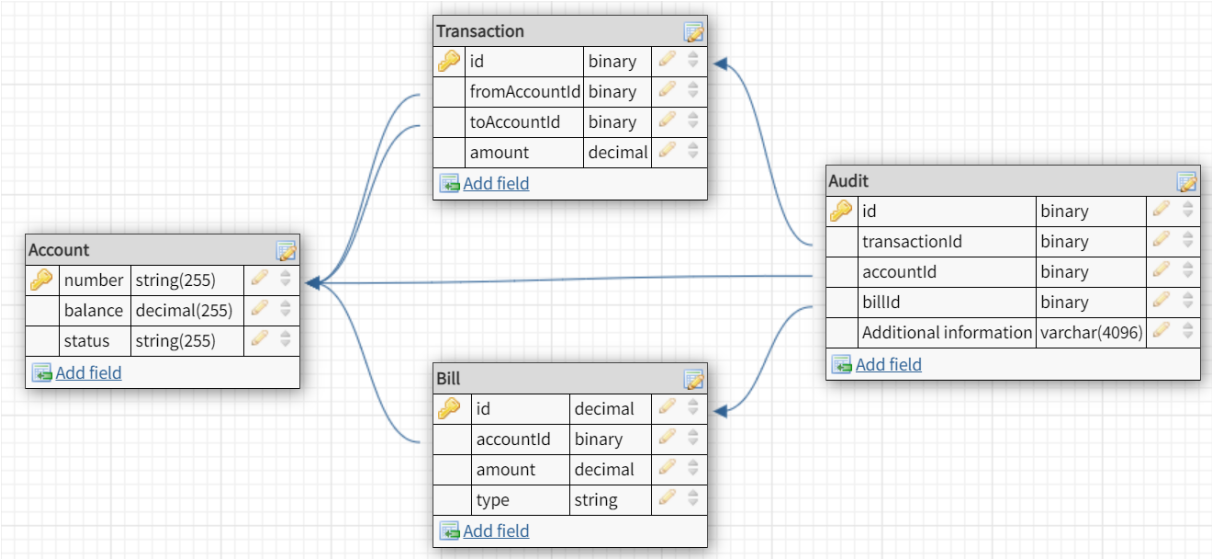

Рисунок 1 – Модель

1.2 Используемые технологии

В качестве технологического стека использовались следующее:

— Язык программирования *Java*;
— Архитектурные паттерны *Event Sourcing* и *CQRS*
— *Axon Framework* — предоставляет единый продуктивный способ разработки приложений Java, которые могут развиваться без значительного рефакторинга от монолита до микросервисов, управляемых событиями [5];
— *Spring Framework* — универсальный фреймворк с открытым исходным кодом для Java-платформы [6];
— В качестве базы данных выбрана *PostgreSQL*;

— Для проведения нагрузочного тестирования и замеров производительности использовалось средство *Apache JMeter*;

1.2.1 Spring Framework

Spring Framework — это платформа Java, которая обеспечивает всестороннюю поддержку инфраструктуры для разработки приложений Java. Spring управляет инфраструктурой, поэтому вы можете сосредоточиться на своем приложении [7].

В работе используются 3 основных модуля для разработки Web сервисного приложения:

— *Spring Boot* — это среда с открытым исходным кодом, используемая для создания сервиса. Является основой для Spring приложения. Модуль помогает легко инициализировать все конфигурации приложения.
— *Spring JPA* — предоставляет простые интерфейсы для описания подключения к любым, поддерживаемым языком программирования Java базам данных. Внутри модуля используется *Hibernate*, который предоставляет огромное количество различных возможностей работы с базами данных.
— *Spring Web* — фокусируется на предоставлении инфраструктуры для создания и запуска многофункциональных веб-приложений. Предоставляет механизмы для простой конфигурации API сервиса.

