Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет программной инженерии и компьютерной техники

**Дисциплина:** Low-level Programming

Отчёт по лабораторной работе №3

Вариант 2 (JSON)

**Выполнил:** Нгуен Нгок Дык

**Студент группы**: P33302

**Преподаватель:** Кореньков Юрий Дмитриевич

Санкт-Петербург

2023 г

## 1. Objective:

On the basis of this transport format, describe the scheme of the information exchange protocol and use the existing library of choice to implement the module that ensures its operation. The protocol should include the presentation of information about the commands to create, retrieve, modify and delete data in accordance with this form, and the results of their execution.

Using the modules created as a result of completing tasks, develop two programs as a console application: client and server parts. The server part receives requests and operations of the described format over the network and sequentially performs them on the data file using the module from the first task. Get the name of the data file to work with command line arguments, create a new one if it does not exist. The client part - in a loop, receiving the command text on standard input, extracting information about the requested operation from it using the module from the second task. Next, it is sent to the server using the module for information exchange, and the response is received, parsed for output in a format convenient for reading to standard output.

## 2. Requirement

1. Explore the selected library
    a. The library must provide serialization and deserialization with validation according to the schema
    b. It is preferable to choose libraries that support schema-based code generation
    c. The library can support data transfer over a TCP connection
        - Else, use network sockets via POSIX API
    d. The library can provide remote call dispatching
        - Else, implement call dispatch based on command type information
2. Based on the existing library, implement a module that provides interaction
    a. Describe the protocol scheme in a format supported by the library
        - Description should include information about commands, their arguments and results
        - The schema can include additional entities (for example, for an iterator)
    b. Connect the library to the project and form the public interface of the module using the built-in or generated data structures of the used library:
        - Support establishing a connection, sending commands and receiving their results
        - Support receiving incoming connections, receiving commands and sending their results
    c. Implement the public interface through the library in accordance with p1
3. Implement a module that uses a parser to parse the query language
    a. The application accepts as command line arguments:
        - Local endpoint address to listen for incoming connections
        - Name of the data file to be opened, if it exists, otherwise create
    b. Works with the data file through the module from task 1
    c. Accepts incoming connections and interacts with clients through the module from p2
    d. The incoming information about the requested operations is converted from the data structures of the interaction module to the data structures of the data management module and vice versa
4. Implement the client side as a console application
    a. As command line arguments, the application accepts the address of the endpoint to connect to
    b. Connects to the server and interacts with it through the module from p2
    c. Reads the text of commands from standard input and parses them using the module from task 2
    d. Converts the result of parsing the command to the data structures of the module from n2, transfers them to the server for processing, outputs the returned results to the standard output stream

5. Present the test results in the form of a report, which includes:
    a. In part 3, give an example of a session of the developed programs
    b. In part 4, describe the solution implemented in accordance with paragraphs 2-4
    c. In part 5, include the drawn up scheme of clause 2a

## 3. Project Structure

1. Graph database implemented as part of laboratory work No. 1. It is assembled into a static library and provides an interface for working with the database. Connects to the server.

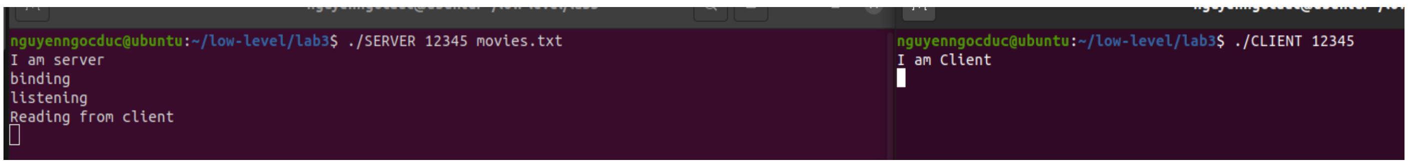
2. The Gremlin query language parser implemented as part of lab #2. It is assembled into a static library and provides an interface for processing input data. Connects to the client.

3. Parser using the json-c library. It is assembled into a static library and provides an interface for client-server interaction. Connects to client and server.

