Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 20Б.11-мм

Разработка серверной части платформы для управления автоматизациями

Федькин Александр Андреевич

Отчёт по производственной практике

Научный руководитель:
ст. преп. С.Ю. Сартасов

Консультант:
руководитель группы в ООО «Яндекс.Технологии» А.В. Полозенко

Санкт-Петербург
2023

Оглавление

Введение

Автоматизация (automation) — автоматизированный процесс, который сокращает вмешательство человека в процессы получения, преобразования и передачи информации. В настоящее время они используются повсеместно [8]. Например, автоматические рассылки почтовых сообщений, системы восстановления забытого пароля, когда робот совершает звонок на привязанный к аккаунту номер телефона, — все эти процессы происходят без участия человека, что позволяет существенно сэкономить время людей, позволяя не нанимать операторов для выполнения рутинных действий.

В отделении краудсорсинга компании Яндекс автоматизированы рассылки писем, переносы обращений пользователей между тематическими очередями обработки, а также созданы чат-боты, которые общаются с пользователями, обратившимися в поддержку. Данные автоматизации настраиваются с помощью YAML [14] файлов и скриптов, написанных на языке программирования groovy [3]. Такие конфигурации не являются наглядными в силу отсутствия их визуализации, их довольно сложно поддерживать, что влечет необходимость наличия специальной технической группы для их создания или редактирования.

Несмотря на наличие инструментов для создания автоматизаций как внутри компании, — Nirvana, так и снаружи, — Apache Airflow [1], Apache NiFi [2], PLynx [7], их специализация — управление пайплайнами обработки больших объемов данных, а не управление легковесными автоматизациями. Более того, в них отсутствует часть необходимой нам функциональности, например, остановки в определенной вершине графа исполнения и продолжения работы с входными параметрами, полученными из внешнего вызова API, — подобное поведение требуется для реализации чат-ботов.

Поэтому требуется создать платформу для управления автоматизациями, позволяющую пользователю создавать и редактировать графы исполнения автоматизаций с помощью графического интерфейса, а также определять условия, при которых автоматизация должна за-

пускаться. В рамках данной производственной практики предлагается реализовать серверную часть описанной системы.

1. Постановка задачи

Целью данной производственной практики является разработка серверной части платформы для управления автоматизациями. Для достижения цели были поставлены следующие задачи:

- сформулировать требования к создаваемому продукту;
- произвести обзор существующих инструментов для управления автоматизациями;
- спроектировать архитектуру системы;
- реализовать компоненты системы;
- произвести апробацию системы.

2. Требования

После обсуждения с командой и бизнес-руководителями были сформулированы следующие требования.

2.1. Функциональные требования

Было принято решение для первоначальной версии продукта сосредоточиться на базовом наборе функциональности, позволяющей впоследствии расширять продукт.

- создание, редактирование, версионирование автоматизаций;
- возможность наличия циклов в графах автоматизаций;
- возможность приостановить автоматизацию и ожидать сообщение от пользователя (необходимо для реализации чат-ботов);
- настройка условий автозапуска (триггеров);
- возможность создавать автоматизации без опыта в программировании.

2.2. Нефункциональные требования

Следующие требования были сформулированы в связи с политикой команды и компании.

- язык разработки — Java или Kotlin;
- кеширование графов;
- неблокирующее взаимодействие с сервисами по HTTP;
- слабая связность компонентов платформы;
- легкая расширяемость.

3. Обзор

В этом разделе представлен обзор существующих инструментов для управления автоматизациями.

3.1. Существующие решения

В настоящее время существует много инструментов для управления автоматизациями различных процессов. В результате поиска было выделено четыре решения. Перед началом обзора введем термин:

ETL — часто возникающий в современных системах процесс, состоящий из трех этапов:

1. **Extract** — извлечение данных из различных источников, например, из реляционной базы данных, логов, социальных сетей;
2. **Transform** — преобразование данных. Применение фильтраций, группировок, аггрегирования к данным, собранным на предыдущем этапе, чтобы преобразовать данные в готовый к анализу датасет;
3. **Load** — отправка обработанной информации в некоторое хранилище, место конечного использования, например, озеро данных.

3.1.1. Yandex Nirvana

Nirvana — внутренний сервис Яндекса, предоставляющий облачную платформу для управления вычислительными процессами. Каждый процесс визуализируется как ориентированный ациклический граф и состоит из блоков операций, данных и других графов.

