

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 “НТТР сервер”

|  |  |  |
| --- | --- | --- |
| Виконав(ла)  студент(ка) I курсу  групи КП-62  Теслюк Ілля Ігорович  (*прізвище, ім’я, по батькові*)  варіант №\_\_\_\_\_\_\_ |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладач  Гадиняк Руслан Анатолійович  (*прізвище, ім’я, по батькові*) |

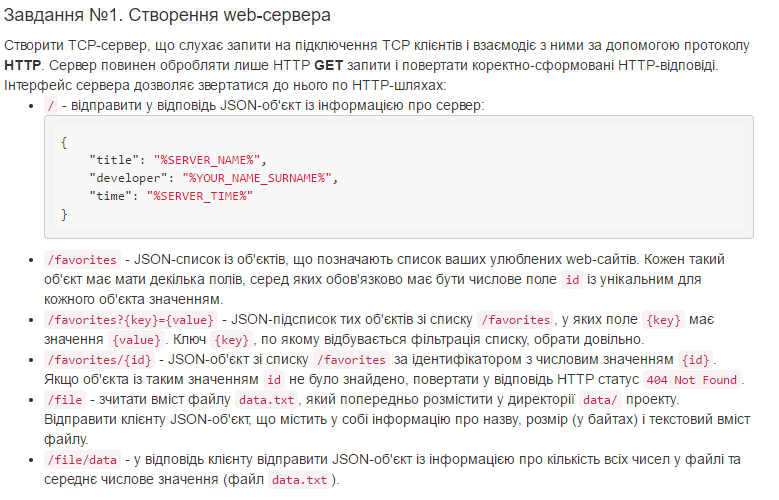
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Штрафні бали:   |  |  | | --- | --- | | **Термін здачі** | **Оформлення звіту** | |  |  | | Нараховані бали:   |  |  |  | | --- | --- | --- | | **Корект. програм (2 бала)** | **Відп. на теор. питання (1 бал)** | **Відп. на прогр. питання (2 бала)** | |  |  |  | | Сумарний бал:   |  | | --- | |  | |

