|  |  |
| --- | --- |
| Министерство науки и высшего образования РФ  Федеральное государственное автономное образовательное учреждение высшего образования «Пермский государственный национальный исследовательский университет» | |
|  | Кафедра информационных технологий |
|  | |
| **ОТЧЕТ**  по лабораторной работе №4 «Разработка распределенного приложения для локальной сети»  по дисциплине «Технологии разработки распределенных приложений» | |
|  | Работу выполнили студенты  группы *ФИТ-2-17 (ММ/О-8) 4* курса  механико-математического факультета  *Габов Егор Владимирович*  *Менькина Дарья Александровна*  *Рыжкова Елена Сергеевна*  Проверил:  доцент кафедры математического обеспечения вычислительных систем механико-математического факультета,  к.ф-м.н., доц. *Деменев Алексей Геннадьевич* |

|  |
| --- |
| Пермь 2021 |

**Оглавление**

[1 Введение 3](#_Toc66897606)

[1.1 Цель 3](#_Toc66897607)

[1.2 Формируемые компетенции 3](#_Toc66897608)

[1.3 Организация выполнения работы 3](#_Toc66897609)

[2 Основная часть 4](#_Toc66897610)

[2.1 Постановка задачи 4](#_Toc66897611)

[2.2 Используемые технологии 6](#_Toc66897612)

[2.3 Описание программы 7](#_Toc66897613)

[2.3.1 Введение в предметную область 7](#_Toc66897614)

[2.3.2 Описание работы распределенного приложения 8](#_Toc66897615)

[2.4 Разработка приложения 11](#_Toc66897616)

[2.4.1 Пользователь 11](#_Toc66897617)

[2.4.2 Исполнитель 12](#_Toc66897618)

[2.4.3 Контроллер 14](#_Toc66897619)

[2.5 Сценарии использования 15](#_Toc66897620)

[2.6 Тестирование 16](#_Toc66897621)

[3 Заключение 20](#_Toc66897622)

[Библиографический список 22](#_Toc66897623)

# Введение

## 1.1 Цель

Изучение методов коммуникации процессов в сети, а также средств динамического конфигурирования распределенных приложений.

## 1.2 Формируемые компетенции

Способность работы с информацией из различных источников, включая сетевые ресурсы сети Интернет, для решения профессиональных задач; способность применять на практике теоретические основы и общие принципы разработки распределенных систем; уверенное знание теоретических и практических основ построения распределенных баз данных; способность использовать на практике стандарты сетевого взаимодействия компонент распределенной системы.

## 1.3 Организация выполнения работы

Задание выполняется в группе (не более трех студентов) или индивидуально.

# 2 Основная часть

## 2.1 Постановка задачи

Ниже приведены требования к выполнению работы.

1. Приложение должно обеспечивать параллельную работу нескольких клиентов и серверов. Дополнительное требование: возможность запуска нескольких серверов на одном компьютере.
2. Клиентские приложения должны автоматически находить серверы для обслуживания и выполнения заданных функций.
3. Серверы системы могут выполнять различные функции.
4. При разрыве сеанса приложения должны автоматически восстанавливать свою работоспособность.
5. Для хранения данных и доступа к ним применить ADO и/или ADO.NET (или их аналоги).
6. Приложения должны поддерживать возможность взаимодействия в различных режимах.
7. Для организации взаимодействия нужно использовать различные средства коммуникации (именованные каналы, мейлслоты, сокеты, MSMQ, .Net Remoting, web-сервисы, WCF-сервисы), сравнив их возможности.

В программе необходимо использовать открытые данные.

По окончании выполнения задания необходимо подготовить отчёт. Отчет по выполнению задания должен включать:

1. Общее описание приложения. Постановка задачи, введение в предметную область.
2. Архитектура системы. Обоснование выбора данного типа архитектуры распределенного приложения. Алгоритм работы приложения в целом.
3. Архитектура каждого из логических компонент системы (серверы, клиенты, диспетчеры). Подходы к реализации. Алгоритмы работы. Многопоточность, обоснование.
4. Методы коммуникаций компонентов системы (клиент → сервер, сервер → клиент и т.д.). Обоснование выбора этих методов коммуникации.
5. Способ передачи данных (синхронная / асинхронная, однонаправленная / двунаправленная и т.д.). Обоснование.
6. Структура передаваемых данных. Вид протоколов, обоснование выбора.
7. Отказоустойчивость системы. Как система поведет себя, если «исчезнет» один или несколько ее компонент. Что произойдет с системой, если «исчезнувший» компонент будет восстановлен на другом узле сети.
8. Работа с базой данных (если используется). Обоснование.
9. Исходный код приложений с комментариями.

