|  |
| --- |
|  |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |
| Институт информационных технологий (ИТ) |
| Кафедра инструментального и прикладного программного обеспечения |

|  |  |
| --- | --- |
| **ОТЧЕТ**  **ПО ПРАКТИЧЕСКОЙ РАБОТЕ №4** | |
| **по дисциплине** |  |
| **«Архитектура клиент-серверных приложений»**  **Тема:**  **«Разработка клиент-серверного приложения с использованием**  **технологии Websocket и Spring Framework»** | |
| Выполнил студент группы: ИКБО-20-21 | Костенко П.А. |
| Принял преподаватель кафедры ИиППО | Волков М.Ю. |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | « » 2023 г. |  |
| «Зачтено» | « » 2023 г. |  |

Москва 2023

**Теоретическое введение**

Websocket позволяет создать канал связи между клиентом и сервером. В частности, канал связи, который использует протокол WebSocket в качестве протокола связи. Протокол WebSocket совместим с протоколом HTTP. который также работает через TCP / IP . Однако он имеет исключительные улучшения, в части меньших накладных расходов, чем HTTP, и двунаправленную веб-связь. Таким образом, они в первую очередь предназначены для веб-приложений, которым требуется постоянное соединение с сервером. Связь WebSocket может использоваться между любыми типами приложений, но чаще всего WebSocket используется для облегчения связи между серверным приложением и приложением клиентом на основе браузера.

Традиционное веб-общение с использованием протокола HTTP работает следующим образом:

* во-первых, клиент (обычно веб-браузер) должен подключиться к серверу;
* затем клиент отправляет запрос ресурса (например, запрос веб-страницы);
* после этого сервер отвечает;
* закрывает канал связи.

С другой стороны, WebSocket работает следующим образом:

* держит соединение открытым;
* кроме того, и клиент, и сервер могут делать запросы и отправлять ответы.

Протокол WebSocket призван заменить существующие обходные механизмы HTTP и предоставить эффективный протокол для одновременной двунаправленной связи с малой задержкой между браузерами и серверами по одному TCP-соединению.  HTTP изначально был разработан для передачи ресурсов типа "запрос-ответ" в распределенных гипермедийных системах, но не для одновременной двунаправленной связи. Для преодоления этих архитектурных ограничений используются несколько HTTP-механизмов (сгруппированных под неофициальным названием Comet ), которые часто бывают сложными и неэффективными.

В этой работе кратко описываются отношения между WebSocket и HTTP / 1.1.

Поскольку HTTP не был разработан для поддержки сообщений, инициируемых сервером, для этого было разработано несколько механизмов, каждый из которых имеет свои преимущества и недостатки.

WebSocket — это протокол, который позволяет одновременную двунаправленную передачу текстовых и двоичных сообщений между клиентами (в основном браузерами) и серверами по одному TCP-соединению. WebSocket может обмениваться данными через TCP на порт 80 (схема «ws») или через TLS / TCP на порт 443 (схема «wss»).

WebSocket разработан для добавления поддержки сокетов TCP с минимальными изменениями связи браузера с сервером, обеспечивая необходимые ограничения безопасности в Интернете. WebSocket добавляет минимальную функциональность поверх TCP, не более чем следующее: модель безопасности на основе происхождения, преобразование IP-адресов, используемых в TCP, в URL-адреса, используемые в Интернете, протокол сообщений поверх протокола байтового потока, заключительное рукопожатие.

Протокол WebSocket разработан как простой протокол и обеспечивает основу для построения на его основе подпротоколов приложений, подобно тому, как протокол TCP позволяет создавать протоколы приложений (HTTP, FTP, SMTP, POP3, Telnet и т. Д.).

Стандарт WebSocket состоит из двух частей: протокола WebSocket, стандартизованного как RFC 6455, и API WebSocket .

Сетевой протокол WebSocket состоит из двух компонентов:

* открывающее рукопожатие для согласования параметров соединения WebSocket
* формирование бинарных сообщений для отправки текстовых и бинарных сообщений

Когда вышел Spring, это была более простая и легкая альтернатива J2EE, облегчающая разработку J2EE. Enterprise Java относится к корпоративному программному обеспечению Java. Это компьютерное программное обеспечение, используемое для удовлетворения потребностей организации, а не отдельных пользователей.