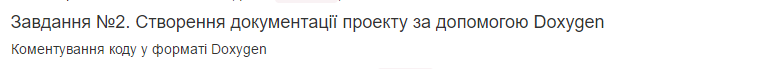
Київ 2017

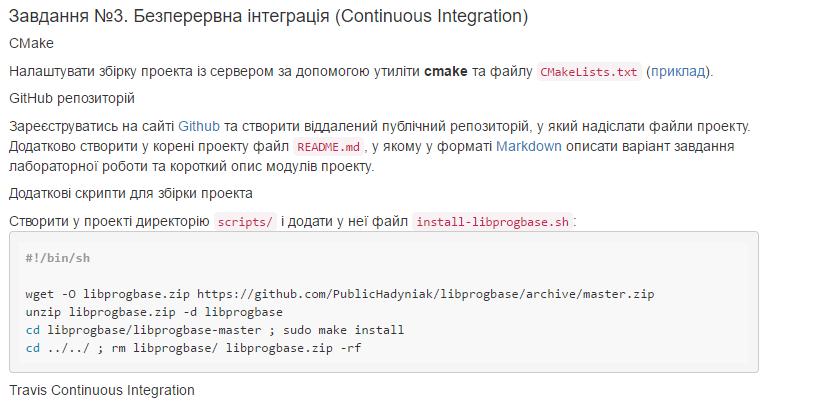
**Мета роботи**

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**









**Тексти коду програм**

|  |
| --- |
| server.c |
| #include <stdlib.h>  #include <stdio.h>  #include <errno.h>  #include <string.h>  #include <ctype.h>  #include <time.h>  #include <progbase/net.h>  #include <progbase/list.h>  #include "website.h"  #include "request.h"  #include "storage.h"  #include <time.h>  #define BUFFER\_LEN 10240  int main(int argc, char \* argv[]) {  if (argc < 2) {  puts("Please, specify server port in command line arguments");  return 1;  }  srand(time(0));  const int port = atoi(argv[1]);  TcpListener \* server = TcpListener\_init(&(TcpListener){});  IpAddress \* address = IpAddress\_initAny(&(IpAddress){}, port);  if(!TcpListener\_bind(server, address)) {  perror("tcp bind");  return 1;  }  if (!TcpListener\_start(server)) {  perror("tcp server start");  return 1;  }  printf("TCP Server is listening on port %d\n",  IpAddress\_port(TcpListener\_address(server)));    NetMessage \* message = NetMessage\_init(  &(NetMessage){},  (char[BUFFER\_LEN]){},  BUFFER\_LEN);  TcpClient client;  List\* websites = Storage\_readAsJson("source.json");  while (1) {  puts(">> Waiting for connection...");  TcpListener\_accept(server, &client);    if(!TcpClient\_receive(&client, message)) {  perror("recv");  return 1;  }  IpAddress \* clientAddress = TcpClient\_address(&client);  printf(">> Received message from %s:%d (%d bytes): `%s`\n",  IpAddress\_address(clientAddress),  IpAddress\_port(clientAddress),  NetMessage\_dataLength(message),  NetMessage\_data(message));  Request req = Request\_parse(NetMessage\_data(message));  Response res;  Response\_init(&res);  Request\_process(&req,&res, websites);  Response\_toMessage(&res, message);    if(!TcpClient\_send(&client, message)) {  perror("send");  return 1;  }  printf(">> String sent to client:\r\n%s\r\n", NetMessage\_data(message));  TcpClient\_close(&client);  }  TcpListener\_close(server);  List\_free(&websites);  return 0;  } |