## 4. Demonstration of the functionality of the implemented module.

### Client-Server connection

1. ./SERVER <port> <filename>
2. ./CLIENT <port>

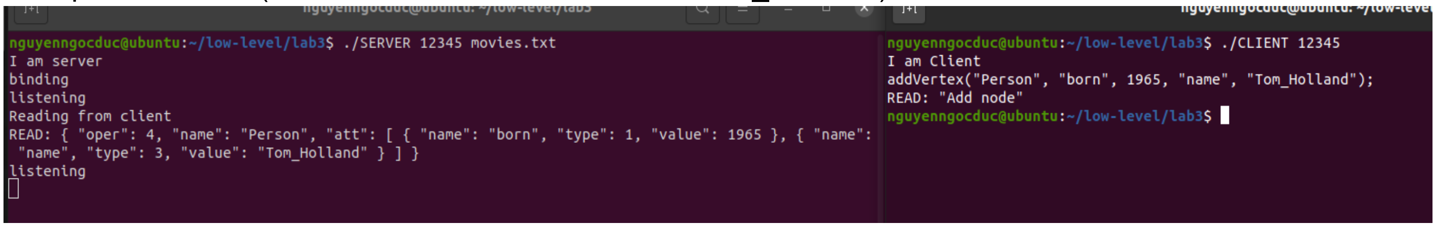


```
nguyenngocduc@ubuntu:~/low-level/lab3$ ./SERVER 12345 movies.txt
I am server
binding
listening
Reading from client
```

```
nguyenngocduc@ubuntu:~/low-level/lab3$ ./CLIENT 12345
I am Client
```

### Add Vertex

example: addVertex("Person", "born", 1965, "name", "Tom_Holland");

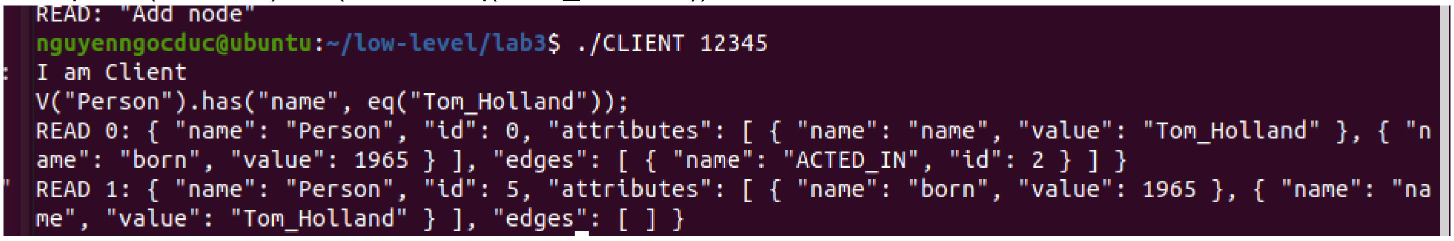


```
nguyenngocduc@ubuntu:~/low-level/lab3$ ./SERVER 12345 movies.txt
I am server
binding
listening
Reading from client
READ: { "oper": 4, "name": "Person", "att": [ { "name": "born", "type": 1, "value": 1965 }, { "name":
 "name", "type": 3, "value": "Tom_Holland" } ] }
listening
```

```
nguyenngocduc@ubuntu:~/low-level/lab3$ ./CLIENT 12345
I am Client
addVertex("Person", "born", 1965, "name", "Tom_Holland");
READ: "Add node"
nguyenngocduc@ubuntu:~/low-level/lab3$
```

### Select

example: V("Person").has("name", eq("Tom_Holland"));

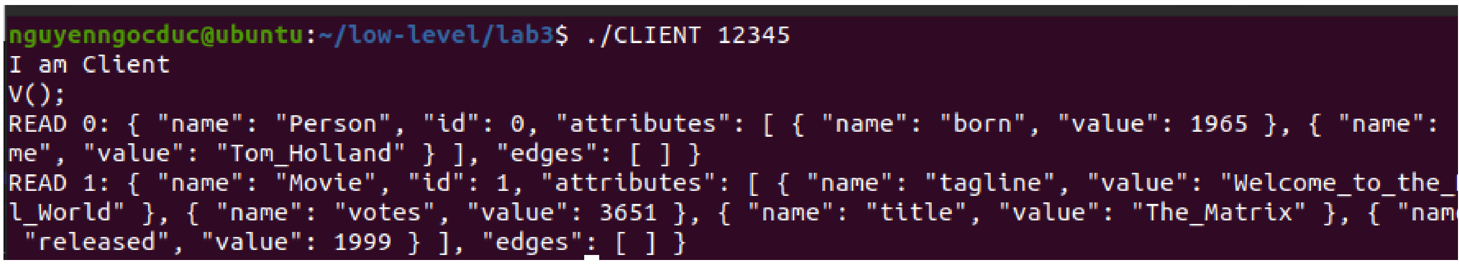


```
READ: "Add node"
nguyenngocduc@ubuntu:~/low-level/lab3$ ./CLIENT 12345
I am Client
V("Person").has("name", eq("Tom_Holland"));
READ 0: { "name": "Person", "id": 0, "attributes": [ { "name": "name", "value": "Tom_Holland" }, { "n
ame": "born", "value": 1965 } ], "edges": [ { "name": "ACTED_IN", "id": 2 } ] }
READ 1: { "name": "Person", "id": 5, "attributes": [ { "name": "born", "value": 1965 }, { "name": "na
me", "value": "Tom_Holland" } ], "edges": [ ] }
```

