

МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №** 8

з дисципліни “ Основи програмування ”

Тема “**HTTP сервер** ”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Виконав  студент I курсу  групи КП-61  Свинарчук Максим Владиславович  (*прізвище, ім’я, по батькові*)  варіант №23 | |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 2017 р.  викладач  Гадиняк Руслан Анатолійович  (*прізвище, ім’я, по батькові*) | |
| Штрафні бали:   |  |  | | --- | --- | | **Термін здачі** | **Оформлення звіту** | |  |  | | Нараховані бали:   |  |  |  | | --- | --- | --- | | **Корект. програм (2 бала)** | **Відп. на теор. питання (1 бал)** | **Відп. на прогр. питання (2 бала)** | |  |  |  | | | | Сумарний бал:   |  | | --- | |  | |

Київ 2017

**Мета роботи**

Навчитися створювати простий web-сервер із взаємодією через протокол HTTP. Навчитися оформляти віддалений Git репозиторій із кодом, перевіряти збірки проекту за допомогою безперервної інтеграції та створювати документацію коду проекту.

**Постановка завдання**

**Завдання №1. Створення web-сервера**

Створити TCP-сервер, що слухає запити на підключення TCP клієнтів і взаємодіє з ними за допомогою протоколу HTTP. Сервер повинен обробляти лише HTTP GET запити і повертати коректно-сформовані HTTP-відповіді. Інтерфейс сервера дозволяє звертатися до нього по HTTP-шляхах:

* / - відправити у відповідь JSON-об'єкт із інформацією про сервер:
* {  
   "title": "%SERVER\_NAME%",   
   "developer": "%YOUR\_NAME\_SURNAME%",   
   "time": "%SERVER\_TIME%"  
  }
* /favorites - JSON-список із об'єктів, що позначають список ваших улюблених місць. Кожен такий об'єкт має мати декілька полів, серед яких обов'язково має бути числове поле id із унікальним для кожного об'єкта значенням.
* /favorites?{key}={value} - JSON-підсписок тих об'єктів зі списку /favorites, у яких поле {key} має значення {value}. Ключ {key}, по якому відбувається фільтрація списку, обрати довільно.
* /favorites/{id} - JSON-об'єкт зі списку /favorites за ідентифікатором з числовим значенням {id}. Якщо об'єкта із таким значенням id не було знайдено, повертати у відповідь HTTP статус 404 Not Found.
* /file - зчитати вміст файлу data.txt, який попередньо розмістити у директорії data/ проекту. Відправити клієнту JSON-об'єкт, що містить у собі інформацію про назву, розмір (у байтах) і текстовий вміст файлу.
* /file/data - у відповідь клієнту відправити JSON-об'єкт із інформацією про кількість літер у верхньому регістрі та кількість літер у нижньому регістрі, які містить файл (файл data.txt).

**Завдання №2. Створення документації проекту за допомогою Doxygen**

**Коментування коду у форматі Doxygen**

Виконати коментування інтерфейсів модулів у форматі Doxygen ([приклад](https://github.com/PublicHadyniak/libprogbase/blob/master/include/progbase/list.h)).

Для цього кожен заголовочний файл повинен починатися на коментар виду:

*/\*\* @file  
 @brief Some text about this header  
\*/*

Перед кожною визначеною структурою даних має бути коментар:

*/\*\*  
 @struct List  
 @brief defines a generic List data type  
\*/*

Перед кожною функцією інтерфейсу також розмістити коментар із інформацією про призначення функції, її параметри та результат, що повертається:

*/\*\*  
 @brief remove item from list by value  
 @param ref - reference to memory from where to read compared value  
 @returns true if success removal  
\*/*  
**bool** **List\_remove**(List \* self, **void** \* ref);

**Генерація HTML-документації утилітою Doxygen**

Встановити Doxygen:

sudo apt-get install doxygen graphvis

Перейти у директорію проекта і згенерувати файл із конфігурацією (Doxyfile):

doxygen -g

Задати у Doxyfile такі значення для вказаних налаштувань:

PROJECT\_NAME = "Project Name Here"  
EXTRACT\_ALL = NO  
FILE\_PATTERNS = \*.c \*.cpp \*.h  
RECURSIVE = YES

Згенерувати документацію командою:

doxygen Doxyfile

Doxygen створить директорію docs/ із піддиректоріями html/ і latex/, які міститимуть документацію у різних форматах (не забудьте додати docs/ у .gitignore).