1.2.2 Event Sourcing

Паттерн *Event Sourcing* предполагает, что вместо хранения конечного состояния должна храниться последовательность **событий**, которые приводят к конечному состоянию [1].

Для того, чтобы получить актуальное состояние приложения, необходимо воспроизвести всю последовательность событий. [8]

Благодаря Event Sourcing мы гарантируем, что все изменения в объектах домена инициируются объектами событий. Это дает некоторые плюсы, такие как:

— **Полная перестройка (Complete rebuild)**: Мы имеем средства, которые могут нам позволить отбросить текущее состояние моделей и восстановить его, повторно запустив события из журнала;
— **Запрос по времени (Temporal Query)**: Можно восстановить состояние приложения на любой момент времени;

— **Переигрывание событий (Event Replay)**: Если обнаружилось, что какое-то событие работало некорректно. Мы можем сравнить последствия, переиграв эти события после исправления. И если нас все устраивает, то принять это состояние как актуальное.
— **Откат события (Back Replay)**. Так же как и переигрывание событий, можно применить событие в обратном направлении, чтобы вычислить предыдущее состояние.

1.2.3 CQRS

Паттерн *CQRS* подразумевает разделение запросов на **Команды** и **Запросы**. Запросы возвращают результат и не изменяют наблюдаемое состояние системы. Команды изменяют состояние системы, но не обязательно возвращают значение.

Кроме того, модели команд может отличаться от моделей запросов [9]. Это помогает избежать множественные объединения таблиц, которые тратят много времени. Вместо этого можно хранить композитные данные в удобном для чтения виде.

Так же вместе с CQRS используются другие механизмы, которые вносят свои плюсы в реализацию [10]. Вот определения некоторых из них:

— Агрегат (Aggregate) — это шаблон, описанный в Domain-Driven Design (DDD) [11], который логически группирует различные сущности путем привязки сущностей к агрегированному корню. Агрегаты обычно хранят кешированное состояние для повышения производительности, но могут отлично работать и без него.
— Проекция (Projection) — еще один важный паттерн, который приносит большую пользу CQRS. По сути, проекция означает представление объектов предметной области в различных формах и структурах. Эти проекции исходных данных доступны только для чтения и оптимизированы для обеспечения более удобного чтения.
— Проектор (Projector) — механизм для формирования проекций из хранилища команд.

1.2.4 AXON Framework

AXON Framework [3] — это фреймворк, который помогает разработчикам создавать масштабируемые и расширяемые приложения, решая эти проблемы

непосредственно в архитектуре. Благодаря этому фреймворку можно с легкостью построить масштабируемые и расширяемые приложения [12].

Axon Framework использует Spring framework [13], который позволяет проводить простые конфигурации приложений.

Так же Axon предоставляет удобные средства для мониторинга подключенных приложений, кол-ва вызовов команд и запросов , список всех происходящих событий с подробностями.

В основе приложения, созданного на Axon Framework лежит Axon Server, который выполняет роль связующего компонента для всех подключенных приложений. Работа сервера базируется на принципах Event Driven Architecture [14]. Он занимается распределением нагрузки между инстансами сервисов и распределением запросов по приложениям [15]. Такие серверы называются **Аркестраторами**.

1.2.5 PostgreSQL

PostgreSQL — это гибкая система управления объектно-реляционной базой данных с открытым исходным кодом с функциями, предназначенными для соответствия изменениям рабочих нагрузок, от отдельных компьютеров до хранилищ данных и веб-сервисов с множеством одновременных пользователей [16].

1.2.6 JMeter

JMeter — инструмент для проведения нагрузочного тестирования, разрабатываемый Apache Software Foundation [17]. Этот инструмент позволяет создавать сценарии для проведения нагрузочного тестирования Web сервисов и, в качестве результата своей работы, выводит все необходимые метрики [18].

1.3 Теоретические основы разработки проектов с использованием объектно-ориентированного подхода

Объектно-ориентированный подход к моделированию системы относится к самым популярным. Это обуславливается простотой понимания и построения архитектуры приложения [19].

