Факультет
компьютерных наук

Программная
Инженерия

Москва
2025

Приложение для агрегации финансовых данных

Financial Data Aggregator Application

Хадзакос Николай
БПИ236

Под руководством доцента департамента программной инженерии ,
заведующего научно-учебной лабораторией процессно-ориентированных
информационных систем (ПОИС), PhD
Нестерова Романа Александровича

Назначение и предметная область

Разрабатываемое приложение предназначено для объединения данных из различных биржевых API. Это дает пользователям возможность получать оперативную информацию о состоянии рынков и отдельных активов в реальном времени с минимальными задержками.

Программа будет применима в области финансового рынка в целях поиска актуальной информации по популярным и необходимым активам, которые торгуются на тех или иных рынках.

Актуальность проекта

35,1 млн

Количество частных инвесторов, имеющих брокерские счета на Московской бирже. За 2024 год **увеличилось на 5,4 млн**, ими открыто более 64,3 млн счетов (**+12 млн за 2024 год**).

1,3 трлн

Вложили в ценные бумаги в 2024 году частные инвесторы на Московской бирже **(+18% по сравнению с 2023 годом).**

74%

Составила доля частных инвесторов в объеме торгов акциями в 2024 году.

Источник

Цель и задачи проекта

Цель:
Исследование и экспериментальная реализация параллельных алгоритмов отработки больших потоков данных.

Задачи:
- Организовать и реализовать масштабируемую архитектуру для удобного добавления новых источников данных.
- Реализовать надежный и быстрый алгоритм балансировки нагрузок в сервисе обработки данных, для избежания ситуаций переполнения очереди сообщений и записи неактуальных данных.

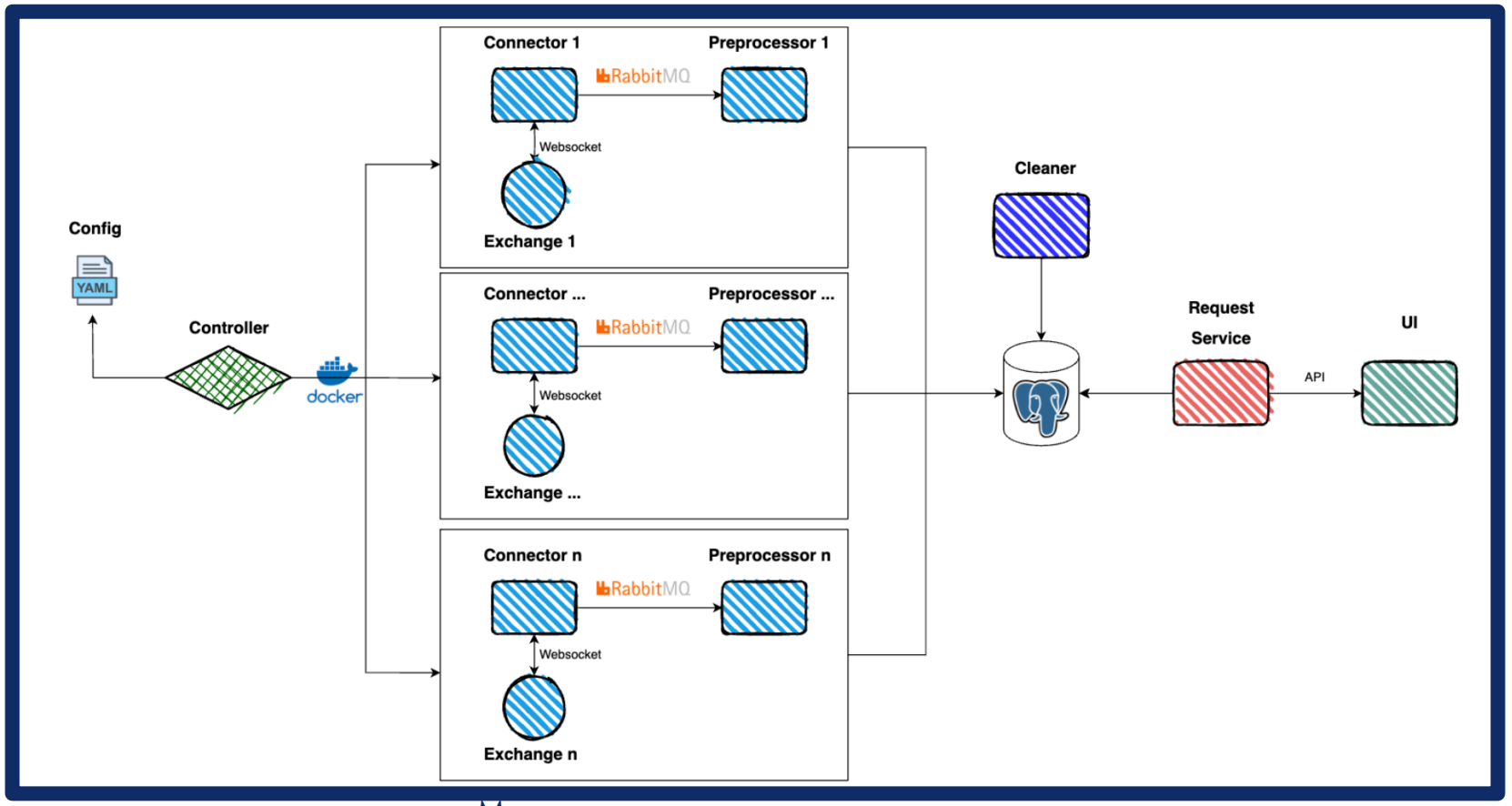
Что по аналогам?
В чем преимущество разработки?