Для перегляду згенерованої документації перейдіть у директорію html/ і відкрийте файл index.html у веб-браузері.

**Завдання №3. Безперервна інтеграція (Continuous Integration)**

**Cmake**

Налаштувати збірку проекта із сервером за допомогою утиліти cmake та файлу CMakeLists.txt ([приклад](https://github.com/PublicHadyniak/c-project-template/blob/master/CMakeLists.txt)).

**GitHub репозиторій**

Зареєструватись на сайті [Github](https://github.com/) та створити віддалений публічний репозиторій, у який надіслати файли проекту. Додатково створити у корені проекту файл README.md, у якому у форматі [Markdown](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet) описати варіант завдання лабораторної роботи та короткий опис модулів проекту.

**Додаткові скрипти для збірки проекта**

Створити у проекті директорію scripts/ і додати у неї файл install-libprogbase.sh:

**#!/bin/sh**  
wget -O libprogbase.zip https://github.com/PublicHadyniak/libprogbase/archive/master.zip  
unzip libprogbase.zip -d libprogbase  
cd libprogbase/libprogbase-master ; sudo make install  
cd ../../ ; rm libprogbase/ libprogbase.zip -rf

**Travis Continuous Integration**

Додати у кореневу директорію проекта файл .travis.yml із вмістом:sudo: required

language: c

os:

- linux

compiler:

- gcc

before\_install:  
 - sudo apt-get install libssl-dev

- sh scripts/install-libprogbase.sh

script:

- cmake .

- make .

Зареєструватись на сайті [Travis-CI](https://travis-ci.org/) та синхронізувати його із профілем Github. Налаштувати у Travis-CI збірку репозиторія зі сервером. Після кожного push комітів у віддалений репозиторій GitHub, Travis-CI збиратиме проект і оновлюватиме статус його збірки (passing | failing). Додати у файл README.md репозиторія мітку із зображенням статусу збірки та посиланнями на історію збірок :

[![Travis-CI][travis-badge]][travis-builds]  
  
[travis-badge]: https://travis-ci.org/PublicHadyniak/libprogbase.svg?branch=master  
[travis-builds]: https://travis-ci.org/PublicHadyniak/libprogbase/builds

де замінити PublicHadyniak на ваш нікнейм на GitHub, а libprogbase на назву репозиторія із проектом сервера.

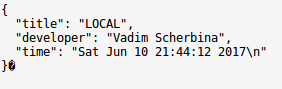
**Тексти коду програм**

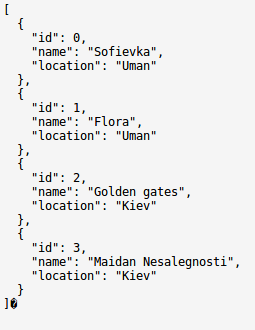
|  |
| --- |
| main.cpp |
| #include <iostream>  #include "http\_server.h"  int main()  {  http\_server();  return 0;  } |

