Protocol Buffer

Strukturált adatok szerializációjához használható megoldás a Google fejlesztésében. Ennél az adatintegrációs módszernél is megjelenik az interfész leírás.

A protokol buffer a szerializáció miatt bináris. Viszont nagy előnye, hogy sok technológiát támogat, ez által növelve a plattformfüggetlenséget.

További részletek itt olvashatók:

https://developers.google.com/protocol-buffers/docs/tutorials

Python mintafeladat

1.) Telepítsük fel a hivatalos oldalról a fordítót. https://github.com/protocolbuffers/protobuf/releases - windows esetén keressük meg a protoc-XXXXXX-win64.zip állományt és csomagoljuk ki.

2.) Hozzunk létre egy könyvtárat ./proto néven és a book.proto állományt a következő tartalommal:

```
syntax = "proto3";

message Book {
    int32 id = 1;
    string title = 2;
    string author = 3;
    float price = 4;
}

message Books {
    repeated Book books = 1;
}
```

Két üzenetet hoztunk létre, Book és Books néven. A Books több Book-ot tartalmazhat. A sorok végén az = 1, = 2 a struktúra mező belső pozícióját adja meg, egytől indul a számozás.

3.) Futtassuk le a következő parancsot:

```
.\protoc\bin\protoc.exe --python_out=.\ book.proto
```

A futtatás után létrejön a book_pb2.py ami generált forráskód, és az adat interfészt tartalmazza. Ennek segítségével lehet kezelni (szerializálni és de-szerializálni) az adatokat.

4.) Futtassuk le a **pip install –upgrade protobuf** parancsot.

5.) Hozzuk létre a server.py fájlt a következő tartalommal:

```
import socket
import book_pb2
import create_books as c

# protoc/bin/protoc --python_out=./ book.proto
# pip3 install --upgrade protobuf

books = c.create_books()

book_store = book_pb2.Books()
for book in books:
    book_store.books.append(book)

bytes_to_send = book_store.SerializeToString()

#TCP socket server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((socket.gethostname(), 4100))
s.listen(10)

while True:
    client_socket, address = s.accept()
    print(f"server> Connection from {address} has been established!\n")

    client_socket.send(bytes_to_send)
    print(f"server> Message sent: {bytes_to_send}\n")

    msg = client_socket.recv(1024)
    print(f"client> {msg}\n")
    client_socket.close()

    if msg == b'bye':
        break

s.close()
```

6.) Hozzuk létre a create_books.py állományt az alábbi tartalommal:

```
import book_pb2

def create_books():
    books = []

    books.append(book_pb2.Book())
    books[0].id = 1
    books[0].title = "Solaris"
    books[0].author = "Stanislaw Lem"
```

```
    books[0].price = 7.54

    books.append(book_pb2.Book())
    books[1].id = 2
    books[1].title = "Dune"
    books[1].author = "Frank Herbert"
    books[1].price = 9.87

    books.append(book_pb2.Book())
    books[2].id = 3
    books[2].title = "Foundation"
    books[2].author = "Isaac Asimov"
    books[2].price = 5.07

    return books
```

7.) Hozzuk létre a client.py állományt az alábbi tartalommal:

```
import socket
import book_pb2
from google.protobuf.json_format import MessageToJson
import json

#TCP socket client
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((socket.gethostname(), 4100))

msg = s.recv(1024)
print(f"server> {msg}\n")

s.sendall(b'bye')
print(f"client> Message sent: {b'bye'}\n")

s.close()

books = book_pb2.Books()
books.ParseFromString(msg)

json_obj = MessageToJson(books)
print(f"client> The server's message in JSON:\n{json_obj}")

dict_obj = json.loads(json_obj)

with open('data.json', 'w', encoding='utf-8') as f:
    json.dump(dict_obj, f, ensure_ascii=False, indent=4)
    print("client> data.json saved\n")

with open('data.bytes', 'wb') as fb:
    fb.write(msg)
```

```
    print("client> data.bytes saved\n")
```

8.) Futtassuk le a szervert és klienst. **python server.py** majd a **python client.py** parancsokat és nézzük meg és elemezzük mi történik?