Таблица 1 – Требования к заданию и распределение баллов

|  |  |
| --- | --- |
| **Требования к заданию** | **Максимальное количество баллов** |
| Распределенное приложение спроектировано с учетом особенностей предметной области. Выбрана наиболее подходящая модель распределенной системы. Если используется распределенная база данных, то тиражирование данных имеет подходящий для данной предметной области механизм и архитектуру. | 3 |
| Архитектура системы является оптимальной для заданных при разработке критериев. В отчете присутствует обоснование выбора данного типа архитектуры. | 3 |
| Приложение обеспечивает параллельную работу нескольких клиентов и серверов, в том числе на одном компьютере. Серверы распределенной системы выполняют различные функции. | 2 |
| Приложение является масштабируемым, позволяет добавлять новых участников взаимодействия без переписывания кода и перезапуска приложений. | 2 |
| Существует возможность динамического реконфигурирования системы. | 4 |
| Для организации взаимодействия компонент распределенной системы используется не менее четырех различных средств коммуникации. В отчете присутствует четкое обоснование выбора средств взаимодействия для каждого конкретного случая. | 4 |
| Система является отказоустойчивой. В случае если один и/или несколько компонент системы аварийно завершают свою работу. | 2 |
| Распределенное приложение продолжает работать и в случае, если после аварийного завершения некоторого компонента, он восстановлен на другом узле вычислительной сети. | 3 |
| Отчет содержит подробное описание архитектуры каждого компонента распределенного приложения. | 3 |
| В отчете описана структура передаваемых данных, формат сообщений и вид протокола, используемого для этого. | 2 |
| В отчете представлено описание способа передачи сообщений при коммуникации компонентов распределенной системы с обоснованием. | 2 |

## 2.2 Используемые технологии

Для разработки серверного и клиентского приложений был выбран объектно-ориентированный язык программирования Python 3.8 [1], распространяемый свободно [2]. В качестве среды разработки была выбрана JetBrains PyCharm Community Edition 2019.2.3 [3, 4], распространяемой по Apache [5].

## 2.3 Описание программы

### 2.3.1 Введение в предметную область

Предметная область – получение исторический значений курса акций.

В августе 2020 года на Мосбирже частные клиенты открыли 515 500 новых брокерских счетов. Это исторический рекорд для российского фондового рынка. На данный момент на бирже зарегистрировано уже 6,2 млн частных инвесторов (то есть обычных физлиц) — на 2,4 млн больше, чем в январе 2020-го. В связи с этим можно сказать, что актуальной стала проблема получения исторических и текущих данных по курсу финансовых инструментов, это позволит проводить анализ волатильности и других показателей, влияющих на оценку риска и экономической целесообразности инвестирования.

Разрабатываемое приложение будет позволять оперативно получать курс акции на любую дату или за любой интервал дат (до последнего доступного дня), что может послужить развитию выгодных розничных инвестиций на Московской бирже.

Особенностью предметной области является потребность в получении исторических данных, что учитывалось при проектировании распределенного приложения.

При проектировании распределенного приложения были заданы следующие критерии: приложение должно быть **отказоустойчивым** и **не нагружать** клиентскую машину. Приложение отказоустойчиво за счет наличия программы-*Контроллера* и масштабируемо в ширину за счет того, что программы *Сервера*, *Контроллера* и *Клиента* можно запускать на нескольких рабочих машинах. Кроме того, приложение позволяет получать пользователю данные без излишней нагрузки на компьютер клиента, все ресурсоемкие операции производятся на сервере.

Исходя из вышеперечисленного можно сказать, что выбранная архитектура является оптимальной, так как критерии были выполнены, а также был достигнут критерий масштабируемости, который не был задан изначально.

### 2.3.2 Описание работы распределенного приложения

Описать работу распределенного приложения можно следующим образом:

1. Пользователем формируется и посылается запрос
2. Запрос пользователя принимается «Исполнителем» (Server).
3. «Исполнителем» посылается запрос к источнику открытых данных (www.moex.com) [6].
4. Получив результат от источника данных, «Исполнитель» отправляет результат пользователю.
5. В случае, если «Исполнитель» (Server) аварийно завершился «Контроллер» (Controller) его запускает.