|  |
| --- |
| request.c |
| #include <string.h>  #include <ctype.h>  #include <jansson.h>  #include "request.h"  #include "website.h"  #include "list.h"  #include "storage.h"  #include <time.h>  void Response\_badRequest(Response\* responce) {  responce->status = 400;  strcpy(responce->description,"Bad request");  }  void Response\_notFound(Response\* responce) {  responce->status = 404;  strcpy(responce->description,"Not found");  }  void Response\_success(Response\* responce, char\* data) {  responce->status = 200;  strcpy(responce->description, "OK");  strcpy(responce->data, data);  }  void Response\_init(Response \* self) {  self->status = 0;  strcpy(self->description, "");  strcpy(self->data, "");  }  char\* Response\_serverInfo(void) {  const char name[] = "TCP/HTTP server";  const char surname[] = "Tesliuk";    time\_t now = time(NULL);  struct tm \* local = localtime(&now);  char serverTime[50];  strftime(serverTime, 50, "%X", local);    json\_t \* jobj = json\_object();  json\_object\_set(jobj, "title", json\_string(name));  json\_object\_set(jobj, "developer", json\_string(surname));  json\_object\_set(jobj, "time", json\_string(serverTime));  char \*jStr = json\_dumps(jobj, JSON\_COMPACT);  json\_decref(jobj);  return jStr;  }  Request Request\_parse(const char \* str) {  Request req = {  .command = ""  };  if(str[0] != '\0') {  char\* pch = strstr(str, " HTTP");  strncpy(req.command, str, pch-str);  }  return req;  }  void Request\_process(Request\* request, Response\* responce, List\* websites) {  char\* command = request->command;  if(strncmp(command, "GET",3) == 0) {  char\* begin = strchr(command, '/')+1;  if(\*begin == ' ' || strlen(begin) == 0) {  Response\_success(responce, Response\_serverInfo());  } else {  if(strstr(begin,"favourites") != NULL) {  begin += strlen("favourites");  if(\*begin == ' ' || strlen(begin) == 0) {  Response\_success(responce, List\_toJson(websites));  } else if(\*begin == '?') {  begin++;  char key[20];  char value[20];  char\* sign = strchr(begin, '=');  strncpy(key, begin, sign-begin);  sign++;  strcpy(value, sign);    List\* list = List\_filter(websites,key, value);  if(list == NULL || List\_count(list) == 0) {  Response\_notFound(responce);  } else {  Response\_success(responce, List\_toJson(list));  }  } else if(\*begin == '/') {  int id = -1;  sscanf(begin,"/%i",&id);  int index = List\_findById(websites, id);  if(index != -1) {  Website \* node = List\_get(websites,index);  Response\_success(responce,Website\_toJsonString(node));  } else {  Response\_notFound(responce);  }  } else {  Response\_badRequest(responce);  }  } else if(strstr(begin, "file") != NULL) {  begin += strlen("file");  if(strlen(begin)==0) {  Response\_success(responce, Storage\_fileToJson("data.txt"));  } else if(strstr(begin, "/data") != NULL) {  Response\_success(responce, Storage\_fileContentInfo("data.txt"));  } else {  Response\_badRequest(responce);  }  } else {  Response\_badRequest(responce);  }  }  } else {  Response\_badRequest(responce);  }  }  void Response\_toMessage(Response\* res,NetMessage \* message) {  char msg[BUFFER\_SIZE];  if(strlen(res->data)!=0) {  sprintf(msg,"HTTP/1.0 %i %s\n"  "Content-type: text/json\n"  "Content-length: %i\n"  "%s",res->status,res->description,strlen(res->data),res->data);  } else {  sprintf(msg,"HTTP/1.0 %i %s",res->status,res->description);  }  NetMessage\_setDataString(message, msg);  } |
| list.c |