Особенности:

- предоставление вычислительного кластера;
- построение графов исполнения с помощью drag-and-drop в пользовательском интерфейсе;

- веб-интерфейс, HTTP API;
- библиотека часто используемых операций;
- хранилище промежуточных данных процесса.

3.1.2. Apache Airflow

Airflow [1] — фреймворк для оркестрации, позволяющий разрабатывать, планировать и осуществлять мониторинг рабочих процессов. Для описания процессов исполнения используется язык программирования Python.

Особенности:

- построение графов исполнения с помощью Python-кода;
- веб-интерфейс, HTTP API;
- интеграции с популярными сервисами как Elasticsearch, Docker, Telegram и другими;
- богатая функциональность для мониторинга.

3.1.3. Apache NiFi

NiFi [2], как и Airflow, разработан Apache и имеет с ним много сходств, однако, концептуально NiFi — инструмент для ETL-задач. В отличие от Airflow, Nifi не предоставляет возможности мониторинга в реальном времени, а также имеет ограниченную функциональность для планирования задач. Зачастую он используется вместе с Airflow, например, ETL-конвеер Nifi является одной из вершин графа исполнения в пайплайне Airflow.

Особенности:

- построение графов исполнения с помощью drag-and-drop в пользовательском интерфейсе;
- веб-интерфейс, HTTP API;

- интеграции с популярными сервисами.

3.1.4. PLynx

PLynx [7] — платформа для оркестрации и вычислений. Позиционируется как инструмент для проведения ETL-экспериментов. По словам авторов PLynx, их приоритетом было создание инструмента для быстрого проведения различных экспериментов, а не высоконадежного исполнения пайплайна ETL. Имеется возможность создавать плагины для взаимодействия с различными сервисами, цель этого — абстрагирование аналитиков данных от инженерых задач. Аналогично предыдущим инструментам является Python-ориентированным.

Особенности:

- построение графов исполнения с помощью Python-кода;
- веб-интерфейс, HTTP API;
- плагин-ориентированный.

3.2. Результаты обзора существующих решений

Исходя из требований к продукту, выделим параметры сравнения решений:

- возможность версионировать графы;
- возможность наличия циклов в графах автоматизаций;
- возможность остановки автоматизации и продолжения работы с входными параметрами, полученными из внешнего вызова API;
- возможность создавать автоматизации без опыта в программировании;

Результаты сравнения приведены в таблице 1.

Критерий	Nirvana	Airflow	NiFi	PLynx
Версионирование графов	+	±	+	-
Цикличность графов	-	-	-	-
Ожидание сообщения	-	-	-	-
Опыт в программировании	±	+	±	+

Таблица 1: Сравнение существующих решений

В итоге из анализа конкурентов можно сделать следующие выводы:

- во всех инструментах пайплайн исполнения выражается ориентированным ациклическим графом, что не позволяет описывать автоматизации графами с циклами;
- ни в одном инструменте нет поддержки остановки автоматизации и продолжения работы с входными параметрами, полученными из внешнего вызова API;
- половина инструментов являются Python-ориентированными, что не позволяет создавать автоматизациями людям, не имеющих опыта в программировании;

Таким образом, во всех инструментах отсутствует часть требуемой функциональности, что влечет необходимость создания нового инструмента для управления автоматизациями.

3.3. Используемые технологиии

Kotlin [6] — язык программирования общего назначения, работающий поверх Java Virtual Machine. Исходя из требований единственной альтернативой был язык Java, однако Kotlin был выбран по причине большей лаконичности, встроенной поддержкой сопрограмм и более высокой типобезопасности, выражаемой явным разделением типов, объекты которых либо могут, либо не могут принимать значение null.

Spring Framework [12] — универсальный фреймворк, имеющий множество модулей. В рамках данной работы были использованы модули для

- работы разработки веб-сервисов;
- работы с базой данных;
- управления транзакциями из кода;
- аутентификации и авторизации.

. Среди имеющихся альтернатив, таких как Ktor [5], Micronaut [10], Helidon SE [4], Spring был выбран по следующим причинам:

- богатая функциональность, позволяющая сосредоточиться на написании бизнес-логики, а не шаблонных вещах, как, например, сериализация/десериализация тел HTTP запросов, конвертация исключений в соответствующие HTTP коды;
- имеет большое сообщество, что позволяет быстро решать возникающие проблемы.