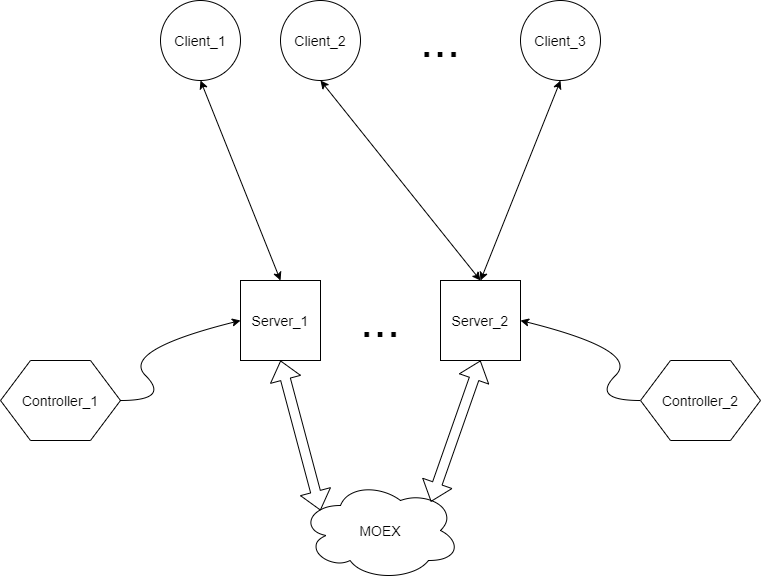


Рисунок 1 – Схема архитектуры распределенного приложения (Клиент - Сервер)

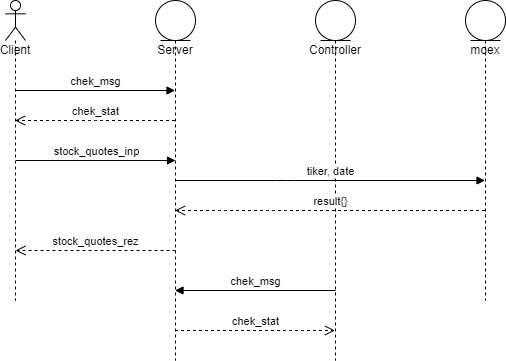


Рисунок 2 – Схема работы (диаграмма последовательностей UML) распределенного приложения

Веб-сервис реализуется с помощью «gRPC» [7], распространяемой по Apache [5] – фреймворка с открытым исходным кодом, созданного «Google». В качестве языка описания web-сервиса используется «Protocol Buffers (Protobuf)» – язык описания интерфейсов (IDL) [8], который подразумевает определение сервиса через указание методов, которые могут быть вызваны удалённо, с их параметрами и типами принимаемых и возвращаемых данных. Это осуществляется с помощью «протофайла» – обычного текстового файла с расширением «.proto». В этом файле описываются «сообщения» («message») – объекты, содержащие набор полей, представляющих собой пары {тип данных – имя}. Также в «протофайле» описываются сервисы, каждый из которых содержит набор методов с описанием типа принимаемых и возвращаемых данных. После составления «протофайла» на его основе с помощью инструмента gRPC создаются автогенерируемые классы для работы с описанными структурами данных и методами, которые будут использоваться в серверных и клиентских приложениях. gRPC обусловлен простотой реализации, распараллеливанием работы сервера для нескольких клиентов. moex передает данные по http в формате json по http, в связи с чем используется http.

Был сформирован следующий «протофайл»:

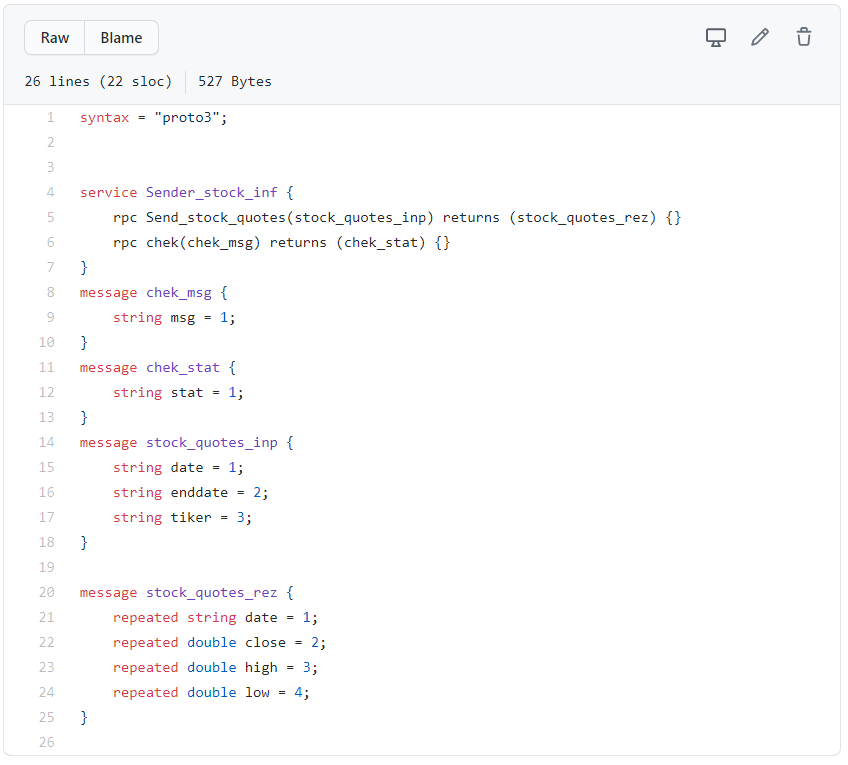


Рисунок 3 – Содержимое файла serv.proto

Таким образом, было спроектировано четыре объекта:

1. «chek\_msg» с одним текстовым полем (string msg) – для отправки серверу запроса на проверку статуса его работоспособности. Будет формироваться клиентским приложением и посылаться серверу.
2. «chek\_stat» с одним текстовым полем (string stat) – для отправки клиенту ответа на запрос о проверке статуса работоспособности сервера. Будет формироваться сервером приложением и посылаться клиентскому приложению.
3. «stock\_quotes\_inp» с тремя текстовыми полями (string date, string enddate, string tiker) – для отправки даты начала и даты конца периода для просмотра стоимости акций, а также идентификатора акции соответственно.
4. «stock\_quotes\_rez» с одним массивом текстовых полей (repeated string date) – для отправки дат, которые входят в промежуток, и трех числовых массивов (repeated double close, repeated double high, repeated double low) для отправки стоимости закрытия акции, наибольшей и наименьшей стоимостей.

## 2.4 Разработка приложения

### 2.4.1 Пользователь

В качестве клиентского приложения используется клиентская программа, запущенная на компьютере пользователя. «Пользователь» отправляет сообщение «chek\_msg» методом «chek» поочередно на несколько портов, на которых работают «Исполнители».

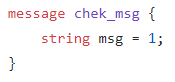


Рисунок 4 – Сообщение «chek\_msg»

Это происходит до тех пор, пока «Пользователь» не получит сообщение «chek\_stat» от любого «Исполнителя».

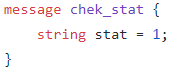


Рисунок 5 – Сообщение «chek\_stat»

После получения «chek\_stat» «Пользователь» записывает порт, с которого это сообщение было получено, т.е. тот, на котором находится «Исполнитель», и начинает с работать с ним, все последующие запросы отправляются на этот порт. Если при отправке последующих запросов «Пользователь» не получает ответ или ему приходит ошибка, он начинает процедуру поиска «Исполнителя» сначала, то есть отправляет сообщение «chek\_msg» до тех пор, пока не будет получено сообщение «chek\_stat».

После связывания с «Исполнителем» «Пользователь» формирует и отправляет запрос «stock\_quotes\_inp» «Исполнителю» методом Send\_stock\_quotes().

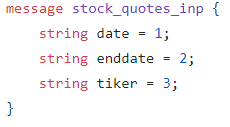


Рисунок 6 – Сообщение «stock\_quotes\_inp»

После этого он ждет ответа от «Исполнителя». В качестве ответа «Пользователь» получает сообщение «stock\_quotes\_rez».

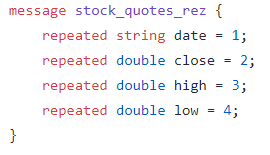


Рисунок 7 – Сообщение «stock\_quotes\_rez»

Полученные данные выводятся на экран.

### 2.4.2 Исполнитель

«Исполнитель» производит Get-запрос к открытому API Московской биржи, позволяет получать данные о стоимости акций на различные даты, по различными инструментам.

«Исполнитель» создает экземпляр класса «Sender\_stock\_inf» для каждого «Пользователя», который производит запрос, позволяет одновременно нескольким «Пользователям» подключаться к одному «Исполнителю» и работать параллельно.

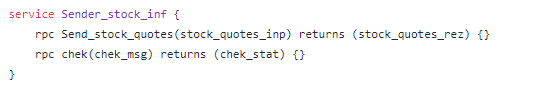


Рисунок 8 – Сообщение «Sender\_stock\_inf»

Get-запрос к moex:

http://iss.moex.com/iss/history/engines/stock/markets/shares/boards/TQBR/securities/"{tiker}".json?iss.json=extended&from={date}

В приведенном выше запросе tiker – идентификатор финансового инструмента, date – дата, с которой начинается промежуток выборки данных.