Каковы преимущества использования Spring при создании корпоративных сложных приложений?

Spring Framework использует Java POJO (простой старый объект Java), что значительно упрощает создание приложений корпоративного класса по сравнению с тяжелыми EJB (из более ранних версий J2EE). POJO - это объект Java, на который не накладываются никакие ограничения, кроме тех, которые установлены спецификацией языка Java. Таким образом, POJO не должен расширять заранее заданные классы или реализовывать заранее заданные интерфейсы во фреймворке.

Преимущества использования Spring Framework:

* Способствует ослаблению зацепления за счет использования внедрения зависимости. Таким образом вместо того, чтобы жестко кодировать зависимости объекта, вы просто указываете зависимости через файл конфигурации (или аннотации, или код Java).
* Использует **АОП (аспектно-ориентированное программирование).**Это позволяет отделить сквозные задачи (например, ведение журнала, аудит, декларативные транзакции, безопасность, кеширование и т. Д.) от бизнес-логики.
* **Минимизирует шаблонный код Java**. В Spring есть набор вспомогательных пакетов и классов, чтобы упростить работу и избежать шаблонного кода, и вам нужно беспокоиться только о тех классах, которые вам нужны, и игнорировать остальные.
* Он использует некоторые из **существующих технологий,** таких как фреймворки ORM, фреймворки журналирования, таймеры JEE, Quartz и JDK и другие.
* Основные функции среды Spring можно использовать при разработке любого приложения Java, но есть расширения для создания **веб-приложений** на основе платформы Java EE.

Чуть подробнее рассмотрим основные концепции, которые лежат в основе Spring Framework.

**Постановка задачи**

Используя информацию из данной практической работы, необходимо реализовать клиент-серверное приложение с использованием Websocket. Суть приложения заключается в следующем. При обращении клиентской части по адресу /webs необходимо выполнять обработку Websocket. В случае получения в вебсокете данных, необходимо ответить их же содержимым.

**Программный код**

На рисунках 1–2 представлена реализация клиентской части приложения.