PostgreSQL [11] — объектно-реляционная система управления базами данных. Выбор обоснован наличием инфраструктуры внутри компании для работы с этой СУБД.

jOOQ [15] — библиотека, позволяющая отобразить сущности базы данных на объекты Java. Является безопасной с точки зрения типов Java, а также обладает встроенным генератором кода, позволяющим сгенерировать шаблонный код по схеме базы данных. В таком случае построение запросов SQL выглядит как вызов методов сгенерированных объектов.

Flyway [9] — инструмент для осуществления изменения схемы базы данных. Миграции описываются с помощью скриптов SQL. Главный конкурент инструмента — liquibase [13], в котором миграции описываются файлами XML, а также отсутствует привязка к конкретной СУБД. Оба инструмента имеют поддержку со стороны Spring Framework. В силу того, что абстрагирование от конкретной СУБД не является необходимым в нашем случае, а также скрипты SQL легче читаются нежели файлы XML, выбор пал на Flyway.

4. Архитектура платформы

Сервис автоматизаций поделен на три ключевых модуля, взаимодействующих через программные интерфейсы, что обеспечивает слабую связность. Один из модулей был разработан другим членом команды, он отмечен темно-серым цветом на диаграмме. За запуск автоматизаций ответственен микросервис триггеров. Диаграмма коммуникации представлена на рисунке 1. Ниже приведен обзор компонентов системы и их взаимодействия друг с другом.

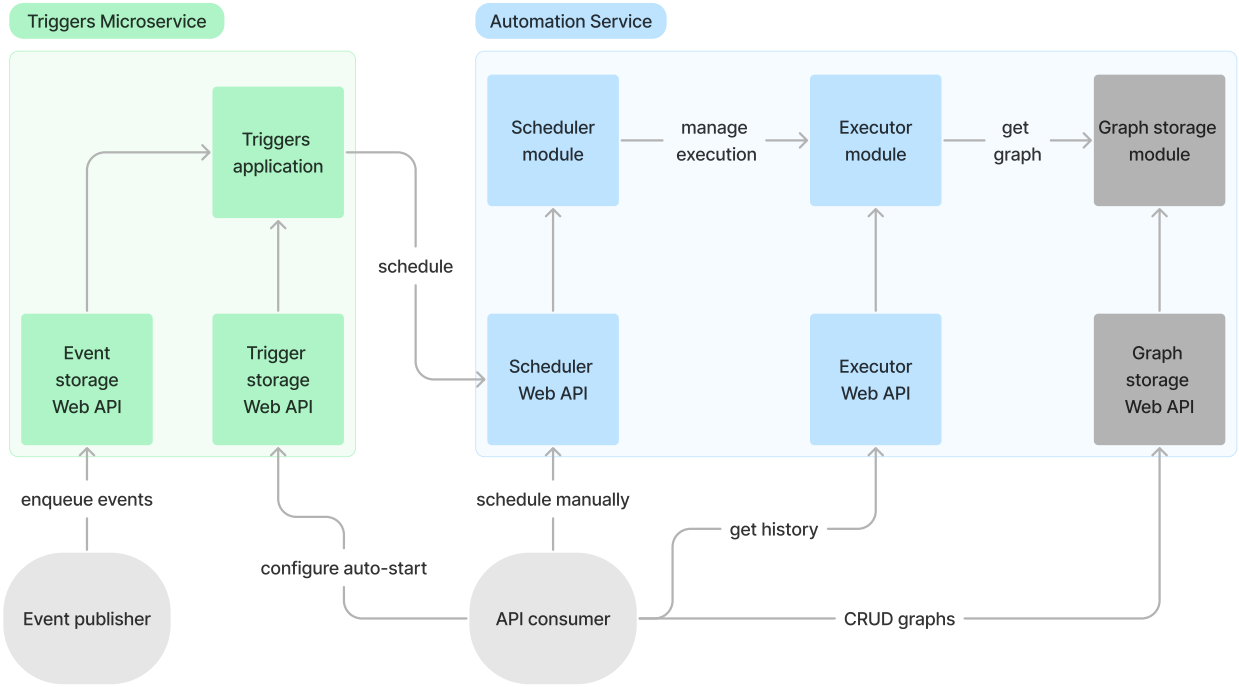

Рис. 1: Диаграмма коммуникации компонентов

Graph storage module — компонент хранения графов автоматизаций. Реализует CRUD-операции (Create, Read, Update, Delete) над графами.