В результате получаем для каждого дня в указанном промежутке, начиная с date до текущего дня, json следующей структуры:

|  |
| --- |
| {  "BOARDID": {"type": "string", "bytes": 12, "max\_size": 0},  "TRADEDATE": {"type": "date", "bytes": 10, "max\_size": 0},  "SHORTNAME": {"type": "string", "bytes": 189, "max\_size": 0},  "SECID": {"type": "string", "bytes": 36, "max\_size": 0},  "NUMTRADES": {"type": "double"},  "VALUE": {"type": "double"},  "OPEN": {"type": "double"},  "LOW": {"type": "double"},  "HIGH": {"type": "double"},  "LEGALCLOSEPRICE": {"type": "double"},  "WAPRICE": {"type": "double"},  "CLOSE": {"type": "double"},  "VOLUME": {"type": "double"},  "MARKETPRICE2": {"type": "double"},  "MARKETPRICE3": {"type": "double"},  "ADMITTEDQUOTE": {"type": "double"},  "MP2VALTRD": {"type": "double"},  "MARKETPRICE3TRADESVALUE": {"type": "double"},  "ADMITTEDVALUE": {"type": "double"},  "WAVAL": {"type": "double"},  "TRADINGSESSION": {"type": "int32"}  } |

OPEN – цена акции во время открытия данного торгового дня на бирже,

LOW – самая низкая цена акции в течении дня на бирже,

HIGH – самая высокая цена акции в течении дня на бирже,

TRADEDATE – дата, за которую отображаются показатели

Пример Get - запроса:

http://iss.moex.com/iss/history/engines/stock/markets/shares/boards/TQBR/securities/"SBER".json?iss.json=extended&from=2020-12-15

Полученный json объект:

|  |
| --- |
| {  "BOARDID": "TQBR",  "TRADEDATE": "2020-12-15",  "SHORTNAME": "Сбербанк",  "SECID": "SBER",  "NUMTRADES": 148863,  "VALUE": 23511052595.1,  "OPEN": 278.53,  "LOW": 276.07,  "HIGH": 283.76,  "LEGALCLOSEPRICE": 276.47,  "WAPRICE": 279.23,  "CLOSE": 278.7,  "VOLUME": 84193230,  "MARKETPRICE2": 279.35,  "MARKETPRICE3": 279.35,  "ADMITTEDQUOTE": 276.47,  "MP2VALTRD": 22187030589.1,  "MARKETPRICE3TRADESVALUE": 22187030589.1,  "ADMITTEDVALUE": 22187030589.1,  "WAVAL": 0,  "TRADINGSESSION": 3  } |

### 2.4.3 Контроллер

«Контроллер» с помощью gRPC запроса и метода chek(), отправляет «Исполнителю» сообщение «chek\_msg».

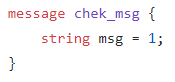


Рисунок 9 – Сообщение «chek\_msg»

«Контроллер» ожидает ответа от «Исполнителя» и получения сообщения «chek\_stat».

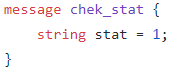


Рисунок 10 – Сообщение «chek\_stat»

Если «Контроллер» не получает статус «Исполнителя», то производит запуск экземпляра «Исполнитель», если же нет, то завершается. Данная процедура производится каждые несколько минут, поочередно. Это позволяет добиться отказоустойчивости, постоянной работы хотя бы одного экземпляра «Исполнителя».

## 2.5 Сценарии использования

#### **2.5.1 Сценарий №1**

Предусловия: отсутствуют запущенные и доступные экземпляры модуля «Исполнитель».

Последовательность действий:

1. Формирование и отправка запроса пользователем
2. Ожидание получения ответа

Ожидаемый результат: пользователь ожидает запуск исполнителя и получения ответа от него.

#### **2.5.2 Сценарий №2**

Предусловие: запущен и доступен хотя бы один экземпляр модуля «Исполнитель».

Последовательность действий:

1. Формирование и отправка запроса пользователем
2. Ожидание получения ответа

Ожидаемый результат: получен ответ в виде соответствующей структуры.

#### **2.5.3 Сценарий №3**

Предусловия: отсутствуют запущенные и доступные экземпляры модуля «Исполнитель».

Последовательность действий:

1. Модуль «Контроллер» производит запуск модуля «Исполнитель»
2. «Исполнитель» запущен и ожидает запрос пользователя

Ожидаемый результат: исполнитель ожидает запрос пользователя.

#### **2.5.4 Сценарий №4**

Предусловие: запущен и доступен хотя бы один экземпляр модуля «Исполнитель».

Последовательность действий:

1. Формирование и отправка запроса пользователем
2. Ожидание получения ответа
3. Подключение и отправка запроса вторым пользователем
4. Ожидание получения ответа

Ожидаемый результат: получен ответ в виде соответствующей структуры в обоих случаях.

#### **2.5.5 Сценарий №5**

Предусловие: запущен и доступен хотя бы один экземпляр модуля «Исполнитель» и к нему подключен хотя бы один пользователь.

Последовательность действий:

1. Подключение пользователя к модулю «Исполнитель»
2. Аварийное завершение модуля «Исполнитель»
3. Модуль «Контроллер» производит запуск модуля «Исполнитель» на другом узле
4. «Исполнитель» запущен и ожидает запрос пользователя

Ожидаемый результат: получен ответ в виде соответствующей структуры.