Сервис	Цена	Тип данных	Частота обновления данных	Сводная информация	Open-source
Heist Finance	Бесплатно	Акции, криптовалюты	Live-time	Да	Да
TradingView	Freemium	Акции, криптовалюты, индексы, форекс	Live-time	Нет	Нет
Finviz	Бесплатно	Активы американского рынка, криптовалюты	Каждые 30 секунд	Да	Нет
Т-Инвестиции	Бесплатно, при наличии брокерского счета	Активы российского рынка	Live-time	Да	Нет

Функционал

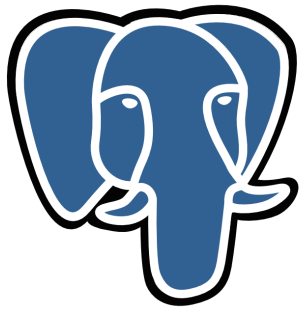
1. Предоставление функции поиска по активам.
2. Сбор текущей информации о активах(цена, объем торгов и т.д.) с частотой получения данных с бирж.
3. Сбор исторических данных по активам.
4. Предоставление данных для графического отображения данных о активе.
5. Предоставление сводной информации(цена, объем торгов, график и т.д.) по популярным активам для зрительного анализа.
6. Предоставление информации о всех активах, торгующихся на бирже.
7. Предоставление сводной информации о популярных активах по каждой бирже.
8. Предоставление информации для подробного описания отдельного актива и графика изменения его цены.

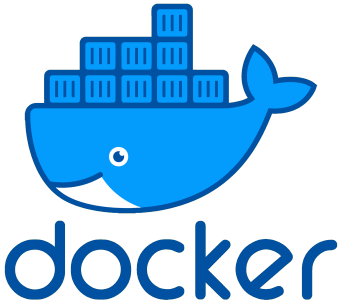
Архитектура приложения



Микросервисная архитектура

Технологии проекта





Средства реализации

Golang

Язык Golang является идеальным языком для создания приложений с микросервисной архитектурой. Язык также славится встроенными инструментами для параллельной работы с данными, а его скорость и, в тоже время, простота позволяет выполнить полноценно поставленную задачу.

Docker

Для простого развертывания отдельных сервисов приложения и всего приложения в целом был выбран Docker, как самый популярный и, в то же время, простой инструмент для создания микросервисов. Его API позволяет сделать систему масштабируемой, к чему мы и стремились.