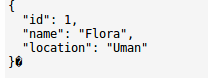
|  |
| --- |
| http\_server.cpp |
| #include <progbase-cpp/net.h>  #include <iostream>  #include <ctime>  #include <fstream>  #include <climits>  #include <cmath>  #include "http\_server.h"  #include "request.h"  #include "song.h"  #include <jansson.h>  using namespace progbase::net;  using namespace std;  const string filePath = "/home/valzavator/progbase2/labs/lab8/HTTP-server/data/data.txt";  void http\_server(void)  {  vector<Song\*> songs;  songsInit(songs);  const int serverPort = 3000;  TcpListener listener;  NetMessage message(10240);  try {  listener.bind(IpAddress("127.0.0.1", serverPort));  listener.start();  while (true) {  cout << ">> Waiting for clients at " << serverPort << "..." << endl;  TcpClient \* client = listener.accept();  client->receive(message);  cout << ">> Received: " << endl << message.dataAsString() << endl;  string data = message.dataAsString();  string response = createResponse(data, songs);  message.setDataString(response);  client->send(message);  cout << ">> Response sent." << endl;  delete client;  }  } catch(NetException const & exc) {  cerr << exc.what() << endl;  }  for (int i = 0; i < (int)songs.size(); i++) delete songs[i];  }  string createResponse(string message, vector<Song \*> songs)  {  Request request;  string msg;  string response =  "HTTP/1.1 400 Bad Request\r\n"  "Server: nginx/1.11.5\r\n"  "Connection: close\r\n\r\n";  if (request.parseString(message)) {  bool failRequest = false;  if (request.getURL() == "/") msg = serverInformation();  else if (request.getURL() == "/favorites") msg = myFavouriteSongs(songs);  else if (request.getURL().find("/favorites?") != string::npos) msg = keySongs(songs, request.getKey(), request.getValue());  else if (request.getURL().find("/favorites/") != string::npos) msg = keySongs(songs, "id", request.getValue());  else if (request.getURL() == "/file") msg = fileInformation();  else if (request.getURL() == "/file/data") msg = fileNumber();  else failRequest = true;  if (msg == "No such songs") {  response =  "HTTP/1.1 404 Not Found\r\n"  "Server: nginx/1.11.5\r\n"  "Date: Sun, 07 May 201 23:28:13 GMT\r\n"  "Connection: close\r\n\r\n";  } else if (!failRequest) {  response =  "HTTP/1.1 200 OK\r\n"  "Server: nginx/1.11.5\r\n"  "Date: Sun, 07 May 2017 23:28:13 GMT\r\n"  "Content-Type: application/json; charset=utf-8\r\n"  "Content-Length: " + to\_string(msg.size()) + "\r\n"  "Connection: close\r\n\r\n" + msg;  }  }  return response;  }  string serverInformation(void)  {  json\_t \* json = json\_object();  json\_object\_set\_new(json, "title", json\_string( "LAB\_8"));  json\_object\_set\_new(json, "developer",json\_string( "Maksym Svynarchuk"));  time\_t seconds = time(NULL);  tm\* timeinfo = localtime(&seconds);  json\_object\_set\_new(json, "time", json\_string(asctime(timeinfo)));  char \* jsonString = json\_dumps(json, JSON\_INDENT(2) | JSON\_PRESERVE\_ORDER);  string toReturn(jsonString);  free(jsonString);  json\_decref(json);  return toReturn;  }  string myFavouriteSongs(vector<Song\*> songs)  {  json\_t \* array = json\_array();  for (int i = 0; i < (int)songs.size(); i++) {  json\_t \* json = json\_object();  json\_object\_set\_new(json, "Name", json\_string(songs[i]->getName().c\_str()));  json\_object\_set\_new(json, "style", json\_string(songs[i]->getStyle().c\_str()));  json\_object\_set\_new(json, "songwriter", json\_string(songs[i]->getSongwriter().c\_str()));  