## 2.6 Тестирование

Ниже, на рисунке 18 представлен график изменений цен на акцию Сбербанка с официального сайта [6] на 12 марта 2021 года для проверки корректности работы приложения.

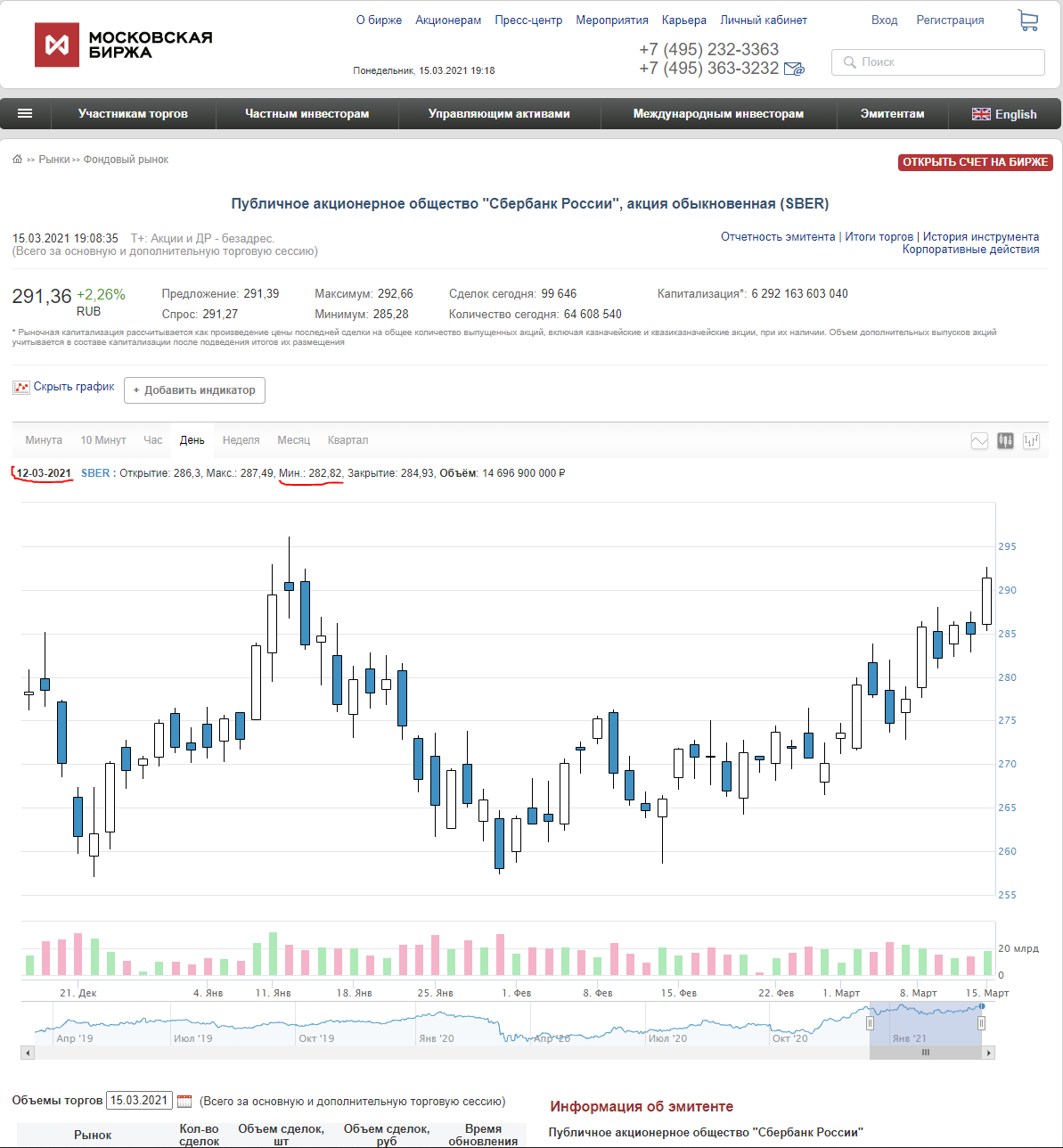


Рисунок 11 – Официальный сайт Московской биржи. Акции Сбербанка

#### **2.6.1 Сценарий №1**

Предусловия: отсутствуют запущенные и доступные экземпляры модуля «Исполнитель».