| #include "list.h"  #include <string.h>  #include <jansson.h>  #include "website.h"  int List\_findById(List\* self, int id) {  for(int i = 0; i < List\_count(self); i++) {  Website \* node = List\_get(self, i);  if(Website\_getId(node) == id) {  return i;  }  }  return -1;  }  List\* List\_filter(List\* self, char\* key, char\* value) {  if(strcmp(key, "name") == 0) {  return List\_filterByName(self, value);  } else if(strcmp(key, "id") == 0) {  return List\_filterById(self, atoi(value));  } else if(strcmp(key, "country") == 0) {  return List\_filterByCountry(self, value);  } else if(strcmp(key, "year") == 0) {  return List\_filterByYear(self, atoi(value));  } else if(strcmp(key, "type") == 0) {  return List\_filterByType(self, value);  } else {  return NULL;  }  }  List\* List\_filterByName(List\* self, char\* name) {  List\* list = List\_new();  for(int i = 0; i < List\_count(self); i++) {  Website \* site = List\_get(self, i);  if(strcmp(Website\_getName(site),name) == 0) {  List\_add(list, site);  }  }  return list;  }  List\* List\_filterById(List\* self, int id) {  List\* list = List\_new();  List\_add(list, List\_get(self, List\_findById(self, id)));  return list;  }  List\* List\_filterByCountry(List\* self, char\* country) {  List\* list = List\_new();  for(int i = 0; i < List\_count(self); i++) {  Website \* site = List\_get(self, i);  if(strcmp(Website\_getCountry(site),country) == 0) {  List\_add(list, site);  }  }  return list;  }  List\* List\_filterByYear(List\* self, int year) {  List\* list = List\_new();  for(int i = 0; i < List\_count(self); i++) {  Website \* site = List\_get(self, i);  if(Website\_getYear(site) == year) {  List\_add(list, site);  }  }  return list;  }  List\* List\_filterByType(List\* self, char\* type) {  List\* list = List\_new();  for(int i = 0; i < List\_count(self); i++) {  Website \* site = List\_get(self, i);  if(strcmp(Website\_getType(site),type) == 0) {  List\_add(list, site);  }  }  return list;  }  char \* List\_toJson(List\* self) {  json\_t\* root = json\_array();  json\_t \* json = NULL;    for(int i = 0; i < List\_count(self); i++) {  Website \* site = List\_get(self, i);    json = json\_object();    json\_object\_set\_new(json, "name", json\_string(Website\_getName(site)));  json\_object\_set\_new(json, "id", json\_integer(Website\_getId(site)));  json\_object\_set\_new(json, "country", json\_string(Website\_getCountry(site)));  json\_object\_set\_new(json, "year", json\_integer(Website\_getYear(site)));  json\_object\_set\_new(json, "type", json\_string(Website\_getType(site)));    json\_array\_append(root, json);  }    char \*jStr = json\_dumps(root, JSON\_COMPACT);  for(int i = 0; i < List\_count(self); i++) {  json\_decref(json\_array\_get(root, i));  }  json\_decref(root);  return jStr;  } |
| storage.c |
| #include "storage.h"  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <ctype.h>  #include <stdbool.h>  #include <jansson.h>  #include "website.h"  #include <progbase/list.h>  bool Storage\_readAllText(const char \* filePath, char \* text) {  char line[100];  FILE\* fin = fopen(filePath, "r");  if (fin == NULL) return false;  if(fgets(line, 100, fin))  strcpy(text, line);  while(fgets(line, 100, fin)) {  strcat(text, line);  }  text[strlen(text)+1] = '\0';  fclose(fin);  return true;  }  List\* Storage\_readAsJson(const char\* filePath) {  char jsonStr[1000] = "";  Storage\_readAllText(filePath, jsonStr);  List\* list = List\_new();  json\_error\_t err;  json\_t \* jsonArr = json\_loads(jsonStr, 0, &err);  int index = 0;  json\_t \* value = NULL;  json\_array\_foreach(jsonArr, index, value) {  Website\* node = Website\_new();  Website\_setName(node, (char \*)json\_string\_value(json\_object\_get(value, "name")));  