Подход базируется на представление всех сущностей в предметной области в виде объектов со всеми соответствующими связями. Данные хранятся в виде конечного состояния и постоянно видоизменяются.

Такие системы зачастую плохо масштабируются. Это связано с тем, что база является узким местом, где возникают проблемы с ожиданием доступа к данным. Например если несколько пользователей одновременно изменяют одни и те же данные, что вызывает очередь на доступ к этим данным [20].

1.4 Теоретические основы разработки проектов с использованием событийно-ориентированного подхода

Данный подход, в отличии от объектно-ориентированного, базируется на хранении в базе событий и разделении всех обращений пользователей на команды и запросы. Благодаря этому удается избежать операций *UPDATE* в базу данных. События только сохраняются и читаются. Отделение модели запросов позволяет так же сократить время на чтение данных, т.к. данные хранятся в том видел, которое требуется для предоставления пользователю [21].

У данного подхода можно выделить следующие преимущества:

— *Быстрая обработка изменений* (Event Sourcing);
— *Быстрое чтение* (CQRS);
— *Формирование и хранение аудита* (Event Sourcing).

Применение паттернов *CQRS* и *Event Sourcing* позволяет добиться быстрой реакции сервиса на запросы и при этом иметь полный аудит событий, происходивших с агрегатами. Для реализации такого сервиса существует много сильных средств. Для реализации данной задачи был выбран AXON Framework.

Так же у подхода есть недостатки. Реализация решения с архитектурными шаблонами *EDA (event driven architecture)* [22] и принципами *DDD (domain-driven design)* [23] имеет большую сложность в отличии от стандартных подходов с использованием Объектно-Ориентированного подхода. Для построения такого решения требуются более компетентные специалисты и больше ресурсов [24].

Среди недостатков можно выделить некоторые пункты [25]:

— Такой подход с первого взгляда интуитивно непонятен и требует высокого уровня компетенции для вхождения;
— Добавляется дополнительная сложность в структуре.
— Добавляются распределенные репозитории, из-за чего может возникнуть проблема с синхронизацией;
— Присутствует дублирование кода.

2 Реализация

2.1 Реализация демо-проекта с объектно-ориентированным подходом

Приложение разрабатывается последовательно и состоит из разделов:

- Модель объектов — содержит в себе информацию об объектах;
- Репозитории — описываются способы получения данных из СУБД;
- Сервисный уровень — описывается бизнес-логика;
- Уровень контроллеров (API) — описывается программный интерфейс.

2.1.1 Модели объектов

Модели объектов программы представляют из себя POJO с использованием аннотаций *@Entity*. Каждый объект модели реализован в виде отдельного Java класса.

2.1.2 Репозитории

Репозитории реализованы с использованием *Spring JPA*. Модуль позволяет написать простую реализацию подключения каждой модели объектов к соответствующей таблице базы данных.

2.1.3 Сервисный уровень

На сервисном уровне содержится вся логика бизнес процессов приложения. Она отвечает за валидацию и правила изменения объектов, с которыми работает программа.

Для ведения аудитов создается дополнительная логика, которая используется там, где необходимо ведение аудитов.

2.1.4 API

Для разработки API были использованы аннотации Spring Framework *@RestController* и *@RequestMapping*.

Для работы с сервисом были реализованы следующие Rest API:

- **post** */account/create* — создание счета
- **post** */account/id/payment?amount=??* — пополнение/снятие со счета
- **get** */account/id* — получение информации по счету
- **get** */bill/account/id* — получение всех биллов по счету
- **get** */bill/id* — получение информации по конкретному биллу

— **get** */transaction/create?accountFromId=??|accountToId=??|amount=??* — перевод с одного аккаунта на другой
— **get** */transaction/id* — получение перевода
— **get** */event/account/id* — получение списка всех событий для аккаунта
— **get** */audit/id* — получение аудита
— **get** */audit/account/id* — получение всех аудитов для аккаунта
— **get** */audit/transaction/id* — получение аудита для перевода
— **get** */audit/bill/id* — получение аудита для билла

2.1.5 Работа приложения

Приложение запускается с помощью Spring-boot. Spring-boot автоматически запускает сервер, открывает необходимые порты и подключается к базе данных. После запуска становится возможным обратиться к серверу через API.