RabbitMQ

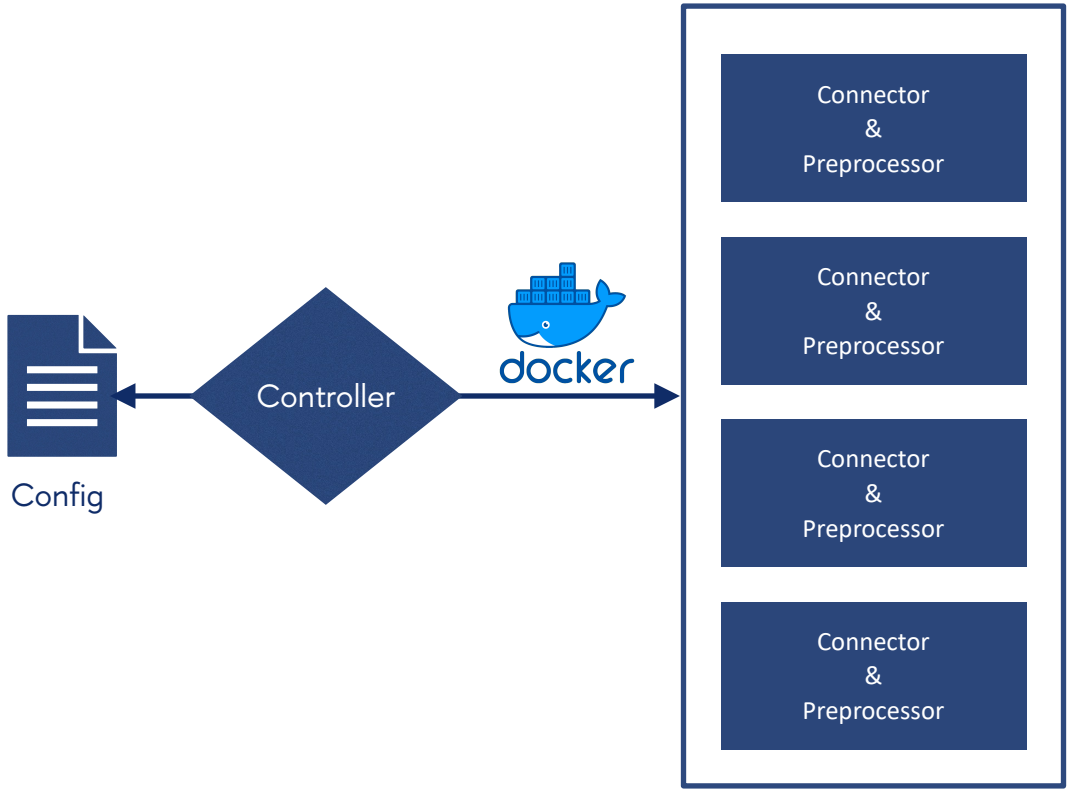
Для общения двух микрсосервисов через очередь сообщений был выбран RabbitMQ, как более легковесная версия популярной Kafka. В нашем приложении нет необходимости использовать всю мощь Kafka, поэтому нам достаточно базового функционала очереди сообщений, который как раз и предоставляет RabbitMQ.

Описание разработанных сервисов и алгоритмов

Controller

Является сердцем сборки и обработки данных. Сервис отвечает за управление экземплярами сервисов Connector и Preprocessor, которые и выполняют всю работу над данными.

Исходя из текущего состояния конфигурационного файла, он поднимает пару вышеперечисленных сервисов, используя для этого технологию контейнеризации - Docker. Для этого используется API Docker.