Website\_setId(node, json\_integer\_value(json\_object\_get(value, "id")));  Website\_setCountry(node, (char \*)json\_string\_value(json\_object\_get(value, "country")));  Website\_setYear(node, json\_integer\_value(json\_object\_get(value, "year")));  Website\_setType(node, (char \*)json\_string\_value(json\_object\_get(value, "type")));  List\_add(list, node);  }  json\_decref(jsonArr);  return list;  }  char\* Storage\_fileToJson(const char\* filePath) {  FILE\* fin = fopen(filePath, "r");  if (fin == NULL) {  puts("Can't open the file'");  return NULL;  }  fseek(fin, 0, SEEK\_END);  int size = ftell(fin);  fclose(fin);  char data[200];  Storage\_readAllText(filePath, data);  json\_t \* jobj = json\_object();  json\_object\_set(jobj, "name", json\_string(filePath));  json\_object\_set(jobj, "size", json\_integer(size));  json\_object\_set(jobj, "data", json\_string(data));  char \*jStr = json\_dumps(jobj, JSON\_COMPACT);  json\_decref(jobj);  return jStr;  }  char\* Storage\_fileContentInfo(const char\* filePath) {  char buffer[200];  Storage\_readAllText(filePath,buffer);  char \* ptr = buffer;  int sum = 0;  int count = 0;  do{  if(isdigit(\*ptr) || (\*ptr == '-' && isdigit(\*(ptr + 1)))) {  sum += strtol(ptr, (char \*\*)&ptr, 10);  count++;  } else {  ptr++;  }  } while(\*ptr != 0);  double average = sum/count;  json\_t \* jobj = json\_object();  json\_object\_set(jobj, "count", json\_integer(count));  json\_object\_set(jobj, "average", json\_real(average));  char \*jStr = json\_dumps(jobj, JSON\_COMPACT);  json\_decref(jobj);  return jStr;  } |
| website.c |
| #include "website.h"  #include <stdlib.h>  #include <string.h>  #include <jansson.h>  #define NAME\_MAXLEN 100  struct Website {  char name[NAME\_MAXLEN];  int id;  char country[40];  int year;  char type[40];  };  Website\* Website\_new(void) {  Website\* self = (Website\*)malloc(sizeof(Website));  strcpy(self->name,"");  self->id = -1;  strcpy(self->country,"");  self->year = 0;  strcpy(self->type,"");  return self;  }  Website\* Website\_setNew(char\* name, int id, char\* country, int year, char\* type) {  Website\* self = (Website\*)malloc(sizeof(Website));  strcpy(self->name,name);  self->id = id;  strcpy(self->country,country);  self->year = year;  strcpy(self->type,type);  return self;  }  void Website\_free(Website\*\* selfPtr) {  Website\* self = \*selfPtr;  free(self);  \*selfPtr = NULL;  }  char\* Website\_getName(Website \* self) {  return self->name;  }  int Website\_getId(Website \* self) {  return self->id;  }  char\* Website\_getCountry(Website \* self) {  return self->country;  }  int Website\_getYear(Website \* self) {  return self->year;  }  char\* Website\_getType(Website \* self) {  return self->type;  }  void Website\_setName(Website \* self,char\* name) {  strcpy(self->name, name);  }  void Website\_setId(Website \* self,int id) {  self->id = id;  }  void Website\_setCountry(Website \* self,char\* country) {  strcpy(self->country, country);  }  void Website\_setYear(Website \* self,int year) {  self->year = year;  }  void Website\_setType(Website \* self,char\* type) {  strcpy(self->type, type);  }  char\* Website\_toJsonString(Website \* self) {  json\_t \* jobj = json\_object();  json\_object\_set(jobj, "name", json\_string(self->name));  json\_object\_set(jobj, "id", json\_integer(self->id));  json\_object\_set(jobj, "country", json\_string(self->country));  json\_object\_set(jobj, "year", json\_integer(self->year));  json\_object\_set(jobj, "type", json\_string(self->type));  char \*jStr = json\_dumps(jobj, JSON\_COMPACT);  json\_decref(jobj);  return jStr;  } |