Последовательность действий:

1. Формирование и отправка запроса пользователем

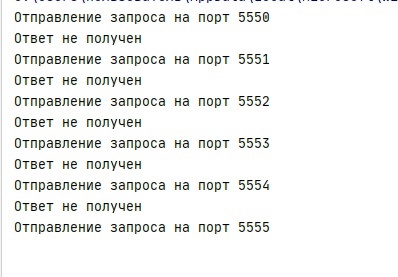


Рисунок 12 – Отправка запроса

1. Ожидание получения ответа

Результат: пользователь ожидает запуск исполнителя и получения от него ответа.

#### **2.6.2 Сценарий №2**

Предусловие: запущен и доступен хотя бы один экземпляр модуля «Исполнитель».

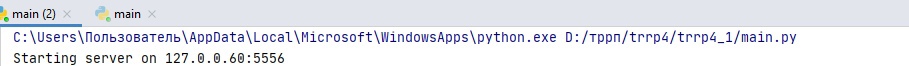


Рисунок 13 – Запуск сервера



Рисунок 14 – Подключение к серверу

Последовательность действий:

1. Формирование и отправка запроса пользователем

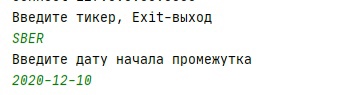


Рисунок 15 – Формирование и отправка запроса

1. Ожидание получения ответа

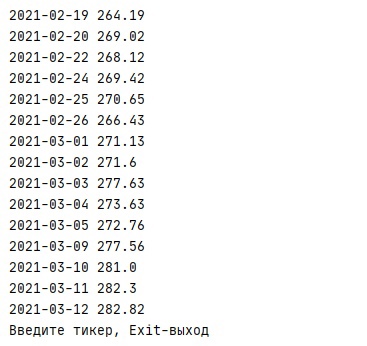


Рисунок 16 – Ответ сервера

Результат: Получен ответ в виде соответствующей структуры.

#### **2.6.3 Сценарий №3**

Предусловия: отсутствуют запущенные и доступные экземпляры модуля «Исполнитель».

Последовательность действий:

1. Модуль «Контроллер» производит запуск модуля «Исполнитель»



Рисунок 17 – Запуск сервера

1. «Исполнитель» запущен и ожидает запрос пользователя



Рисунок 18 – Ожидание запроса сервером

Результат: Исполнитель ожидает запрос пользователя.

#### **2.6.4 Сценарий №4**

Предусловие: запущен и доступен хотя бы один экземпляр модуля «Исполнитель».

Последовательность действий:

* + - 1. Формирование и отправка запроса пользователем

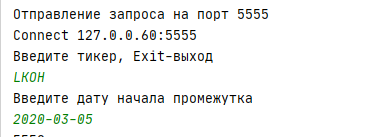


Рисунок 19 – Формирование и отправка запроса

* + - 1. Ожидание получения ответа

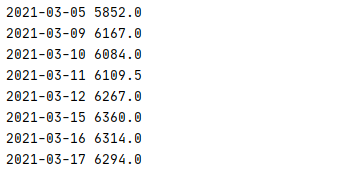


Рисунок 20 – Ответ сервера

* + - 1. Подключение и отправка запроса вторым пользователем

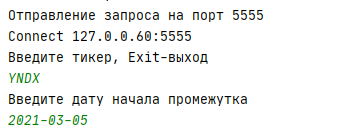


Рисунок 21 – Подключение и отправка запроса вторым пользователем

* + - 1. Ожидание получения ответа

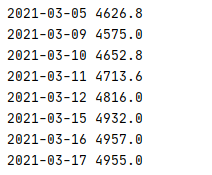


Рисунок 22 – Ответ сервера

Результат: получен ответ в виде соответствующей структуры в обоих случаях.

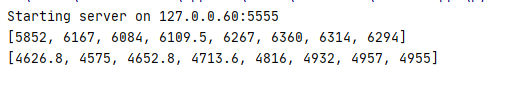


Рисунок 23 – Ответ сервера

#### **2.6.5 Сценарий №5**

Предусловие: запущен и доступен хотя бы один экземпляр модуля «Исполнитель» и к нему подключен хотя бы один пользователь.

Последовательность действий:

Подключение пользователя к модулю «Исполнитель»

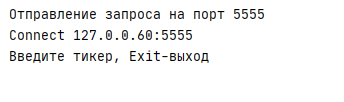


Рисунок 24 – Подключен к модулю

Аварийное завершение модуля «Исполнитель»