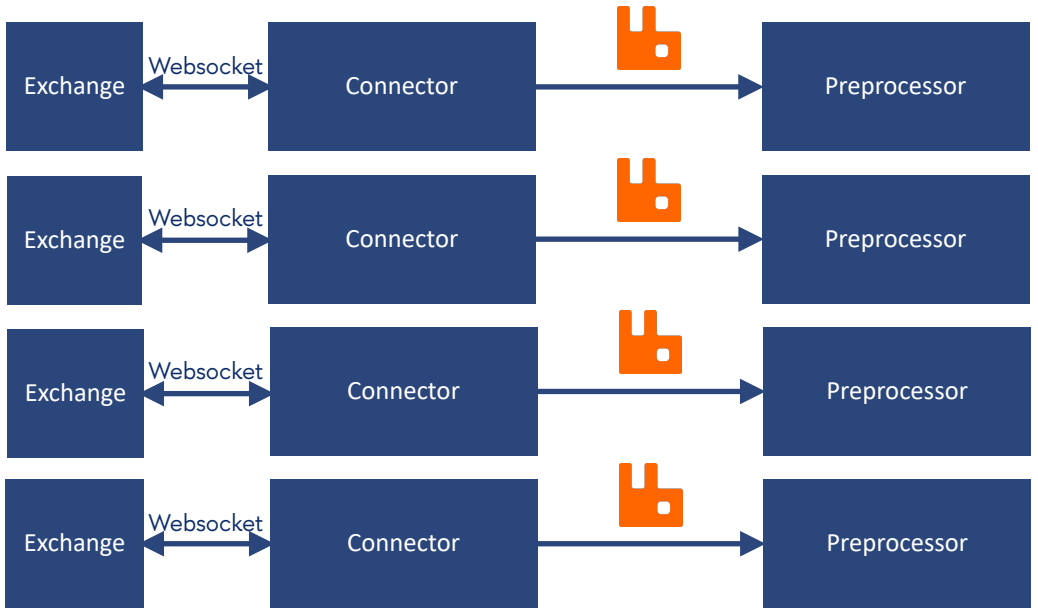
Connector и Preprocessor

Экземпляры сервиса Connector подключаются к биржам через Websocket Stream. Через такой поток передаются все анонимные сделки отдельной биржи. Тем самым, мы получаем актуальные данные с минимальными задержками.

Чтобы не нагружать Connector обработкой данных, он передает в очередь сообщений - RabbitMQ - сырые данные с полученной информацией, которая приводится к единому формату в экземпляре сервиса Preprocessor. Чтобы задержки были минимальные, а очередь не забивалась, было принято решение написать балансировщик нагрузок на уровне предобработки данных - Preprocessor.

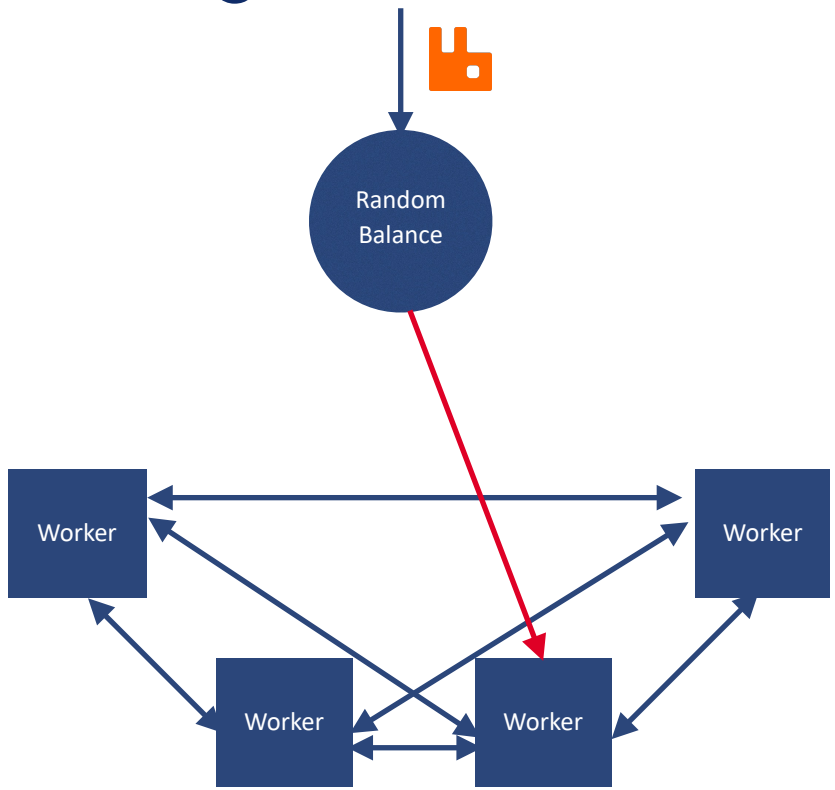
После обработки, данные записываются в соответствующие таблицы базы данных.

Все эти пары экземпляров работают параллельно на уровне отдельных контейнеров. О безопасной параллельной работе с базой данной рассказано на слайде с архитектурой базы данных.