json\_object\_set\_new(json, "duration", json\_real(songs[i]->getDuration()));  json\_object\_set\_new(json, "id", json\_integer(songs[i]->getId()));  json\_array\_append(array, json);  json\_decref(json);  }  char \* jsonString = json\_dumps(array, JSON\_INDENT(2) | JSON\_PRESERVE\_ORDER);  string toReturn(jsonString);  free(jsonString);  json\_decref(array);  return toReturn;  }  string keySongs(std::vector<Song \*> songs, std::string key, std::string value)  {  json\_t \* array = json\_array();  bool check = false;  for (int i = 0; i < (int)songs.size(); i++) {  if (songs[i]->getField(key) == value) {  check = true;  json\_t \* json = json\_object();  json\_object\_set\_new(json, "Name", json\_string(songs[i]->getName().c\_str()));  json\_object\_set\_new(json, "style", json\_string(songs[i]->getStyle().c\_str()));  json\_object\_set\_new(json, "songwriter", json\_string(songs[i]->getSongwriter().c\_str()));  json\_object\_set\_new(json, "duration", json\_real(songs[i]->getDuration()));  json\_object\_set\_new(json, "id", json\_integer(songs[i]->getId()));  json\_array\_append(array, json);  json\_decref(json);  }  }  if (!check) return "No such songs";  char \* jsonString = json\_dumps(array, JSON\_INDENT(2) | JSON\_PRESERVE\_ORDER);  string toReturn(jsonString);  free(jsonString);  // json\_decref(array);  return toReturn;  }  string fileInformation()  {  string fileContent = readFile(filePath.c\_str());  if (fileContent == "Can't open file") {  cerr << "Can't open file" << endl;  return "Can't open file";  }  json\_t \* json = json\_object();  json\_object\_set\_new(json, "filename", json\_string( "data.txt"));  json\_object\_set\_new(json, "file size(byte)",json\_integer(fileContent.size()));  json\_object\_set\_new(json, "file content", json\_string(fileContent.c\_str()));  char \* jsonString = json\_dumps(json, JSON\_INDENT(2) | JSON\_PRESERVE\_ORDER);  string toReturn(jsonString);  free(jsonString);  json\_decref(json);  return toReturn;  }  string fileNumber()  {  string str = readFile(filePath.c\_str());  double min = INT\_MAX, count = 0;  double num = min;  int index = str.find\_first\_of("1234567890");  while (index != (int)string::npos)  {  bool dot = false;  bool m = false;  string buffer;  while (isdigit(str[index]) || (str[index] == '.' && !dot) || (str[index] == '-' && !m)) {  if (str[index] == '.' && !dot) dot = true;  if (str[index] == '-' && !m) m = true;  buffer += str[index];  index++;  }  if (buffer.find\_first\_of("1234567890") != string::npos) num = stod(buffer);  if (min-num > -0.0001) min = num;  count++;  index = str.find\_first\_of("-.1234567890",index+1);  }  json\_t \* json = json\_object();  json\_object\_set\_new(json, "count of numbers",json\_integer(count));  json\_object\_set\_new(json, "smallest number",json\_real(min));  char \* jsonString = json\_dumps(json, JSON\_INDENT(2) | JSON\_PRESERVE\_ORDER);  string toReturn(jsonString);  free(jsonString);  json\_decref(json);  return toReturn;  }  string readFile(const char \* filePath) {  string line;  string toReturn;  ifstream myfile (filePath);  if (!myfile.is\_open()){  return "Can't open file";  }  while (! myfile.eof() ){  getline (myfile,line);  toReturn += line;  }  myfile.close();  return toReturn;  } |
| request.cpp |