Executor module — компонент исполнения графа автоматизации. Использует Graph storage module для получения нужной версии графа.

Scheduler module — компонент планирования автоматизаций. Использует Executor module для исполнения автоматизаций.

Triggers application — приложение триггеров. Его назначение — хранение и редактирование настроек, определяющих, при каких усло-

виях автоматизации должны быть запущены, и асинхронный запуск автоматизаций по поступающим событиям.

Event publisher — производитель событий. Отправляет события в микросервис триггеров.

API Consumer — абстрактный потребитель API, например, фронтенд или какой-то внутренний сервис. Имеет доступ к API хранилищ триггеров и графов, к API модуля исполнений, а также к API планировщика автоматизаций.

5. Реализация платформы

В этом разделе описаны особенности реализации компонент платформы.

5.1. Микросервис триггеров

Рассмотрим принцип работы микросервиса триггеров, представленный на рисунке 2.

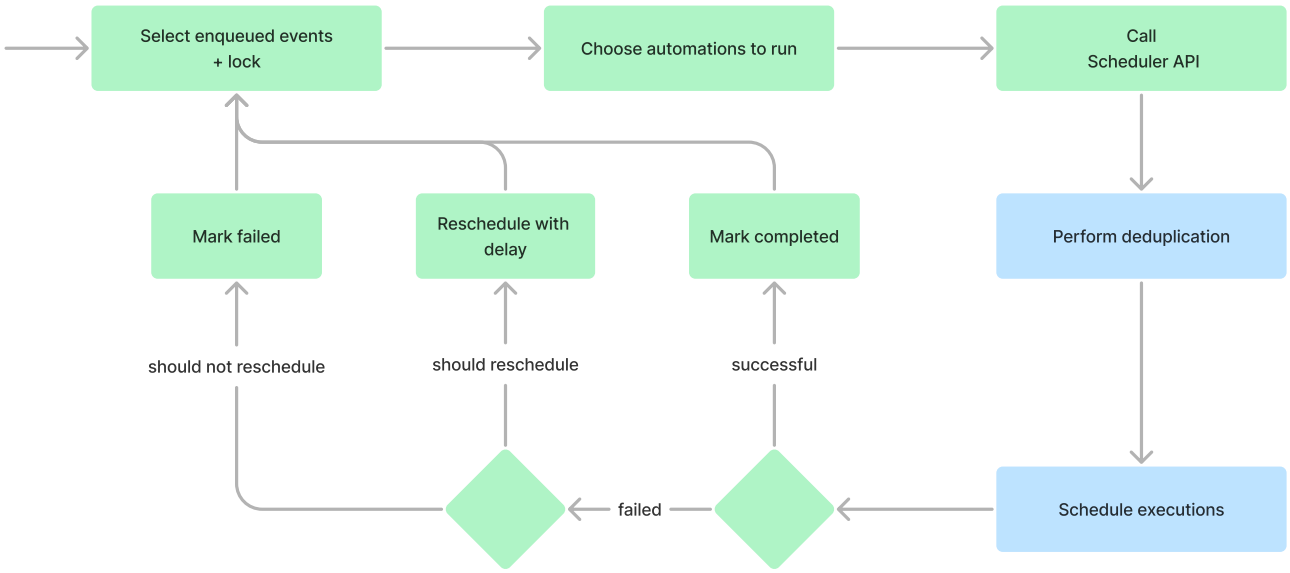

Рис. 2: Принцип работы микросевиса триггеров

В рамках одного экземпляра приложения есть несколько потоков, которые в цикле повторяют следующие действия:

1. взятие из базы данных необработанных событий вместе с блокировкой на них;
2. определение списка автоматизаций, которые нужно запустить по этим событиям;
3. вызов API планировщика с составленным запросом на планирование автоматизаций;

4. на стороне сервиса автоматизаций происходит дедупликация запросов по идентификаторам событий и запуск нужных автоматизаций;

5. если запрос был успешным, события помечаются обработанными, в противном случае последующие действия определяет политика перепланирования.

Стоит отметить, что описанная логика является потокобезопасной в силу того, что используется блокировки с помощью конструкции

```
select ... for update skip locked
```

5.2. Жизненный цикл исполнения

Жизненный цикл исполнения, представлен на рисунке 3. В диаграмме введены два термина:

1. **released** — исполнение не назначено ни на один экземпляр приложения;

2. **acquired** — исполнение назначено на какой-то экземпляр приложения.