Балансировка нагрузок - Work Stealing

Поступающие из очереди сообщений данные распределяются случайным образом. Если данные порождаются **равномерным распределением**, то с ростом объёма выборки эмпирические частоты будут **приближаться к равномерному распределению**. Обработчики содержат в себе очередь ограниченного размера, для того, чтобы обрабатывать данные наборами, а не по одному. Наборы с задачами распределяются случайно. Наш алгоритм будет улучшать неравномерность случайного выбора, путем кражи задач - самобалансировки. Обработчики работают параллельно, на уровне горутин - легковесный поток, если же у него закончились задачи, то он забирает их у более загруженных обработчиков. Тем самым обработчики всегда выполняют какую-то задачую, разгружая более загруженные обработчки или обрабатывая данные. Было замечено, что чем дольше работают обработчики, тем лучше распределены наборы данных между ними. Данный алгоритм является улучшением алгоритма балансировки Random Round Robbin.

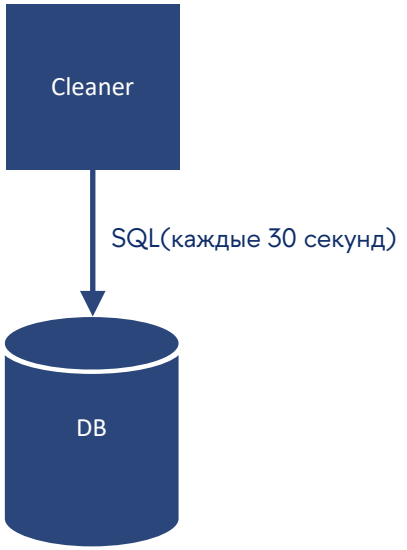


- случайное распределение наборы задач

- steal (кража задач)

Cleaner

Сервис выполняет простую, но очень важную задачу - очистка устаревших данных. Так как live-time данные поступают в базу данных в огромном количестве и ежесекундно, то довольно таки быстро запросы взятия данных начнут подвисать, а память будет израсходоваться впустую. Так как мы разделяем исторические данные и live-time данные, то каждые 30 секунд происходит чистка live-time данных старше 5 минут. Это параллельный процесс, о том как безопасно производятся эти операции рассказано на следующем слайде.

