eCommerce Systems / Professor H Roumani

PROJECT A

Executive Summary

As a precursor to Project-B, you will create a TCP server using an agile development methodology. The server will adopt a custom protocol that was designed to expose a number of concepts, such as basic networking, json/xml marshaling, and cloud computing. In addition, the project will address scalability issues, such as multithreading and throttling, as well as security concerns, such as vulnerabilities and authentication.

The Server

- Developed as a POJO.
- Listens on a port of your choosing.
- Is TCP-based and restful.
- Is multithreaded.
- Has a log.
- Has a firewall.

The Protocol

- Is text based and case insensitive.
- Its request and response strings are EOL-terminated.
- Its request tokens are whitespace delimited (use regex to parse).
- Has 7 methods: *M1* thru *M7*.

The Methods

The table below describes the protocol's methods briefly. Detailed specifications are provided in lecture and in the following sections.

METHOD	REQUEST	RESPONSE
M1	getTime	The full date/time in the server's time zone, e.g. `"Mon 20 Aug 2018 16:29:46 EDT"`
M2	Bye	None. The client disconnects and the event is logged.
M3	Punch <ip>	None. The given ip is added to the firewall's whitelist.
M4	Plug <ip>	None. The given ip is removed from the firewall's whitelist.
M5	Prime <digits>	A prime with the given number of digits, e.g. if digits is 28 then `"1121350615551812829661081651"`
M6	Auth <username> <password>	Either `"Auth Failure"` or `"You are in!`

M7	Roster <course> <xml|json>	The enrollment roster of the given course in json or in xml.

If the request is none of the above (e.g. malformed or completely different) then the response should be "Don't understand <X>", where X is the request.

The Log

A text file with three fields in each record: a timestamp, a main entry, and a subentry. The events to be logged are:

- Server start (subentry: server's IP:port)
- A connection (subentry: client's IP)
- A disconnection (subentry: client's IP)
- An exception (subentry: error message / stack trace)
- A firewall violation (subentry: client's IP)
- Server shutdown (subentry: analytics, if any)

The Firewall

A hash table containing the IP's that are allowed to connect. It should be initialized to the server's own IP (so that you can test via `localhost`).

Implementation Notes

- For M5, it is OK if the number of digits in the response is slightly off but the returned integer must be prime most of the time (over 999 times in 1000). More on this in lecture or on the forum.
- For M6, create a hash table with usernames and passwords of your own choosing. The auth fails if the username is not found or if the password is incorrect (case sensitive).
- For M7, use this API and library to get a course object and then serialize it into the desired format. The best-practice approach involves using Gson or Jackson for json and JAXB for xml. Here is the Gson library. JAXB is built into the JDK. More on this in lecture or on the forum.

The Development Process

It is highly recommended that you use the virtual environment on your home machine as it is similar to the environment of the lab machines and the loaner laptops. If you don't, you will waste too much time downloading and configuring (and debugging) tens of software packages, and we will not be able to help if you get stuck.

In Eclipse, create a Java project named EECS4413 and create within it a package named "a" (stands for Project-A). All classes for Project-A will reside in this package. Download the needed jar files and add them (as external jars) to the project's build path. Do not build everything and then test it! Instead, use an iterative methodology that builds in increments and test each release before proceeding to the next. Use `telnet` to simulate clients (or `Netcat` on newer MacOS).

Here is a suggested development schedule:

Release 0.1

- Build the **`TCPServer`** class using the template provided in lecture. It is a single-threaded server that handles one connection at a time and closes the connection after issuing the response.
- The server remains available as long as a file named "*running.txt*" exists in a designated directory.

- Implement the firewall and the logging (data structures and functionality) within this class.
- Upon receiving a connection from an allowed IP, the class instantiate a helper class named **`Worker`** and passes the socket to its `handle` method.
- Build the Worker class and implement methods M1 to M4 in its `handle` method.

Release 0.2

- Implement the *keep-alive* feature: the server maintains the connection until the client disconnects (by issuing M2) or if an exception is thrown.
- Make the server multithreaded by forking a separate thread per connection for the Worker instance to operate within it.

Release 0.3

- Add support for M5.
- Add support for M6.

Release 0.4

- Add support for M7.

Persist Your Work

Follow these steps to persist / backup your work:

- Right-click your project (in the Project Explorer) and select Export.
- In the *General* category, select Archive File.
- Accept the defaults but let Eclipse create the archive in the home directory (in the VBox: `/home/user/EECS4413.zip`).

Now copy the created zip file to your Google Drive, DropBox, or some other cloud service, or copy it to a USB drive. If you later need to restore this backup into a fresh workspace, do this:

- Launch Eclipse in a new workspace.
- Right-click anywhere in the Project Explorer and select Import.
- In the *General* category, select Existing Projects into Workspace (rather than archive file!).
- Point to your archive file, select the project EECS4413, and click Finish.

Try the above procedure end-to-end (by using two machines; by switching to a different workspace on the same machine; or by deleting your workspace after the zip file has been created). Make sure you are comfortable backing up and restoring your course project. Do not wait until the day of the test to learn how to do this on a loaner laptop!

Do not delay the backup until the work is done! Do it often (at least after every release). If you are familiar with `github` then use it instead of the above (more on that in lecture or on the forum).

Reflections

- Security-wise, provide a critique of the punch/plug methods. Are they secure or do they create a vulnerability that can be exploited?
- Security-wise, provide a critique of the prime methods. Does it create a vulnerability that can be exploited?
- Security-wise, provide a critique of the way passwords are stored. Is it insecure against certain attack vectors? Is there an alternative? If so, refactor your implementation.
- Security-wise, provide a critique of the way the server handles invalid requests.
- Scalability-wise, can you throttle the connections (e.g. by imposing a limit on the number of requests per connection)? If so, refactor your implementation.

- Scalability-wise, how can thread pooling improve the server's throughput?
- Scalability-wise, can a single instance of the Worker improve the server's throughput? If so, refactor your implementation.

• EECS4413 • Professor Roumani • Don't miss the forest for the trees •