Задание «Клиент-серверное приложение на основе WebSocket»

Загружать решения в виде Zip архива в проект Smart LMS под названием HW4:

- Название ZIP архива должно совпадать с Вашей фамилией.
- В архиве должны быть:
  - папка с программным проектом
  - файл «фамилия.txt» с оценкой, на которую выполнялось задание
- Пожалуйста, создавайте Maven/Gradle проект.
- В одном проекте иметь и код клиента (JS) и код сервера (Java).
- Язык для выполнения работы – Java; нужно использовать Netty, Oracle JDK 21.
- Код на другом языке либо без Netty не принимается.

Мотивация.
Одна из наиболее частых задач программной инженерии при использовании компьютерных сетей – создание

(1) собственного протокола (бизнес-протокола, custom protocol) вместе с
(2) собственным программным клиентом и сервером (работающими по новому протоколу с поддержкой множества клиентов)

В этом задании Вам следует познакомиться с:

(1) best-practices создания сетевых протоколов на Application Layer (OSI)
(2) межязыковым сетевым взаимодействием на примере JavaScript ←→ Java
(3) Netty Framework. При разработке клиент-серверного приложения Вы задействуете Netty Framework себе в помощь и познакомитесь с лучшими архитектурными patterns разработки сетевых программных серверов.

Почему Java?
Netty Framework – лучший в своем классе. Конечно, есть библиотеки для C++, Go, Kotlin либо других языков, но они не настолько активно развиваются, менее гибкие, имеют не такую обширную функциональность, либо чрезмерно рутинные.

Netty Framework – один из наиболее популярных Framework для разработки сетевых приложений. Его используют многие ведущие компании, включая RedHat, Twitter, Infinispan, HornetQ, Vert.x, Finagle, Akka, Yandex, Apache Cassandra, Elasticsearch, Facebook. Проект активно развивается уже долгие годы, почти каждый день есть обновления (commits): https://github.com/netty/netty

Зависимость для Maven проекта:

```
<!-- https://mvnrepository.com/artifact/io.netty/netty-all -->
<dependency>
  <groupId>io.netty</groupId>
  <artifactId>netty-all</artifactId>
  <version>4.1.109.Final</version>
</dependency>
```

Оценивание задания. Во-первых, код будет анализироваться на точность выполнения требований этого задания. Затем будет учитываться корректность и качество кода (форматирование, обработка исключений, иные стандартные элементы качества кода). Дополнительно может быть проведена устная защита выборки домашних работ.

НЕОБХОДИМО ВЫБРАТЬ ОДНО ИЗ ПРЕДЛОЖЕННЫХ НИЖЕ ЗАДАНИЙ. ЛУЧШАЯ СТРАТЕГИЯ: ВЫПОЛНИТЬ 1ОЕ, ЗАТЕМ 2ОЕ ЗАДАНИЕ.

Задание на оценку до 8 баллов.

Реализовать клиент-серверное приложение «Мой текстовый чат (MyTextChat)».

Общие требования:

- Клиент – чистый JavaScript (без frameworks)
- Сервер – Netty Java Server
- Протокол общения – WebSocket (Text Mode)
- Пользователь чата может вводить сообщения и отправлять их серверу. Сервер дублирует сообщение всем остальным пользователям чата.

Требования к клиенту:

- Единая HTML страница вместе с кодом JavaScript
- В одном браузере можно открыть несколько HTML страниц (это будут разные клиенты чата)
- HTML страница содержит 3 элемента:
  - Тестовая область (1 строка) для ввода сообщений (до 25 символов)
  - Тестовая область (10 строк, readonly, по желанию можно добавить прокрутку) для просмотра сообщений от всех пользователей
  - Кнопка «отправить», которая отправляет всем пользователям чата текстовое сообщение из строки ввода (сообщение отображается у всех пользователей включая отправителя в текстовой области для просмотра сообщений; добавляется к уже имеющимся сообщениям)
- Сразу при открытии HTML страницы клиент пытается установить соединение с сервером по протоколу WebSocket
- При успешном подключении выводится история сообщений, полученная от сервера
- Если соединение установить не удалось, то
  - через 10 секунд повторить попытку установки соединения (использовать таймер)
  - строка ввода и кнопка «отправить» - неактивны и выведено на странице соответствующее сообщение и оставшееся число секунд (обновляемое поле) до повтора попытки соединения

Требования к серверу:

- Использовать Netty Framework, протокол WebSocket
- Поддерживать соединения от нескольких пользователей (до 100)
- Реализовать логику приема и дублирования сообщений от пользователей
- Вести историю сообщений (до 50 строк)
- При подключении нового пользователя отсылать ему историю сообщений

Задание на оценку до 10 баллов.

Реализовать клиент-серверное приложение «Мой текстовый чат (MyTextChat)» (см. выше), с дополнительными требованиями.

Дополнительные общие требования:

- В отличие от предыдущего задания, необходимо разработать свой собственный бизнес-протокол (custom, business protocol).
- Теперь в этом задании клиент-серверное взаимодействие будет осуществляться не просто рассылкой текста, а с помощью JSON. Для простоты по-прежнему можно использовать WebSocket TEXT MODE (хотя можете использовать BINARY MODE).
- Каждое JSON сообщение серверу – запрос (должен содержаться код команды), каждое JSON сообщение от сервера – ответ. Формат JSON сообщений разработайте самостоятельно – это часть задания, пример ниже.

Дополнительные требования к клиенту и серверу:

- Появляется дополнительные: поле «имя» («name») и кнопка «присоединиться»
- Немного изменяется сценарий подключения пользователя к чату: не сразу при открытии страницы, а после ввода имени и нажатии кнопки «присоединиться»
- Алгоритм подключения к серверу аналогичен предыдущему заданию
- После успешного подключения клиент отправляет серверу запрос (JSON строку/объект) – «присоединиться к чату» с указанием своего имени.
- Если в чате нет пользователя с таким именем, сервер отсылает пользователю ответ – история сообщений (JSON строка/объект; ответ). Иначе, приходит JSON ответ – «ошибка» с указанием причины ошибки.
- Далее обычные сообщения посылаются клиентами в виде JSON запросов и приходят другим клиентам в формате JSON сообщений от сервера.
- Пользователь может написать @<name> <текст сообщения>. Тогда формируется соответствующий JSON запрос серверу, чтобы введенный <текст сообщения> отправлялся только пользователю с именем <name>.
- Сервер возвращает ошибку (JSON ответ с соответствующим кодом), если в чате сейчас нет пользователя с именем <name>, которому адресовано сообщение (клиент должен отобразить ошибку соответствующим образом).

Пример JSON:

```
{
  "command" : 1,
  "message": "сообщение от сервера, если ошибка",
  "text": "<текстовое сообщение от пользователя>",
  "source": "<myname>"
  "target": "<name>"
}
```

Внимание: не обязательно все поля передавать по сети каждый раз (например, если сообщение передается всем пользователям чата, то `"target"` может отсутствовать; аналогично – поле `"source"` может присутствовать только при первичном подключении клиента к серверу). Состав полей JSON должен определяться значением поля `"command"`

Примеры JavaScript Client & Netty Java Server (работают на WebSocket):

https://github.com/Wikience/WRRS-JS
https://github.com/Wikience/RServer

Работа с JSON, WebSocket – лекция и учебник:
https://learn.javascript.ru/websockets
https://learn.javascript.ru/json

Почему WebSocket?

It enables fast, asynchronous, real-time Web I/O
Its support is ubiquitous

https://caniuse.com/#feat=websockets