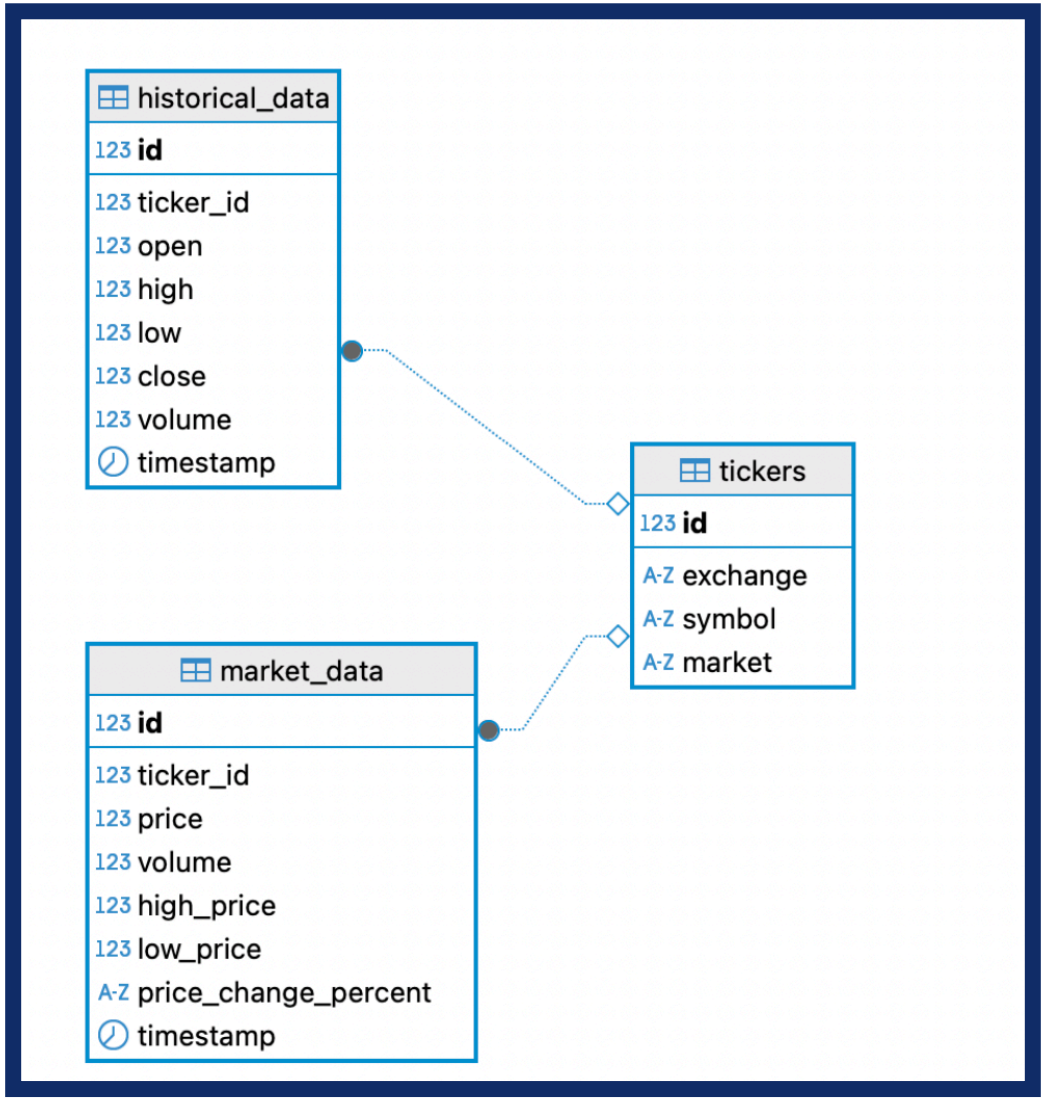


Архитектура базы данных

В качестве базы данных используется PostgreSQL. PostgreSQL - это популярный инструмент хранения данных. В случае временных рядов можно было выбрать и другие виды баз данных, однако, для нашей задачи было достаточно простых таблиц, в которых мы создаем отношение на определенный тикер актива и сохраняем необходимые данные о данном активе.

market_data - таблица, в которой хранятся live-time данные.
historical_data - семейство таблиц, в которых хранятся исторические данные в зависимости от промежутка записей этих данных.

Для безопасной записи и чтения используется пул подключений, который выполняет все операции с базой данных атомарно, а также дает возможность подключать к одному ресурсу множество экземпляров одинаковых объектов. Как раз наличие такого простого для интеграции инструмента также определило выбор базы данных.



Демонстрация

Основные результаты и выводы

- Разработано приложение для агрегации финансовых данных, которое автоматизирует сбор, обработку и предоставление актуальных данных о котировках финансовых активов, делая это параллельно.
- Приложение имеет микросервисную архитектуру. Это обеспечивает масштабируемость и возможность добавления новых бирж и функций за счет создания экземпляров шаблонных сервисов и их обновления.
- Для балансировки нагрузки между обработчиками данных применяется самобалансирующийся алгоритм Work Stealing, который обеспечивает надежность, скорость и эффективность обработки данных. Тем самым получилось избавиться от узкого горлышка в обработке данных.
- Устаревшие данные очищаются, тем самым оптимизируя запросы к базе данных.
- Получилось достигнуть отказоустойчивости в нормальных условиях работы приложения(около 1 миллиона записей в таблицу в минуту).

Направление дальнейшей работы

- Добавление продвинутой аналитики для активов(метрики, дополнительные построения на графиках, **построение графа для обнаружения арбитражных сделок**).
- Добавление новых бирж и видов активов(ETF, облигации, форекс).
- Подготовка программы для полноценного self-host развертывания.

Список использованных источников

1. Golang [Электронный ресурс] URL: https://go.dev/ (Дата обращения: 06.11.2024)
2. PostgreSQL [Электронный ресурс]. URL: https://www.postgresql.org (Дата обращения: 20.11.2024)
3. Docker [Электронный ресурс]. URL: https://www.docker.com (Дата обращения: 27.11.2024)
4. RabbitMQ [Электронный ресурс]. URL: https://www.rabbitmq.com/ (Дата обращения: 27.11.2024)
5. pgxpool package [Электронный ресурс]. URL: https://pkg.go.dev/github.com/jackc/pgx/v4/pgxpool (Дата обращения: 19.02.2025)


ВЫСШАЯ • ШКОЛА • ЭКОНОМИКИ •