2.2 Реализация демо-проекта событийно-ориентированным подходом

2.2.1 Модели команд

Модели команд представляют из себя набор данных для выполнения конкретного действия с агрегатом. Созданы следующие команды:

— **Создать новый счет** (CreateAccountCommand);
— **Изменить статуса счета на «закрыт»** (CloseAccountCommand);
— **Поступление средств на баланс счета** (DebitAccountCommand);
— **Снятие средств с баланса счета** (CreditAccountCommand);
— **Перевод с одного счета на другой** (TransactionsToAccountCommand).

2.2.2 Модели событий

Результатом выполнения команд должны быть события. События представляют из себя набор данных, необходимый для изменения состояния объекта. Созданы следующие события:

— **Счет создан** (AccountCreatedEvent);
— **Счет закрыт** (AccountClosedEvent);
— **Поступили средства на баланс** (AccountDebitAddedEvent);
— **Сняты средства с баланса** (AccountCreditSubbedEvent);
— **Создан перевод со счета на счет** (TransactionCreatedEvent).

2.2.3 Агрегат

Для выполнения поставленной задачи понадобится только один агрегат — **Account**.

Каждый экземпляр агрегата имеет свой ID для идентификации команд и событий и отвечает за их обработку.

2.2.4 Модели данных для запросов (DTO)

Модели DTO (Data Transfer Object) — это то, что получает пользователь в ответ на запрос данных. Для приведения списка событий к виду, в котором пользователь получает данные используется *"проекция"*. Созданы следующие модели:

- **AccountView**
- **BillView**
- **TransactionView**

Исключением является только сущность **Аудит**. Аудит будет строиться из хранилища событий.

Так же для этих сущностей создан интерфейс подключения к базе данных.

2.2.5 Модели запросов

Модели запросов, так же как и модель команд, содержат в себе только информацию для получения данных. Созданы следующие модели запросов:

- **Получить аккаунт** (AccountQuery);
- **Получить все биллы для аккаунта** (FindBillsByAccountQuery);
- **Получить билл** (FindBillByIdQuery);
- **Получить все переводы для аккаунта** (FindTransactionsByAccountQuery);
- **Получить перевод** (FindTransactionsByIdQuery).

2.2.6 Проекция

Проекторы — одна из основных частей реализации. Отвечают за проекцию событий на модель чтения (view). Так же как и агрегаты имеют обработчики событий.

2.2.7 API

Для работы с сервисом были реализованы следующие Rest API:

— **post** */account/create* — создание счета
— **post** */account/id/close* — закрытие счета
— **post** */id/transaction/targetId?amount=??* — создание перевода
— **post** */account/id/debit?amount=??* — пополнение счета
— **post** */account/id/credit?amount=??* — снятие со счета
— **get** */account/id* — получение информации по счету
— **get** */account/id/bills* — получение всех биллов по счету
— **get** */account/id/transactions* — получение всех переводов по счету
— **get** */bill/id* — получение била
— **get** */transaction/id* — получение перевода
— **get** */event/account/id* — получение списка всех событий для аккаунта

2.2.8 Работа приложения

Приложение запускается с помощью Spring-boot. Spring-boot автоматически запускает сервер, открывает необходимые порты и подключается к базе данных.

2.3 Сравнение

Для сравнения двух реализаций проводилось нагрузочное тестирование, чтобы увидеть как ведут себя обе реализации при разных нагрузках. Для проведения нагрузочного тестирования задействована утилита JMeter.

