B5 - Advanced C++ Programming

B-PAV-530

Zia

Communication is key

42.0

Zia

binary name:	zia
group size:	3-6
repository name:	cpp_zia
repository rights:	ramassage-tek

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.

The goal of the **Zia** project is to create an HTTP server. This server will be able to serve typical HTTP documents and page requests, as well as CGI execution and more. The server **MUST** be written in C++, with support for interoperable modules.

INTRODUCTION

All the knowledge you've acquired from previous C++ knowledge units will be put to use in this final, large scale project. To finish this project in time, you'll need to make clever use of all the abstractions you've created up until now. This project will be your last object-oriented design and implementation at **Epitech** (except for your end of studies project, of course).

During this project, every design diagram and any code you write must be of **professional** quality. Think! Write code you'll be proud of!

However, design and implementation are not at the heart of the **Zia** project. The core part of this project is to overcome a challenge none of you have faced during your scholarship: **teaming up with your nation-wide class**! More on this will come.

PROJECT LIFETIME

The **Zia** project is split into 5 steps:

- The API follow-up
- The API election
- The first implementation follow-up
- The second implementation follow-up
- The final defense

These items will all be discussed in details below.

Common parts

+ Protocol

The **Zia MUST** be an exhaustive implementation of the **HTTP/1.1** protocol, as described in RFC 2616, with the exception of proxy support.

If you wrote your own RFC during the **R-Type** project, reading this big one shouldn't be a problem. Refer to the **R-Type**'s subject for more information about RFCs.

Here is a **NON-EXHAUSTIVE** list of what we consider mandatory to know about **RFC 2616**:

- The request structure
- The response structure
- Http methods
- Http response codes

+ Server Configuration

The server must be fully configurable by means of a configuration file. Any software configuration done by re-compiling the program (such as macros) will **NOT** be accepted and have severe consequences.

You an use any format for your configuration files, such as XML, Json, INI…

Some notes about the configuration file:

- You **MAY** use a parsing library specific to XML, Json or any other language you choose. Before you ask, **Boost::Spirit** is **NOT** authorized.
- You **MUST** implement a regular and coherent parser if you choose to not use any parsing library.
- You **MUST NOT** use XML or Json libraries for any other purpose than parsing the configuration file. This will be considered as cheating and have severe consequences.
- The server **MUST NOT** crash if it the configuration file is corrupt or missing: you **MUST** set default values.
- The configuration file **MUST NOT** be opened with an absolute path.
- It **MUST** be possible to reload the configuration file without restarting nor recompiling the server.

MODULES

As a modular server, the **Zia MUST** be able to handle modules. The **Zia executable** could be seen as an empty shell that must be filled with **modules** in order to work.

A module is an atomic processing unit that can receive input from other modules, and send output to them. It **MUST** be possible to use any number of modules together to create a processing line that will handle an HTTP request and create an appropriate HTTP response.

You **MUST** design a complete Application Programming Interface to handle you **Zia**'s modules. Your API will be evaluated during the **API follow-up**.

Although submitting your API to the election is optional, you **MUST** design one and defend it during the **API follow-up**.

Once the election is finished, your modules **MUST** conform to the nationwide API.

No matter what API is elected, each group **MUST** provide two mandatory modules for the final defense: the **secure connection module** and the **PHP CGI** module. Once these two modules work perfectly, you **MUST** add as many other modules as you like in order to raise your final grade.

As always, when designing your API, question its flexibility. How easy would it be to add a **log** module that would keep a log file of all incoming requests, and that would let other modules send it log messages? How easy would it be to add a **video game** module that would be a **Snake** clone in which each incoming request spawns a food item?

Having a flexible API will make it possible for you to add any custom behavior to your **Zia**

Interoperability

Modules from different groups **MUST** be interoperable. A module from a given group **MUST** run seamlessly in another group's server. It **MUST** be possible to add or remove modules **without recompiling or restarting the server**.

This task is particularly complex and requires some reflection

Here are a few rules:

- Modules **MUST** be able to hook up to any stage of request processing, from connection establishment to page rendering.
- The server **MUST** be able to load and unload modules dynamically.

To make it possible for your modules to run with any group's server nationwide, you **MUST** conform to the elected API.

