Networking Assignment

Reza Hassanpour

Ahmad Omar

Afshin Amighi

Andrea Minuto

Assignment

The networking assignment requires to develop a program that will establish communication between programs over THE INTERNET. This communication involves a scenario in which clients and servers exchange data.
The assignment simulates a **distributed library application**.
We assume the *client* sends a request to the *server* (librarian) enquiring about a specific book and in case to borrow the specific book. The requested data may be in one of the data files maintained by the *helper servers* as shown in the figure below.

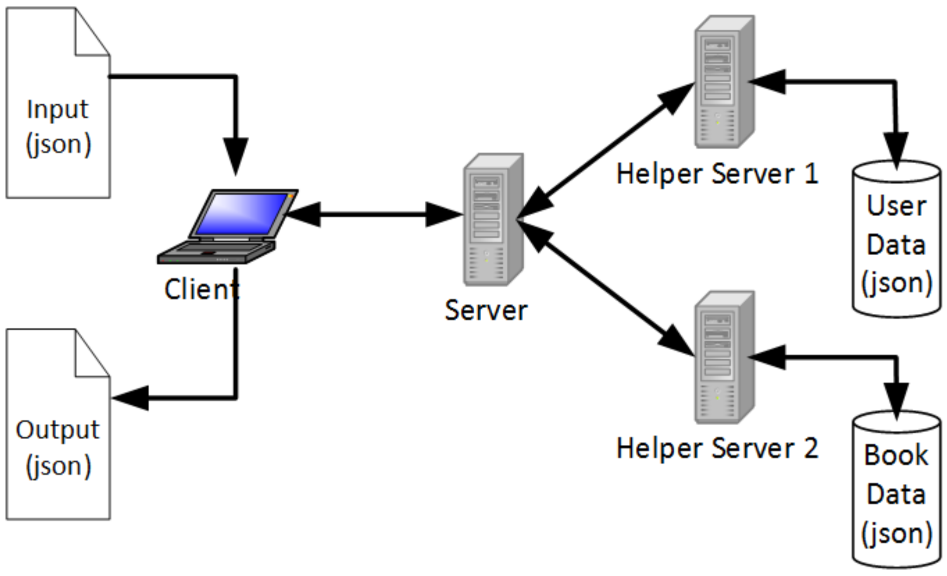
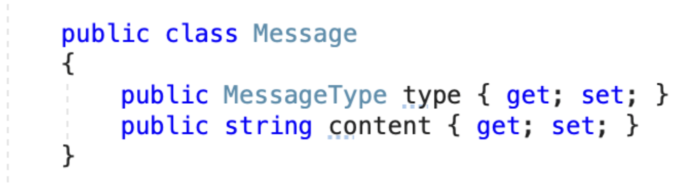


Figure 1: Topology of communications

Client:
Each *Client* program needs to read a book title from a JSON-file called “*LibInput.json*”. For each entry in this file, a client sends a query to the server asking for the *status of the book*.
After it receives the book *information* from the server it writes the book title and its status in another JSON-file called “*LibOutput.json*”. In case a book is **borrowed** it must write additionally the *name* and the email address of the person who borrowed the book. For example:

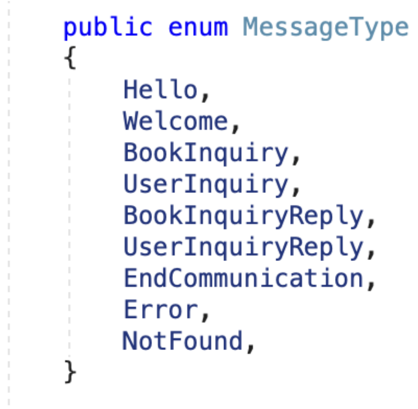
```
[{
        "Client_id": "Client 0",
        "BookName": "War and Peace",
        "Status": "Borrowed",
        "BorrowerName": "Tim Parker",
        "BorrowerEmail": "in@Pellentesqueultricies.co.uk"
   },
   {
        "Client_id": "Client 1",
        "BookName": "Harry Potter",
        "Status": "Available",
        "BorrowerName": null,
        "BorrowerEmail": null
   }]
```

Message:
Any message communicated between programs is a serialized json of the **Message** type (see the picture below):



```
public class Message
{
    public MessageType type { get; set; }
    public string content { get; set; }
}
```

The type can take one of the values defined in **MessageType**:

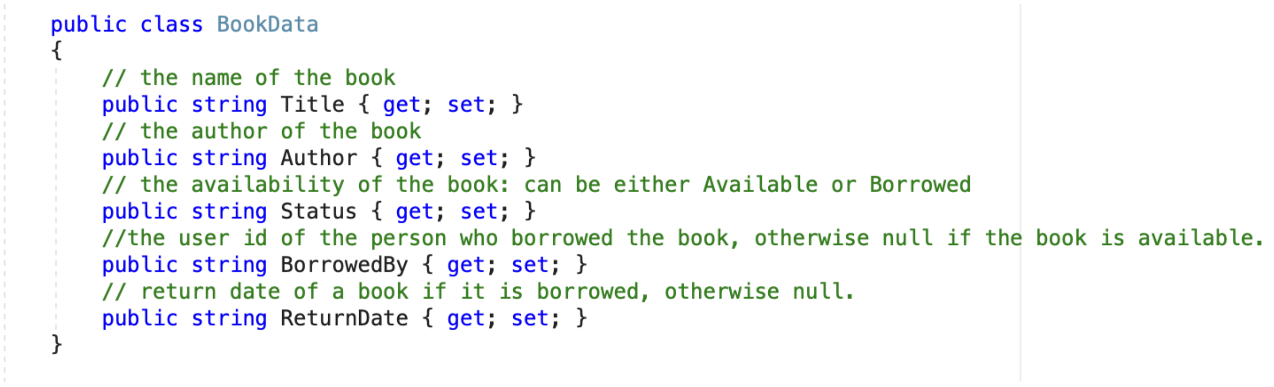


```
public enum MessageType
{
    Hello,
    Welcome,
    BookInquiry,
    UserInquiry,
    BookInquiryReply,
    UserInquiryReply,
    EndCommunication,
    Error,
    NotFound,
}
```

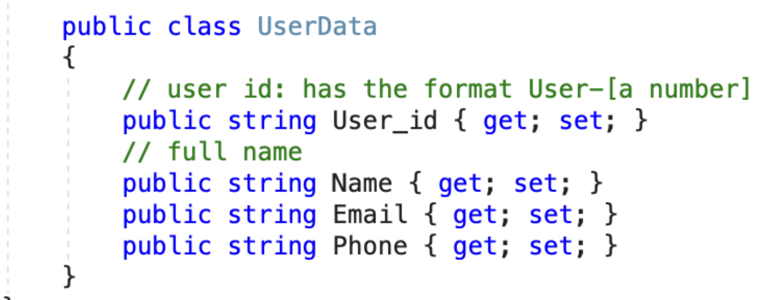
The content field of the message can be one of the following values:

- empty string which indicates the content is not important.

- *Title* of the book when the type is *BookInquiry*.
- *Id* of the user when the type is *UserInquiry*.
- Json serialized of type *UserData* when *the* type is *UserInquiryReply*.
- Json serialized of type *BookData* when the type is *BookInquiryReply*



```
public class BookData
{
    // the name of the book
    public string Title { get; set; }
    // the author of the book
    public string Author { get; set; }
    // the availability of the book: can be either Available or Borrowed
    public string Status { get; set; }
    //the user id of the person who borrowed the book, otherwise null if the book is available.
    public string BorrowedBy { get; set; }
    // return date of a book if it is borrowed, otherwise null.
    public string ReturnDate { get; set; }
}
```



```
public class UserData
{
    // user id: has the format User-[a number]
    public string User_id { get; set; }
    // full name
    public string Name { get; set; }
    public string Email { get; set; }
    public string Phone { get; set; }
}
```

Protocol:
ALL the communications should be done through ***TCP sockets*** between the client and the server, and two other **TCP sockets** between the server and the helper servers. **Helpers and clients CANNOT have direct access if not through the server (see Figure 1).** Each message has *a type* and *a content*. The type of the message defines what kind of message it is, and the content field is used to carry additional information.
For each book request **three scenarios** are possible. Below the communications for each scenario is presented as UML sequence diagram:

Book is Available

1. Each client is instantiated with a *client_id* and a *bookName*.
2. As soon as clients start their communications, they send a message with type *Hello* and content as their *client_id.*
3. The server responds with a message typed as *Welcome*. Content is not important in this message.
4. As soon as the client receives the *Welcome* message from the server, it sends its book request: message type is *BookInquiry* with *bookName* as the content.
5. The server forwards the *book* request to the BookHelper.
6. The BookHelper replies with *BookInqReply* message type. The content of this message must be the complete information extracted from given book information as a json file.
7. The server forwards the book information to the client.