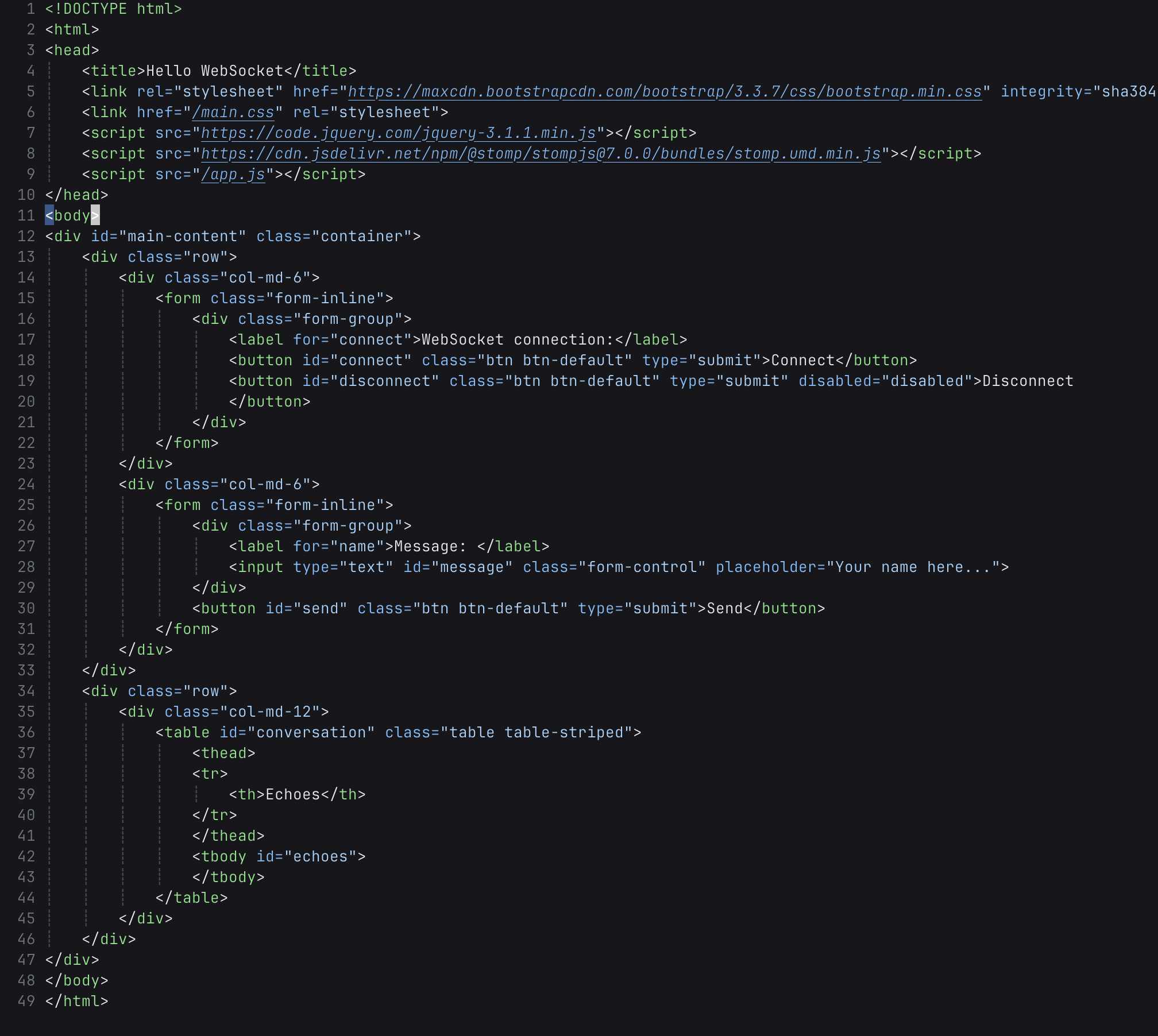


Рисунок 1 – index.html



Рисунок 2 – файл app.js

На рисунках 3–7 представлена реализация серверной части.