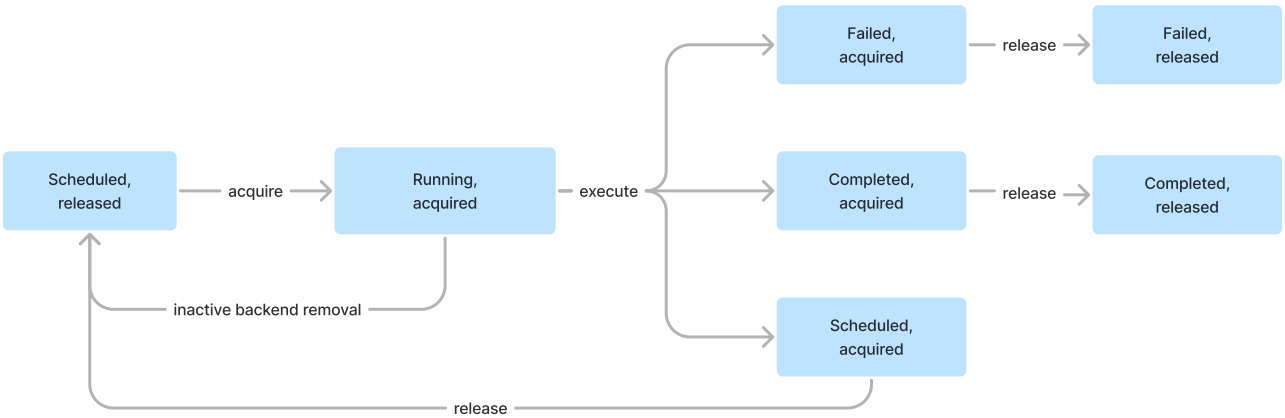

Рис. 3: Жизненный цикл исполнения

Также отметим, что все изменения статусов исполнения происходят с помощью оптимистических блокировок посредством ввода дополнительной колонки revision. Выбраны именно оптимистические блокировки, поскольку они позволяют избежать долгих транзакций, которые имели бы место в силу большого количества обращений к внешним системам в период исполнения автоматизации.

5.3. Модуль планирования

Принцип работы модуля планирования представлен на рисунке 4.

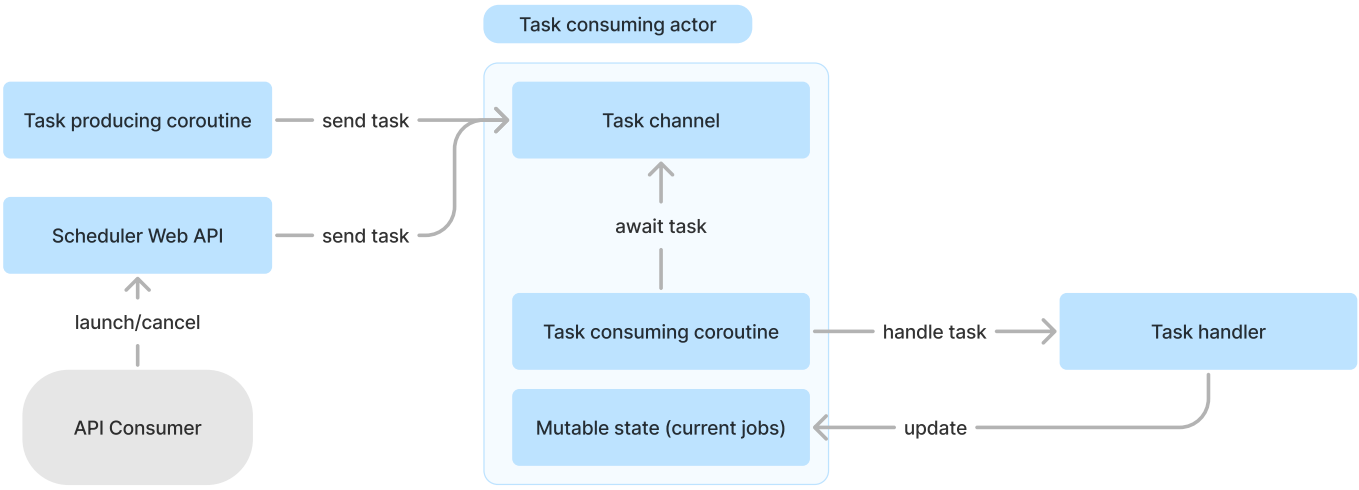

Рис. 4: Принцип работы модуля планирования

В основе модуля планирования лежит Task consuming actor, состоящий из:

1. очереди заданий;
2. сопрограммы, которая инкапсулирует изменяемое состояние и обрабатывает поступающие в очередь задачи с помощью Task handler.