| #include "request.h"  Request::Request() {  \_command = "";  \_URL = "";  \_key = "";  \_value = "";  }  Request::~Request() {}  bool Request::parseString(std::string message)  {  ////////// parse command /////////  int posCommand = message.find(" ");  if (posCommand != (int)std::string::npos && posCommand > 2) {  \_command = message.substr(0, posCommand);  } else return false;  ////////// parse URL /////////  int pos1 = message.find("GET ");  int pos2 = message.find(" HTTP/1.");  if (pos1 != (int)std::string::npos && pos2 != (int)std::string::npos) {  \_URL = message.substr(pos1 + 4, pos2 - pos1 - 4);  int posKey1 = \_URL.find("?");  int posKey2 = \_URL.find("=");  if (posKey1 != (int)std::string::npos && posKey2 != (int)std::string::npos) {  \_key = \_URL.substr(posKey1 + 1, posKey2 - posKey1 - 1);  \_value = \_URL.substr(posKey2 + 1, \_URL.size() - posKey2 - 1);  } else {  int posValue = \_URL.find("/favorites/");  if (posValue != (int)std::string::npos)  \_value = \_URL.substr(posValue + 11, \_URL.size() - posValue - 11);  }  }  else return false;  return true;  }  void Request::setCommand(std::string command)  {  \_command = command;  }  void Request::setURL(std::string URL)  {  \_URL = URL;  }  void Request::setKey(std::string key)  {  \_key = key;  }  void Request::setValue(std::string value)  {  \_value = value;  }  std::string Request::getCommand()  {  return \_command;  }  std::string Request::getURL()  {  return \_URL;  }  std::string Request::getValue()  {  return \_value;  }  std::string Request::getKey()  {  return \_key;  } |
| song.cpp |
| #include "song.h"  Song::Song() {}  Song::Song(std::string name, std::string style, std::string songwriter, float duration, int id)  {  \_name = name;  \_style = style;  \_songwriter = songwriter;  \_duration = duration;  \_id = id;  }  Song::~Song() {}  void Song::setName(std::string name)  {  \_name = name;  }  void Song::setStyle(std::string style)  {  \_style = style;  }  void Song::setSongwriter(std::string songwriter)  {  \_songwriter = songwriter;  }  void Song::setDuration(float duration)  {  \_duration = duration;  }  void Song::setId(int id)  {  \_id = id;  }  std::string Song::getName()  {  return \_name;  }  std::string Song::getStyle()  {  return \_style;  }  std::string Song::getSongwriter()  {  return \_songwriter;  }  float Song::getDuration()  {  return \_duration;  }  int Song::getId()  {  return \_id;  }  std::string Song::getField(std::string field)  {  if (field == "name") return \_name;  if (field == "style") return \_style;  if (field == "songwriter") return \_songwriter;  if (field == "duration") return std::to\_string(\_duration);  if (field == "id") return std::to\_string(\_id);  return NULL;  }  void songsInit(std::vector<Song\*> &songs)  {  songs.push\_back(new Song("Symphony No 9", "symphony","Ludwig van Beethoven", 24.37, 1));  songs.push\_back(new Song("Believer", "electric\_blues","Imagion Dragons", 3.23, 2));  songs.push\_back(new Song("When You Gonna Learn?", "pop\_music","Jamiroquai", 3.55, 3));  songs.push\_back(new Song("Junk Of The Heart (Happy)", "pop\_music","The Kooks", 3.08, 4));  songs.push\_back(new Song("Stressed Out", "pop\_rock","Twenty One Pilots", 3.21, 5));  songs.push\_back(new Song("Automaton", "techno","Jamiroquai", 4.48, 6));  songs.push\_back(new Song("Pumped Up Kicks", "pop\_music","Foster The People", 4.13, 7));  songs.push\_back(new Song("Ride", "rock","Twenty One Pilot", 3.32, 8));  songs.push\_back(new Song("Bartholomew", "pop\_rock","The Silent Comedy", 3.46, 9));  songs.push\_back(new Song("Way Down We Go", "pop\_music","Kaleo", 3.39, 10));  } |