### Add Edge

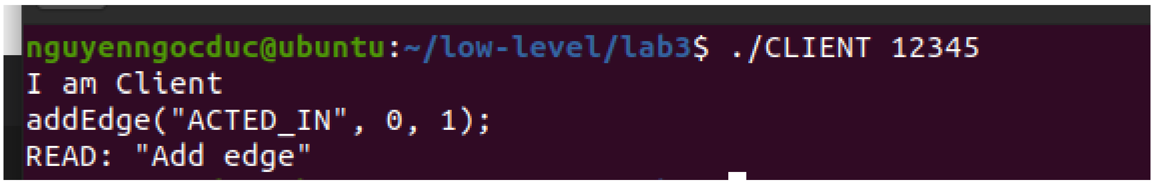
We will add a Person and a Movie.

addVertex("Person", "born", 1965, "name", "Tom_Holland");
addVertex("Movie", "tagline", "Welcome_to_the_Real_World", "votes", 3651, "title", "The_Matrix", "released", 1999);



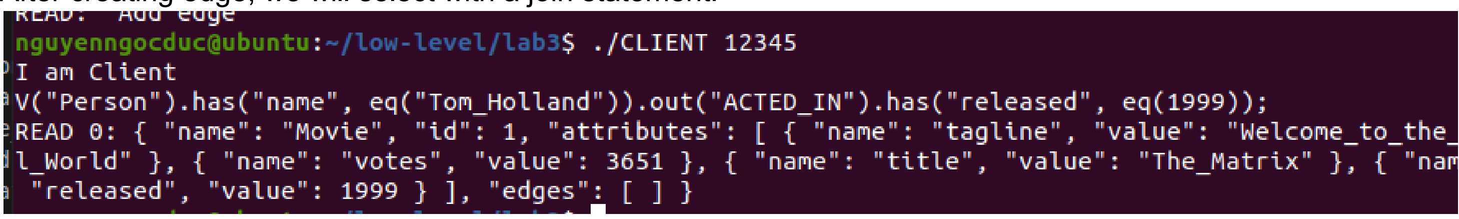
```
nguyenngocduc@ubuntu:~/low-level/lab3$ ./CLIENT 12345
I am Client
V();
READ 0: { "name": "Person", "id": 0, "attributes": [ { "name": "born", "value": 1965 }, { "name":
me", "value": "Tom_Holland" } ], "edges": [ ] }
READ 1: { "name": "Movie", "id": 1, "attributes": [ { "name": "tagline", "value": "Welcome_to_the_
l_World" }, { "name": "votes", "value": 3651 }, { "name": "title", "value": "The_Matrix" }, { "nam
 "released", "value": 1999 } ], "edges": [ ] }
```

Then will create edge between two these nodes.


```
nguyenngocduc@ubuntu:~/low-level/lab3$ ./CLIENT 12345
I am Client
addEdge("ACTED_IN", 0, 1);
READ: "Add edge"
```