JMeter — инструмент для проведения нагрузочного тестирования, разрабатываемый Apache Software Foundation [17]. Этот инструмент позволяет создавать сценарии для проведения нагрузочного тестирования Web сервисов и, в качестве результата своей работы, выводит все необходимые метрики [18].

Для проведения тестирования создан сценарий, общее число запросов в котором $= 4 * 20 + 3 * 20 = 140$.

Сценарии запускаются с разным количеством потоков, которые имитируют пользователей, одновременно обращающихся к серверу. Сценарии прогоняются при следующих значениях потоков:

— 8 потоков — максимальное значение. Ограничено количеством потоков на процессоре;
— 4 потока — медианное значение для наполнения статистики;

— 2 потока;
— 1 поток;

2.3.1 Замеры объектно-ориентированного приложения

В результате анализа данных видно, что с ростом количества клиентов растет и общее время обработки запросов. Это связано с блокирующим поведением при обращении к базе данных для обновления данных.

2.3.2 Замеры событийно-ориентированного приложения

В результате анализа данных видно, что с ростом количества клиентов растет и общее время обработки команд. А время на выполнение запросов чтения остается практически неизменным.

2.3.3 Общее

Замеры показывают, что при росте нагрузки, подход с использованием Axon Framework, при правильной конфигурации, может обрабатывать запросы пользователей с большей скоростью.

В некоторых случаях второй подход может показывать результаты хуже. Многое зависит от правильности построения и конфигурации.

ЗАКЛЮЧЕНИЕ

В результате выполнения бакалаврской работы были разработаны 2 демо-сервиса с различными архитектурными подходами и произведены замеры их производительности. Были собраны данные, и на их основе можно делать выводы о преимуществах использования паттернов CQRS и *Event Sourcing* и Axon Framework.

На основе созданного демо-проекта можно заключить, что при правильном использовании комбинация паттернов CQRS и *Event Sourcing* можно получить такие преимущества, как прирост производительности обработки запросов пользователей и записи данных. Так же получить возможность легко масштабировать сервис, что позволит принять большую нагрузку на систему.

Так же паттерн *Event Sourcing* предоставляет возможность вести аудит, не добавляя при этом никакой дополнительной логики, и позволяет *«откатывать»* события до необходимого момента. История событий обладает всей необходимой информацией.

Axon Framework — мощный механизм, который реализует данные архитектурные подходы и упрощает разработку приложения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Fowler, M.* Event Sourcing [Электронный ресурс] / M. Fowler. — URL: https://martinfowler.com/eaaDev/EventSourcing.html (Дата обращения 12.04.2021). Загл. с экр. Яз. англ.

2 *Fowler, M.* CQRS [Электронный ресурс] / M. Fowler. — URL: https://martinfowler.com/bliki/CQRS.html (Дата обращения 15.04.2021). Загл. с экр. Яз. англ.

3 Axon Reference Guide [Электронный ресурс]. — URL: https://docs.axoniq.io/reference-guide/v/3.3/part-i-getting-started/introduction (Дата обращения 10.05.2021). Загл. с экр. Яз. англ.

4 Building a RESTful Web Service [Электронный ресурс]. — URL: https://spring.io/guides/gs/rest-service/ (Дата обращения 15.05.2021). Загл. с экр. Яз. англ.

5 About Axon [Электронный ресурс]. — URL: https://axoniq.io/product-overview/axon (Дата обращения 12.04.2021). Загл. с экр. Яз. англ.

6 Spring Framework [Электронный ресурс]. — URL: https://ru.wikipedia.org/wiki/Spring_Framework (Дата обращения 12.04.2021). Загл. с экр. Яз. англ.

7 Introduction to Spring Framework [Электронный ресурс]. — URL: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/overview.html (Дата обращения 14.04.2021). Загл. с экр. Яз. англ.