Secure Connection Module

The server **MUST** let clients establish secure connections using **SSL** or **TLS**. This feature **MUST** be a module. You are allowed to use **OpenSSL**.

PHP CGI Module

This module **MUST** make it possible for the server to execute **PHP** scripts. The scripts **MUST** run as CGI.

DEVELOPMENT CONSTRAINTS

+ GENERAL

Your **Zia MUST** compile and run on at least one **Windows** distribution **AND** one **Unix** distribution.

On Windows, you MUST use **Microsoft Visual Compiler (MSVC)** to build your project. **Using MinGW is NOT allowed.**
Your **Zia MUST** be multi-threaded.

You **SHOULD** use all the abstractions you've designed and implemented in C++ up until now.

You **SHOULD** write **unit tests**.

You **MUST** write **C++** code. **C** and **C+** will **NOT** be tolerated.

+ LIBRARIES

Boost is forbidden in the server core but authorized in request processing modules.
Qt MAY be tolerated **for GUI purposes ONLY**.

Any library that is not explicitly authorized is forbidden.

Testing modules and servers

During your final defense, your server must be usable with these client programs:

- Telnet
- Lynx
- Google Chrome
- Mozilla Firefox
- Opera
- MS Edge
- Siege

Project steps

+ API follow-up

Although submitting your API for the election is a choice, defending **YOUR** API during this follow-up is mandatory.

This follow-up will be taken into account when validating the knowledge unit.

Not registering, or not having designed an API, will have severe consequences.

+ API Election

The election will last from the day following the API follow-up until the votes are closed. Refer to the **Yammer** group for the list of candidate APIs and a poll to vote. A few important rules:

- Submitting your API is optional
- Teachers and assistants are forbidden from interacting with you during this period. You are alone, and must make your own choices as to which API is the most flexible.
- The API election is the same nationwide. Candidates **MUST** communicate with ALL other cities.
- If the elected group ignores help requests from other groups or cities, it will face severe consequences.
- Having your API elected for nationwide use is rewarded with a **LOT** of points during the final defense.

Once the election is finished, you will start implementing your **Zia** by **CONFORMING TO** the elected API.

+ Implementation follow-ups

These follow-ups let us assert your progress on the project and evaluate its quality. It is very important that you be very dynamic during these follow-ups, as they are the last steps before the final defense.

+ Final defense

During this final defense, teachers will not only evaluate your product, but also your **personal C++ knowledge**. Each of you will spend a small part of the defense face to face with a teacher who will ask you a simple question about **C++** and **Object-Oriented Design** basics.

If you can't answer this question, even if your group's project was perfect, you **WILL** fail the knowledge unit. You've been warned.

Read that last paragraph one more time.

Groups and Realization

The same grade will be given to the entire group, but knowledge unit validation **WILL** differ depending on your **personal skills and implication**.

If the teacher requests it, or if over 50% of the group members request it, the grade can be made individual to avoid lazy students hiding in working groups.

General setpoints

You are (more or less) free to implement the client and server any way you please. However, here are a few restrictions:

- The only authorized functions from the **libc** are the ones that wrap system calls (and don't have C++ equivalents!)
- Any solution to a problem **MUST** be object-oriented.
- Any not explicitly authorized library is explicitly forbidden.
- Any value passed by copy instead of reference or pointer MUST be justified, or you'll lose points.
- Any member function or method that does not modify the current instance not **const** MUST be justified, or you'll lose points.
- Koalas don't use any **C++** norm. However, any code that is deemed unreadable, unmaintainable or with unnecessary performance costs **WILL** be arbitrarily sanctioned. Be rigorous! Write code you'll be proud of!
- Any conditional branching longer than **if ... else if ... else ...** is FORBIDDEN. Factorize! Use the STL's **associative containers**.
- Keep an eye on this subject regularly, it could be modified.
- We pay great attention to our subjects. If you run into typos, spelling mistakes or inconsistencies, please contact us so we can correct it.
- You can contact the authors by mail. Their adresses can be found on the module's page, under "Module Designers"
- The **C++** Yammer group will contain information and answers to your questions. Please make sure the answer to your question can't be found there before contacting the authors.