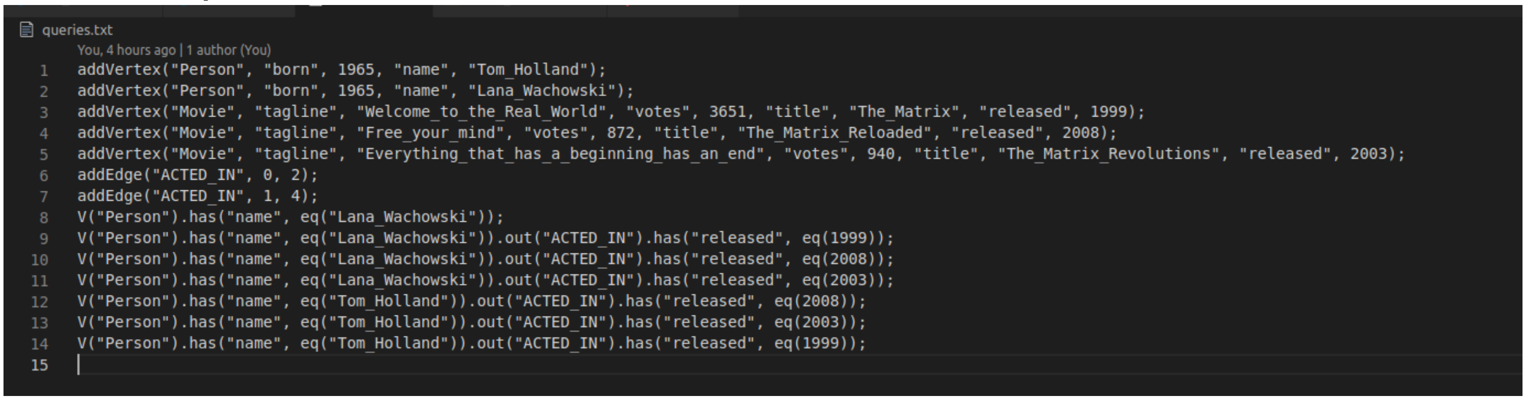
**Syntax**: addEdge(<edge_name>, <elementId_x>, <elementId_y>);

After creating edge, we will select with a join statement.


```
READ:  Add edge
nguyenngocduc@ubuntu:~/low-level/lab3$ ./CLIENT 12345
I am Client
V("Person").has("name", eq("Tom_Holland")).out("ACTED_IN").has("released", eq(1999));
READ 0: { "name": "Movie", "id": 1, "attributes": [ { "name": "tagline", "value": "Welcome_to_the_
l_World" }, { "name": "votes", "value": 3651 }, { "name": "title", "value": "The_Matrix" }, { "nam
 "released", "value": 1999 } ], "edges": [ ] }
```