8 *Melnik, G.* Exploring CQRS and Event Sourcing / G. Melnik, J. Dominguez, F. Simonazzi. — California, USA: Microsoft press, 2012. — 376 pp.

9 *Вернон, В.* Реализация методов предметно-ориентированного проектирования / В. Вернон. — М: Вильямс, 2018. — 688 с.

10 *Chandrakant, K.* CQRS and Event Sourcing in Java [Электронный ресурс] / K. Chandrakant. — URL: https://www.baeldung.com/cqrs-event-sourcing-java (Дата обращения 5.05.2021). Загл. с экр. Яз. англ.

11 *Zimarev, A.* Hands-On Domain-Driven Design with .NET Core. Tackling complexity in the heart of software by putting DDD principles into practice / A. Zimarev. — Birmingham, UK: Packt Publishing, 2019. — 446 pp.

12 *Haup, B.* Предметно-ориентированное проектирование в Enterprise Java / В. Наир. — М: ДМК Пресс, 2020. — 306 с.

13 AxonFramework, автоматическая настройка SpringBoot [Электронный ресурс]. — URL: https://russianblogs.com/article/84291455594/ (Дата обращения 02.06.2021). Загл. с экр. Яз. рус.

14 Предварительный просмотр архитектуры Axon framework [Электронный ресурс]. — URL: https://russianblogs.com/article/1970800907/ (Дата обращения 05.06.2021). Загл. с экр. Яз. рус.

15 *Newman, S.* Building Microservices: Designing Fine-Grained Systems / S. Newman. — California, USA: O'Reilly Media, 2015. — 280 pp.

16 *Hammink, J.* An introduction to PostgreSQL [Электронный ресурс] / J. Hammink. — URL: https://aiven.io/blog/an-introduction-to-postgresql (Дата обращения 17.04.2021). Загл. с экр. Яз. англ.

17 Apache JMeter [Электронный ресурс]. — URL: https://jmeter.apache.org/ (Дата обращения 28.05.2021). Загл. с экр. Яз. англ.

18 *Rodrigues, A. G.* Master Apache JMeter - From Load Testing to DevOps: Master performance testing with JMeter / A. G. Rodrigues, B. Demion, P. Mouawad. — Birmingham, UK: Packt Publishing, 2019. — 466 pp.

19 *Буч, Г.* Объектно-ориентированный анализ и проектирование / Г. Буч, Р. А. Максимчук. — М: Вильямс, 2017. — 720 с.

20 *Metz, S.* Practical Object-Oriented Design / S. Metz. — Boston, USA: Addison-Wesley Professional, 2018. — 288 pp.

21 *Fowler, M.* domain driven design [Электронный ресурс] / M. Fowler. — URL: https://martinfowler.com/tags/domain%20driven%20design.html (Дата обращения 12.04.2021). Загл. с экр. Яз. англ.

22 *Bellemare, A.* Building Event-Driven Microservices / A. Bellemare. — California, USA: O'Reilly Media, 2020. — 324 pp.

23 *Bogatinov, A.* DOMAIN DRIVEN DESIGN VS. OBJECT ORIENTED PROGRAMMING [Электронный ресурс] / A. Bogatinov. — URL: https://www.haselt.com/blog/domain-driven-design-vs-object-oriented-programming (Дата обращения 25.04.2021). Загл. с экр. Яз. англ.

24 *Dahan, U.* When to avoid CQRS [Электронный ресурс] / U. Dahan. — URL: https://udidahan.com/2011/04/22/when-to-avoid-cqrs/ (Дата обращения 28.05.2021). Загл. с экр. Яз. англ.

25 *Hudson, A.* Software architecture is failing [Электронный ресурс] / A. Hudson. — URL: https://www.alexhudson.com/2017/10/14/software-architecture-failing/ (Дата обращения 26.05.2021). Загл. с экр. Яз. англ.