Задачи в очередь могут поступать из:

1. сопрограммы, которая в цикле раз в определенный промежуток времени отправляет задачи;

2. внешнего потребителя API, который, например, сделал запрос на отмену исполнения автоматизации.

Такой подход позволяет избежать использования разделяемого изменяемого состояния, а также обеспечить асинхронность.

6. Апробация

В этом разделе описан процесс апробации серверной части платформы.

6.1. Апробация на тестовых данных

Задача серверной части платформы — исполнить автоматизацию после совершения пользователем действия, на которое настроен триггер. Поэтому в тестовом окружении для проверки корректности работы платформы были настроены автоматизации, совершающие простые действия, как например, изменение очереди обращения, а также триггеры на определенные события, запускающие эти автоматизации. Далее для начала всей цепочки действий совершалось действие в системе, для которого настроен триггер, а затем была проконтролирована корректность исполнения следующих этапов:

1. получение и сохранение события микросервисом триггеров;
2. отправка запроса на запуск нужных автоматизаций по необработанному событию;
3. дедупликация полученного запроса по идентификатору события;
4. исполнение автоматизации.

Корректное исполнение каждого этапа гарантирует корректную работу всей платформы.

6.2. Апробация на реальных данных

На данный момент в рамках разработанной платформы работает более ста автоматизаций, которые занимаются обработкой обращений пользователей, а именно автоматическими ответами на обращения, добавлением тегов, изменением очередей обращений, фильтрацией спама. Ежесуточно платформа обрабатывает более миллиона событий и производит более десятка тысяч исполнений автоматизаций.

Заключение

В ходе выполнения данной работы были выполнены все поставленные задачи:

- сформулированы требования к создаваевому продукту;
- произведен обзор существующих инструментов для создания автоматизаций;
- спроектирована архитектура системы;
- реализованы компоненты системы;
- произведена апробация.

Список литературы

[1] Apache. Apache Airflow. — https://airflow.apache.org/. — 2022. — (дата обращения: 29.11.2022).

[2] Apache. Apache NiFi. — https://nifi.apache.org/. — 2022. — (дата обращения: 29.11.2022).

[3] Apache. The Apache groovy programming language. — https://groovy-lang.org/. — 2022. — (дата обращения: 29.11.2022).

[4] Helidon. Helidon Project. — https://helidon.io/. — 2022. — (дата обращения: 29.11.2022).

[5] Jetbrains. Create asynchronous client and server applications. — https://ktor.io/. — 2022. — (дата обращения: 29.11.2022).

[6] Jetbrains. Kotlin programming language. — https://kotlinlang.org/. — 2022. — (дата обращения: 29.11.2022).

[7] PLynx-team. PLynx. — https://plynx.com/. — 2022. — (дата обращения: 29.11.2022).

[8] Quixy. Workflow Automation Statistics. — https://quixy.com/blog/workflow-automation-statistics-and-forecasts/. — 2022. — (дата обращения: 29.11.2022).

[9] Redgate. Increase reliability of deployments by versioning your database. — https://flywaydb.org/. — 2022. — (дата обращения: 29.11.2022).

[10] Rocher Graeme. Micronaut framework. — https://micronaut.io/. — 2022. — (дата обращения: 29.11.2022).

[11] Stonebraker Michael. PostgreSQL: The World's Most Advanced Open Source Relational Database. — https://postgresql.org/. — 2022. — (дата обращения: 29.11.2022).

[12] Tanzu VMWare. Spring framework. — https://spring.io/. — 2022. — (дата обращения: 29.11.2022).

[13] Voxland Nathan. Fast database change. Fluid delivery. — https://liquibase.org/. — 2022. — (дата обращения: 29.11.2022).

[14] YAML. The Official YAML Web Site. — https://yaml.org/. — 2022. — (дата обращения: 29.11.2022).

[15] jOOQ. jOOQ: The easiest way to write SQL in Java. — https://jooq.org/. — 2022. — (дата обращения: 29.11.2022).