**Execute queries from a file**

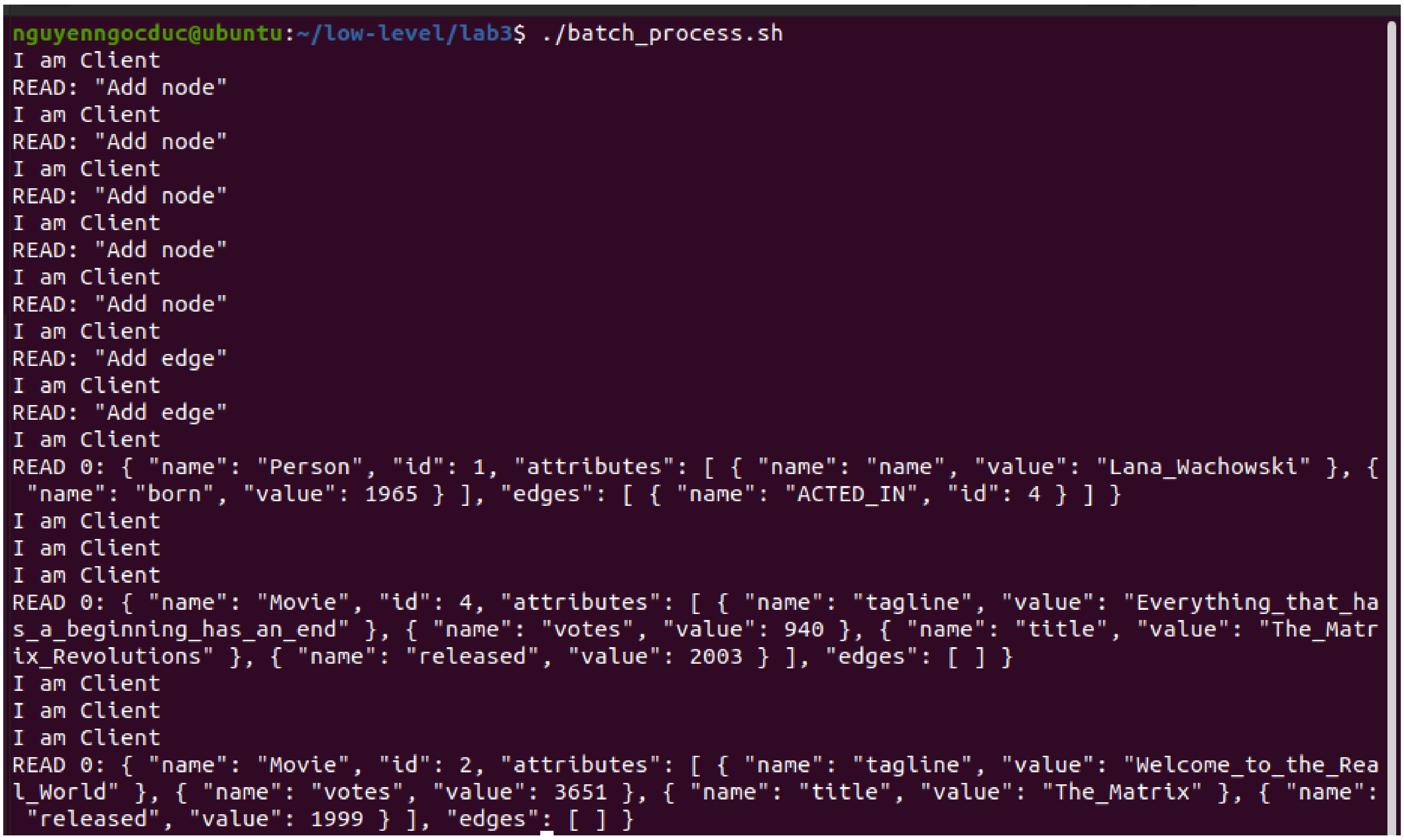
**queries.txt**


```
queries.txt
You, 4 hours ago | 1 author (You)
addVertex("Person", "born", 1965, "name", "Tom_Holland");
addVertex("Person", "born", 1965, "name", "Lana_Wachowski");
addVertex("Movie", "tagline", "Welcome_to_the_Real_World", "votes", 3651, "title", "The_Matrix", "released", 1999);
addVertex("Movie", "tagline", "Free_your_mind", "votes", 872, "title", "The_Matrix_Reloaded", "released", 2008);
addVertex("Movie", "tagline", "Everything_that_has_a_beginning_has_an_end", "votes", 940, "title", "The_Matrix_Revolutions", "released", 2003);
addEdge("ACTED_IN", 0, 2);
addEdge("ACTED_IN", 1, 4);
V("Person").has("name", eq("Lana_Wachowski"));
V("Person").has("name", eq("Lana_Wachowski")).out("ACTED_IN").has("released", eq(1999));
V("Person").has("name", eq("Lana_Wachowski")).out("ACTED_IN").has("released", eq(2008));
V("Person").has("name", eq("Lana_Wachowski")).out("ACTED_IN").has("released", eq(2003));
V("Person").has("name", eq("Tom_Holland")).out("ACTED_IN").has("released", eq(2008));
V("Person").has("name", eq("Tom_Holland")).out("ACTED_IN").has("released", eq(2003));
V("Person").has("name", eq("Tom_Holland")).out("ACTED_IN").has("released", eq(1999));

```

Using bash script to execute queries

**batch_process.sh**

```
#!/bin/bash

cat queries.txt |
while read in; do
   echo "$in" | ./CLIENT 12345
done
```



```
nguyenngocduc@ubuntu:~/low-level/lab3$ ./batch_process.sh
I am Client
READ: "Add node"
I am Client
READ: "Add node"
I am Client
READ: "Add node"
I am Client
READ: "Add node"
I am Client
READ: "Add node"
I am Client
READ: "Add edge"
I am Client
READ: "Add edge"
I am Client
READ 0: { "name": "Person", "id": 1, "attributes": [ { "name": "name", "value": "Lana_Wachowski" }, {
 "name": "born", "value": 1965 } ], "edges": [ { "name": "ACTED_IN", "id": 4 } ] }
I am Client
I am Client
I am Client
READ 0: { "name": "Movie", "id": 4, "attributes": [ { "name": "tagline", "value": "Everything_that_ha
s_a_beginning_has_an_end" }, { "name": "votes", "value": 940 }, { "name": "title", "value": "The_Matr
ix_Revolutions" }, { "name": "released", "value": 2003 } ], "edges": [ ] }
I am Client
I am Client
I am Client
READ 0: { "name": "Movie", "id": 2, "attributes": [ { "name": "tagline", "value": "Welcome_to_the_Rea
l_World" }, { "name": "votes", "value": 3651 }, { "name": "title", "value": "The_Matrix" }, { "name":
 "released", "value": 1999 } ], "edges": [ ] }
```

## 5. Conclusion

In the course of the work, a client and a server were written based on previous works that can communicate with each other using json.
The client side accepts requests from the input in the form of a Gremlyn similar syntax.
The backend processes requests and manages a database based on a graph of nodes.