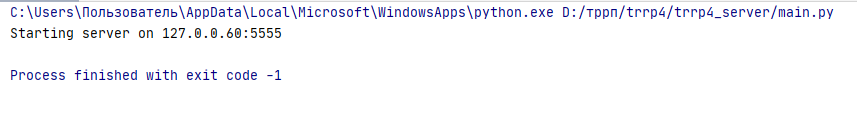


Рисунок 25 – Аварийное завершение

Модуль «Контроллер» производит запуск модуля «Исполнитель» на другом узле

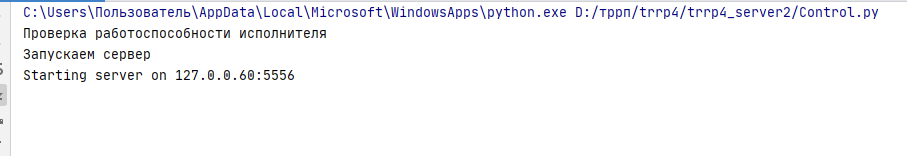


Рисунок 26 – Запуск модуля на другом узле

Пользователь автоматически подключается к доступному Исполнителю

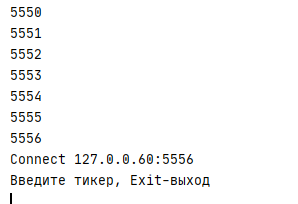


Рисунок 27 – Подключение пользователя

* + - 1. «Исполнитель» запущен и ожидает запрос пользователя



Рисунок 28 – Ожидание запроса сервером

Ожидаемый результат: «Исполнитель» запущен и ожидает запрос пользователя.

.

# 3 Заключение

В ходе лабораторной работы были изучены методы коммуникации процессов в сети, а также средства динамического конфигурирования распределенных приложений.

В результате следующие требования были выполнены полностью:

1. Распределенное приложение спроектировано с учетом особенностей предметной области. Выбрана наиболее подходящая модель распределенной системы. Если используется распределенная база данных, то тиражирование данных имеет подходящий для данной предметной области механизм и архитектуру (3 балла).
2. Архитектура системы является оптимальной для заданных при разработке критериев. В отчете присутствует обоснование выбора данного типа архитектуры» выполнено полностью (3 балла).
3. Приложение обеспечивает параллельную работу нескольких клиентов и серверов, в том числе на одном компьютере. Серверы распределенной системы выполняют различные функции (2 балла).
4. Приложение является масштабируемым, позволяет добавлять новых участников взаимодействия без переписывания кода и перезапуска приложений (2 балла).
5. Система является отказоустойчивой. В случае, если один и/или несколько компонент системы аварийно завершают свою работу (2 балла).
6. Распределенное приложение продолжает работать и в случае, если после аварийного завершения некоторого компонента, он восстановлен на другом узле вычислительной сети (3 балла).
7. Отчет содержит подробное описание архитектуры каждого компонента распределенного приложения (3 балла).
8. В отчете описана структура передаваемых данных, формат сообщений и вид протокола, используемого для этого (2 балла).
9. В отчете представлено описание способа передачи сообщений при коммуникации компонентов распределенной системы с обоснованием (2 балла).

Не было выполнено требование «Существует возможность динамического реконфигурирования системы» не выполнено» (0 из 4 баллов).

Требование «Для организации взаимодействия компонент распределенной системы используется не менее четырех различных средств коммуникации. В отчете присутствует четкое обоснование выбора средств взаимодействия для каждого конкретного случая» выполнено частично (2 из 4 баллов).

Вывод самооценки – получили 24 из 30 баллов, что превышает 40%, поэтому предполагаем, что лабораторную работу сдали успешно.

# Библиографический список

1. Python 3.8.0 [электронный ресурс]. URL: <https://www.python.org/downloads/release/python-380/> (дата обращения: 12.03.2021).
2. Python 3.8.0 / license [электронный ресурс]. URL: <https://docs.python.org/3.8/license.html> (дата обращения: 12.03.2021).
3. Ссылка на скачивание JetBrains PyCharm Community Edition 2019.2.3 [электронный ресурс]. URL: <https://download.jetbrains.com/python/pycharm-community-2019.2.3.exe> (дата обращения: 12.03.2021).
4. Разъяснение системы распространения и правил использования JetBrains PyCharm Community Edition [электронный ресурс]. URL: <https://blog.jetbrains.com/pycharm/2017/09/pycharm-community-edition-and-professional-edition-explained-licenses-and-more/> (дата обращения: 12.03.2021).
5. Ссылка на лицензию Apache 2.0 [электронный ресурс]. URL: <https://github.com/grpc/grpc/blob/master/LICENSE> (дата обращения: 12.03.2021).
6. Московская биржа [электронный ресурс]. URL: <https://www.moex.com/> (дата обращения: 10.03.2021).
7. Documentation / What is gRPC? / FAQ [электронный ресурс]. URL: <https://grpc.io/docs/what-is-grpc/faq/> (дата обращения: 11.03.2021).
8. About the Interface Definition Language Specification Version 4.2 [электронный ресурс]. URL: <https://www.omg.org/spec/IDL/4.2/> (дата обращения: 11.03.2021).