Рисунок 3 – MessagingStompWebsocketApplication,java

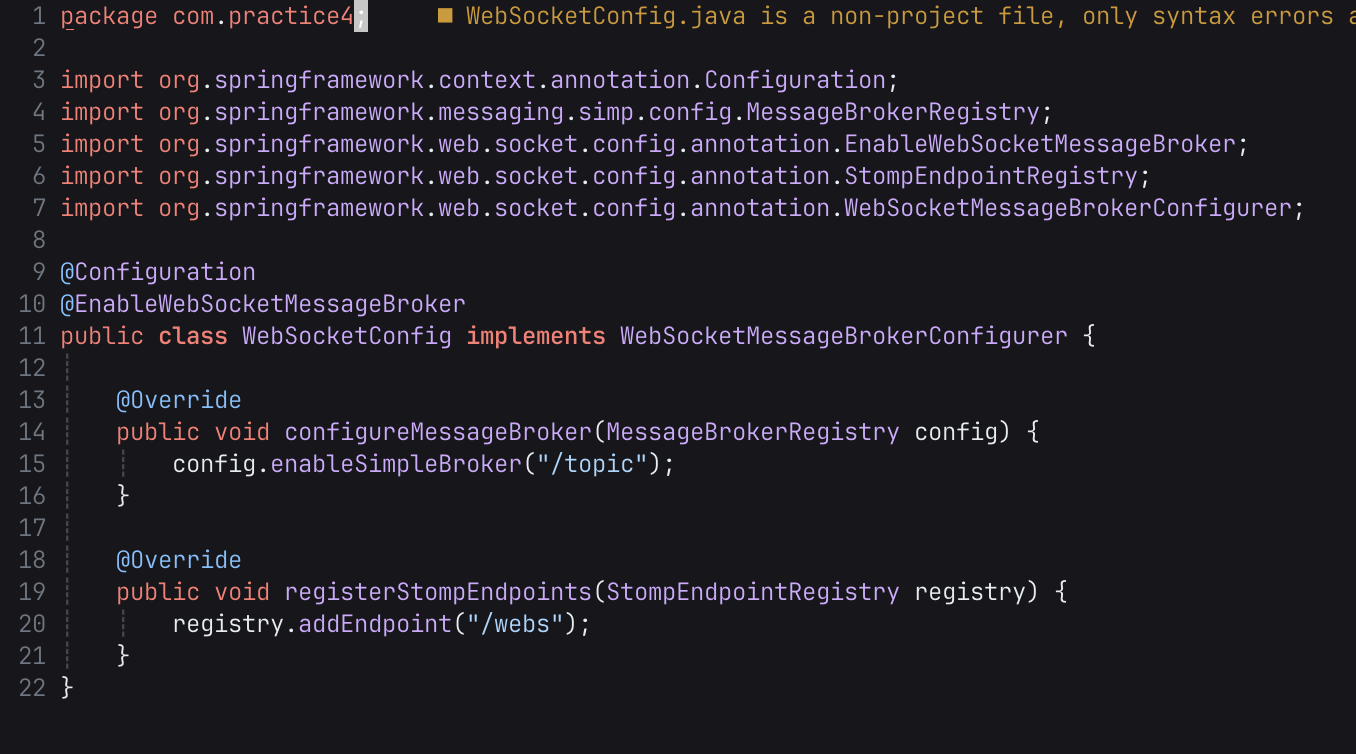


Рисунок 4 – Конфигурация программы

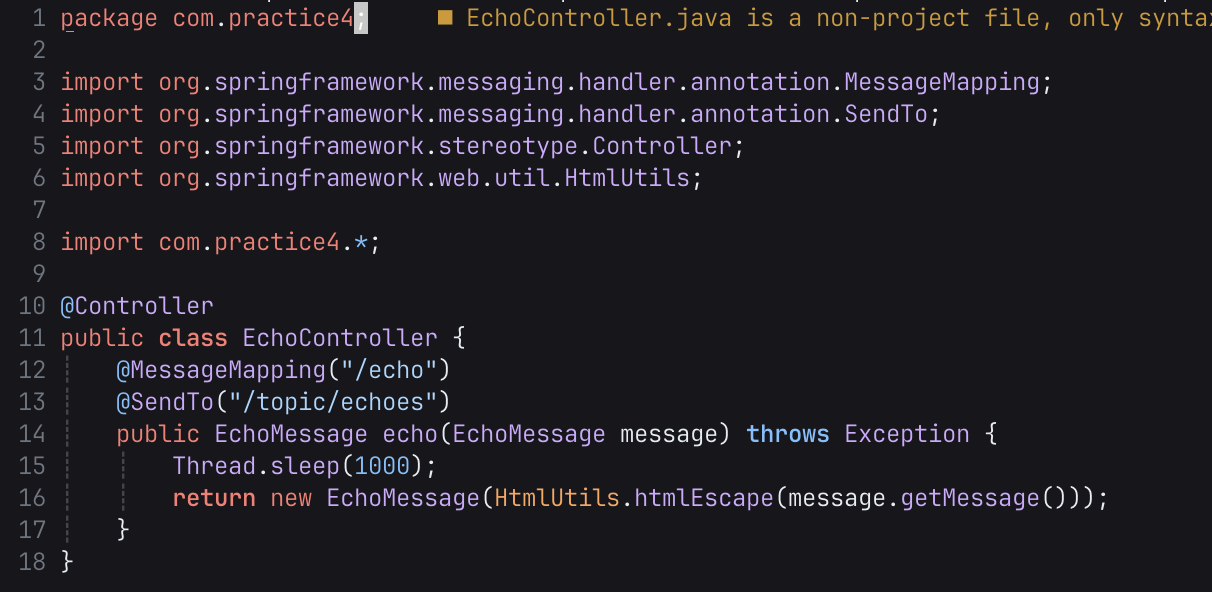


Рисунок 5 – Контроллер, обрабатывающий сообщения

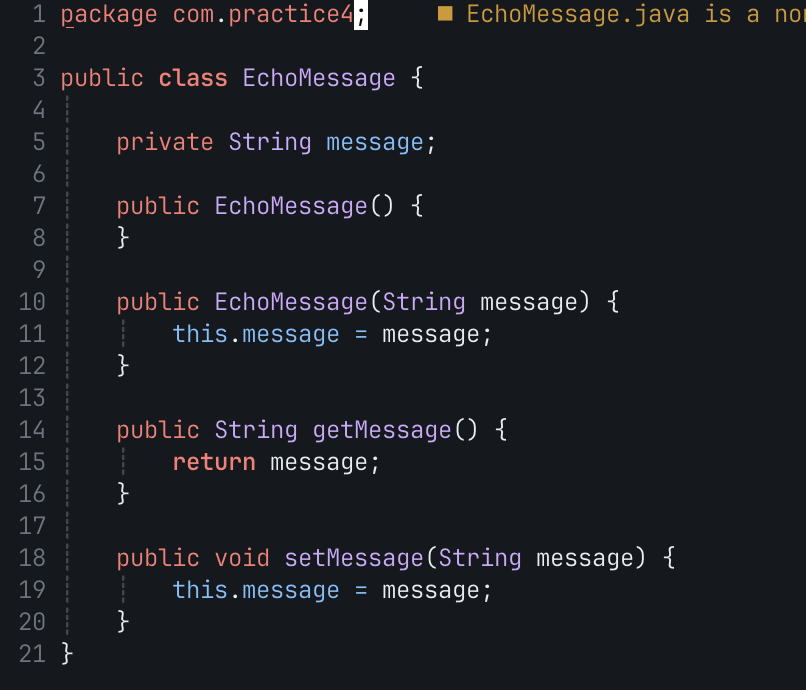


Рисунок 6 – Класс Message

**Вывод программы**

На рисунке 7 представлена работа программы.

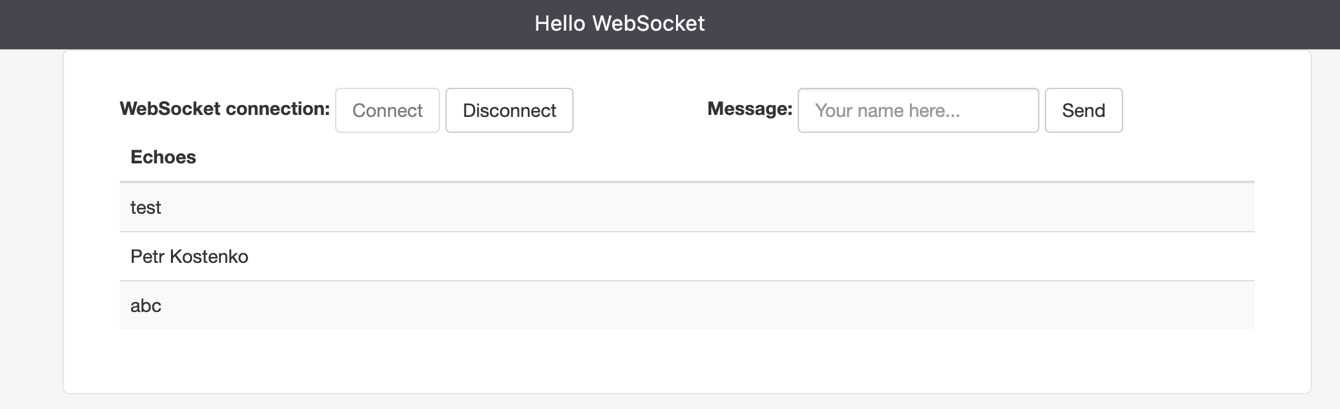


Рисунок 7 – Работа программы

**Вывод**

В результате выполнения были получены навыки работы с WebSocket. И реализовано клиент-серверное приложение с использованием WebSocket.

**Список литературы**

1. Документация Java 17 от Oracle [Электронный ресурс]: URL – <https://docs.oracle.com/en/java/javase/17/docs/api/index.html> [Дата обращения 7.10.2023];
2. Руководство по языку программирования Java [Электронный ресурс]: URL – <https://metanit.com/java/tutorial/> [Дата обращения 7.10.2023];
3. A Guide to the Java API for WebSocket [Электронный ресурс]: URL - <https://www.baeldung.com/java-websockets> [Дата обращения 7.10.2023];
4. WebSocket [Электронный ресурс]: URL – https://developer.mozilla.org/ru/docs/Web/API/WebSocket [Дата обращения 7.10.2023];