|  |
| --- |
| http\_server.h |
| /\*\* @file  @brief - defines Server  \*/  #ifndef HTTP\_SERVER\_H  #define HTTP\_SERVER\_H  #include <iostream>  #include <vector>  #include <song.h>  /\*\*  @brief - HTTP server(works only with GET request)  \*/  void http\_server(void);  /\*\*  @brief - HTTP server(works only with GET request)  @param message - data from client  @param songs  @return string - response from server  \*/  std::string createResponse(std::string message, std::vector<Song \*> songs);  /\*\*  @brief - create information about server  @return response to /  \*/  std::string serverInformation(void);  /\*\*  @brief - create information about my favorites songs  @param songs - my favorites songs  @return response on /favorites  \*/  std::string myFavouriteSongs(std::vector<Song\*> songs);  /\*\*  @brief - create information about some song, which key = value  @param songs - my favorites songs  @param key - name of field  @param value - value of field  @return response on /favorites?influence={value} and /favorites/{id}  \*/  std::string keySongs(std::vector<Song\*> songs, std::string key, std::string value);  /\*\*  @brief - create information about file "data.txt"  @return response to /file  \*/  std::string fileInformation(void);  /\*\*  @brief - create information about numbers in file "data.txt"  @return response to /file/data  \*/  std::string fileNumber(void);  /\*\*  @brief - read file  @param filePath - path to the file and his name  @return information from file  \*/  std::string readFile(const char \* filePath);  #endif // HTTP\_SERVER\_H |
| request.h |
| /\*\* @file  @brief - describes a Request  \*/  #ifndef REQUEST\_H  #define REQUEST\_H  #include <iostream>  class Request  {  std::string \_command;  std::string \_URL;  std::string \_key;  std::string \_value;  public:  /\*\*  @brief - simple constructor  \*/  Request();  /\*\*  @brief - destructor  \*/  ~Request();  /\*\*  @brief - simple constructor  \*/  bool parseString(std::string message);  /\*\*  @brief - set command  @param command - command of request  \*/  void setCommand(std::string command);  /\*\*  @brief - set URL  @param URL - URL of request  \*/  void setURL(std::string URL);  /\*\*  @brief - set key  @param key - key of request  \*/  void setKey(std::string key);  /\*\*  @brief - set value  @param value - value of request  \*/  void setValue(std::string value);  /\*\*  @brief - get command of request  @return command  \*/  std::string getCommand(void);  /\*\*  @brief - get URL of request  @return URL  \*/  std::string getURL(void);  /\*\*  @brief - get key of request  @return key  \*/  std::string getKey(void);  /\*\*  @brief - get value of request  @return value  \*/  std::string getValue(void);  };  #endif // REQUEST\_H |
| song.h |
| /\*\* @file  @brief - describes a Song  \*/  #ifndef SONG\_H  #define SONG\_H  #include <iostream>  #include <vector>  class Song  {  std::string \_name;  std::string \_style;  std::string \_songwriter;  float \_duration;  int \_id;  public:  /\*\*  @brief - simple constructor  \*/  Song();  /\*\*  @brief - constructor  @param name - name of song  @param style - style of song  @param songwriter - author of song  @param duration - duration of song  @param id - id of song  \*/  Song(  std::string name,  std::string style,  std::string songwriter,  float duration,  int id  );  /\*\*  @brief - destructor  \*/  ~Song();  /\*\*  @brief - set name of song  @param name - name of song  \*/  void setName(std::string name);  /\*\*  @brief - set style of song  @param style - style of song  \*/  void setStyle(std::string style);  /\*\*  @brief - set songwriter  @param songwriter - author of song  \*/  void setSongwriter(std::string songwriter);  /\*\*  @brief - set duration of song  @param duration - duration of song  \*/  void setDuration(float duration);  /\*\*  @brief - set id of song  @param id - id of song  \*/  void setId(int id);  /\*\*  @brief - get name of song  @return name  \*/  std::string getName(void);  /\*\*  @brief - get style of song  @return style  \*/  std::string getStyle(void);  /\*\*  @brief - get id of song  @return songwriter  \*/  std::string getSongwriter(void);  /\*\*  @brief - get duration of song  @return duration  \*/  float getDuration(void);  /\*\*  @brief - get id of song  @return id  \*/  int getId(void);  /\*\*  @brief - get some field of song  @param field - name of field  @return field  \*/  std::string getField(std::string field);  };  void songsInit(std::vector<Song\*> &songs);  #endif // SONG\_H |

**Приклади результатів**







**Висновки**

Виконавши дану лабораторну роботу було створено HTTP сервер, оформлено віддалений Git репозиторій із кодом, перевірено збірки проекту за допомогою безперервної інтеграції та створено документацію коду проекту.