Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3

по «Низкоуровневое программирование»

Вариант – 6(JSON)

Выполнил:

Студент группы P33102

Голощапов И.А.

Преподаватели:

Кореньков Юрий Дмитриевич

Санкт-Петербург

2023

Задание 3

На базе данного транспортного формата описать схему протокола обмена информацией и воспользоваться

существующей библиотекой по выбору для реализации модуля, обеспечивающего его функционирование.

Протокол должен включать представление информации о командах создания, выборки, модификации и

удаления данных в соответствии с данной формой, и результатах их выполнения.

Используя созданные в результате выполнения заданий модули, разработать в виде консольного приложения

две программы: клиентскую и серверную части. Серверная часть – получающая по сети запросы и операции

описанного формата и последовательно выполняющая их над файлом данных с помощью модуля из первого

задания. Имя фала данных для работы получать с аргументами командной строки, создавать новый в случае

его отсутствия. Клиентская часть – в цикле получающая на стандартный ввод текст команд, извлекающая из

него информацию о запрашиваемой операции с помощью модуля из второго задания и пересылающая её на

сервер с помощью модуля для обмена информацией, получающая ответ и выводящая его в человекопонятном виде в стандартный вывод.

Порядок выполнения:

1. Изучить выбранную библиотеку

a. Библиотека должна обеспечивать сериализацию и десериализацию с валидацией в

соответствии со схемой

b. Предпочтителен выбор библиотек, поддерживающих кодогенерацию на основе схемы

c. Библиотека может поддерживать передачу данных посредством TCP соединения

 Иначе, использовать сетевые сокеты посредством API ОС

d. Библиотека может обеспечивать диспетчеризацию удалённых вызовов

 Иначе, реализовать диспетчеризацию вызовов на основе информации о виде команды

2. На основе существующей библиотеки реализовать модуль, обеспечивающий взаимодействие

a. Описать схему протокола в поддерживаемом библиотекой формате

 Описание должно включать информацию о командах, их аргументах и результатах

 Схема может включать дополнительные сущности (например, для итератора)

b. Подключить библиотеку к проекту и сформировать публичный интерфейс модуля с

использованием встроенных или сгенерированных структур данных используемой библиотеки

 Поддержать установление соединения, отправку команд и получение их результатов

 Поддержать приём входящих соединений, приём команд и отправку их результатов

c. Реализовать публичный интерфейс посредством библиотеки в соответствии с п1

3. Реализовать серверную часть в виде консольного приложения

a. В качестве аргументов командной строки приложение принимает:

 Адрес локальной конечной точки для прослушивания входящих соединений

 Имя файла данных, который необходимо открыть, если он существует, иначе создать

b. Работает с файлом данных посредством модуля из задания 1

c. Принимает входящие соединения и взаимодействует с клиентами посредством модуля из п2

d. Поступающая информация о запрашиваемых операциях преобразуется из структур данных

модуля взаимодействия к структурам данных модуля управления данными и наоборот

4. Реализовать клиентскую часть в виде консольного приложения

a. В качестве аргументов командной строки приложение принимает адрес конечной точки для

подключения

b. Подключается к серверу и взаимодействует с ним посредством модуля из п2

c. Читает со стандартного ввода текст команд и анализирует их посредством модуля из задания 2

d. Преобразует результат разбора команды к структурам данных модуля из п2, передаёт их для

обработки на сервер, возвращаемые результаты выводит в стандартный поток вывода

Исходный код: https://github.com/Ja-Vani/LLP3

Пример сеанса:

Клиент:

```
V().has("name", eq("Ivan")).has("name", eq("Ivan"));
READ 0: { "name": "person", "id": 0, "attributes": [ { "name": "name", "value": "Ivan" } ], "edges": [ { "name": "know", "id": 1 } ] }
READ 1: { "name": "person", "id": 1, "attributes": [ { "name": "name", "value": "Ivan" } ], "edges": [ ] }
```

Сервер:

```
listening
Reading from client
READ: { "oper": 5, "statements": [ ] }
listening
Reading from client
READ: { "oper": 5, "statements": [ { "type": 0, "conditions": [ { "type": 3, "name": "name", "option": 0, "value": "name" } ] } ] }
listening
Reading from client
READ: { "oper": 5, "statements": [ ] }
listening
```

Описание Запросов и Ответов.

Запрос {Request: {“name”: “”, “type”” 0 “att”: {}}}

Ответ {Response: {“name”: “”, “att”: [{“name”: “”, “value”: “”}], “edges”:[{“name”:””, “id”:0}]}}

Вывод: В ходе выполнения работы был написан клиент и сервер, основанные на предыдущих работах, которые умеют общаться между собой по средствам json. Клиентская часть принимает запросы со входа в виде Gremlyn подобного синтаксиса. Серверная часть обрабатывает запросы и управляет базой основанной на графе узлов.