8. The client extracts the *status* of the book, which should be Available in this scenario and builds an object of Output.

Book is Borrowed

1. Each client is instantiated with a *client_id* and a *bookName*.
2. As soon as clients start their communications, they send a message with type *Hello* and content as their *client_id*.
3. The server responds with a message typed as *Welcome*. Content is not important in this message.
4. As soon as the client receives the *Welcome* message from the server, it sends its book request: message type is *BookInquiry* with *bookName* as the content.
5. The server forwards the book request to the BookHelper.
6. The BookHelper replies with *BookInqReply* message type. The content of this message must be the complete information extracted from given book information as a json file.
7. The server forwards the book information to the client.
8. The client extracts the *status* of the book, which should be Borrowed in this scenario.
9. The client extracts the *user_id* from the received book information.
10. The client sends a message with type *UserInquiry* that carries the *user_id* as the content.
11. The server forwards the message *UserInquiry* to the UserHelper.
12. The UserHelper extracts the user information from the json file containing all the users.
13. The UserHelper replies to the server with the extracted user information. The message type is *UserInqReply*.
14. The server forwards the *UserInqReply* to the client.
15. The client extracts the *name* and *email* the user and builds an object of Output.

Book is not found

16. Each client is instantiated with a *client_id* and a *bookName*.
17. As soon as clients start their communications, they send a message with type *Hello* and content as their *client_id*.
18. The server responds with a message typed as *Welcome*. Content is not important in this message.

1. As soon as the client receives the *Welcome* message from the server, it sends its book request: message type is *BookInquiry* with *bookName* as the content.
2. The server forwards the book request to the BookHelper.
3. The BookHelper tries to find the requested book but it is not available in the given book information. Therefore the BookHelper *replies* with *NotFound* message type.
4. The server forwards the *NotFound* message to the client.
5. The client builds an object of Output with the name of the proper values that indicates the result of the request.

6. **Closing communications**: After all clients requesting their books, there is a special *client_id*=-1 that has an empty book name. This client, after receiving the welcome message from the server, must send a message with type "*EndCommunication*" to the server. This message indicates that

there will be no more communications and all the programs must close their communications. The server forwards the message to both helpers and closes its communications. Helpers must close their communications after receiving such a message from the server. This message has no content.

Assignment Contents:

- **Download provided codes and files from here:** **https://github.com/afshinamighi/Courses/tree/main/Networking/DistLibrary**
- There will be four C# projects, one configuration file, json files for input, books and users, and one sample output file.
- Each project contains required json files, defined classes for UserData, BookData, Message and MessageType.
- All the projects are based on a setting json file that contains configuration parameters.
- Students must follow instructions provided in the projects to implement their assignment.
- The assignment can be done in a group of maximum two students.

Grading requirements:
Any missing, imprecise, partial or incomplete requirement will be graded as a fail.
Other requirements details can be extracted from the description.

- The program should be written in .Net 5.0.*x*.
- Mandatory libraries to implement sockets and de-/ serialization of JSON are: System.Text.Json; System.Net.Sockets;
- Other built in libraries such as System, System.IO are allowed as long as it's not conflicting with the mandatory ones
- You need to use our project **template** to implement your application. Do not change any file-names in the provided template AND FILL IT APPROPRIATELY.
- When running the application there must be no errors/warnings. Apply proper exception handling where needed (failure to do so will result in a FAIL).
- All the communication between the programs must be carried with the proper socket kind with appropriate settings.
- The list of books and users will be provided as json files named *BooksData.json* and *UsersData.json in appropriate format*. Your implementation must use those files. The client program should get the names of the books to be enquired as arguments from the *input.json*.
- The structure of messages, *BookData* and *UserData*, *Output* and interface programs (main programs and clients simulator) MUST NOT CHANGE. Otherwise, your submissions will not work with the testing programs developed by the teachers.
- Your program will be tested using **different json files** (anything that is a valid format, but containing different values) therefore, make sure that the data files are read from the same location as the program running, and their names are as specified here. The path/location should be handled independently from the running operating system (Mac/Windows/Linux).
- The *LibOutput.json* should be properly formatted and respect the object data structure.
- **Use and respect the setting file provided in the project (setting.json), FAILING TO DO SO WILL RESULT IN A NEGATIVE ASSESSMENT.**

The modality for the delivery of the assignment, oral check and any follow up discussion that is required, will be shared in a second moment and this file will be updated.

Feel free to give feedback if you think there is any imperfection or doubt (please do so in the appropriate channel).