**Завдання 2.**

|  |
| --- |
| website.c |
| /\*\* @file  @brief Website module  \*/  #ifndef WEBSITE\_H  #define WEBSITE\_H  #include <progbase/list.h>  /\*\*  @struct Website  @brief defines a Website data type  \*/  typedef struct Website Website;  /\*\*  @brief Website constructor  \*/  Website\* Website\_new(void);  /\*\*  @brief Website constructor setter  @param name - name of website  @param id - unique id of website  @param country - origin country of website  @param year - creation year of website  @param type - functional type of website  \*/  Website\* Website\_setNew(char\* name, int id, char\* country, int year, char\* type);  /\*\*  @brief Website destructor  \*/  void Website\_free(Website\*\* self);  /\*\*  @brief Website name getter  @returns name of Website  \*/  char\* Website\_getName(Website \* self);  /\*\*  @brief Website id getter  @returns id of Website  \*/  int Website\_getId(Website \* self);  /\*\*  @brief Website country getter  @returns country of Website  \*/  char\* Website\_getCountry(Website \* self);  /\*\*  @brief Website year getter  @returns year of Website  \*/  int Website\_getYear(Website \* self);  /\*\*  @brief Website type getter  @returns type of Website  \*/  char\* Website\_getType(Website \* self);  /\*\*  @brief Website name setter  @param name - name of Website  \*/  void Website\_setName(Website \* self,char\* name);  /\*\*  @brief Website id setter  @param id - unique id of Website  \*/  void Website\_setId(Website \* self,int id);  /\*\*  @brief Website country setter  @param country - origin country of Website  \*/  void Website\_setCountry(Website \* self,char\* country);  /\*\*  @brief Website year setter  @param year - creation year of Website  \*/  void Website\_setYear(Website \* self,int year);  /\*\*  @brief Website type setter  @param type - type of Website  \*/  void Website\_setType(Website \* self,char\* type);  /\*\*  @brief Website representation in json string  @returns json representation of Website  \*/  char\* Website\_toJsonString(Website \* self);  #endif |
| List.h |
| /\*\* @file  @brief Search and filter functions for List of Websites  \*/  #ifndef LIST\_H  #define LIST\_H  #include <progbase/list.h>  /\*\*  @brief finds Website in list by its id  @param id - id of Website  @returns index of Website in list or -1 otherwise  \*/  int List\_findById(List\* self, int id);  /\*\*  @brief filters list of Websites by specified field value  @param key - name of field to be filtered by  @param value - value of field  @returns new list of appropriate Websites  \*/  List\* List\_filter(List\* self, char\* key, char\* value);  /\*\*  @brief filters list of Websites by specific name value  @param name - value of name field of Website to be filtered by  @returns new list of appropriate Websites  \*/  List\* List\_filterByName(List\* self, char\* name);  /\*\*  @brief filters list of Websites by specific id value  @param id - value of id field of Website to be filtered by  @returns new list of appropriate Websites  \*/  List\* List\_filterById(List\* self, int id);  /\*\*  @brief filters list of Websites by specific country value  @param country - value of country of Website to be filtered by  @returns new list of appropriate Websites  \*/  List\* List\_filterByCountry(List\* self, char\* country);  /\*\*  @brief filters list of Websites by specific year value  @param year - value of year field of Website to be filtered by  @returns new list of appropriate Websites  \*/  List\* List\_filterByYear(List\* self, int year);  /\*\*  @brief filters list of Websites by specific type value  @param type - type of Website to be filtered by  @returns new list of appropriate Websites  \*/  List\* List\_filterByType(List\* self, char\* type);  /\*\*  @brief list of Websites representation in json string  @returns json representation of list of Websites  \*/  char\* List\_toJson(List\* self);  #endif |
| Storage.h |
| /\*\* @file  @brief a module for work with files  \*/  #ifndef STORAGE\_H  #define STORAGE\_H  #include <stdbool.h>  #include <progbase/list.h>  #include "website.h"  /\*\*  @brief reads whole data from file  @param filePath - path to file to be read  @param text - a string to store information from file  \*/  bool Storage\_readAllText(const char \* filePath, char \* text);  /\*\*  @brief creates a list of Website elements from file  @param filePath - path to file to be read as json  @returns list of Website elements  \*/  List\* Storage\_readAsJson(const char\* filePath);  /\*\*  @brief creates a json string of list of values from file  @param filePath - path to file to be read as json  @returns json string with values from file  \*/  char\* Storage\_fileToJson(const char\* filePath);  /\*\*  @brief creates a json string with information about data in file  @param filePath - path to file to be read  @returns json string with information about data in file  \*/  char\* Storage\_fileContentInfo(const char\* filePath);  #endif |
| Request.h |
| /\*\* @file  @brief Request and Responce module for HTTP communication  \*/  #pragma once  #include <progbase/net.h>  #include <progbase/list.h>  #define MAX\_COMMAND\_LEN 100  #define BUFFER\_SIZE 1024  /\*\*  @struct Request  @brief defines a Request data type for HTTP communication  \*/  typedef struct {  char command[MAX\_COMMAND\_LEN];  } Request;  /\*\*  @struct Response  @brief defines a Response data type for HTTP communication  \*/  typedef struct {  int status;  char description[MAX\_COMMAND\_LEN];  char data[BUFFER\_SIZE];  } Response;  /\*\*  @brief parses request HTTP message  @param str - request HTTP message  @returns Request element  \*/  Request Request\_parse(const char \* str);  /\*\*  @brief processes Request and forms Response  @param request - Request element to be processed  @param responce - new Responce element  @param websites - list of Website elements  \*/  void Request\_process(Request\* request, Response\* response, List\* websites);  /\*\*  @brief forms initial Response element  \*/  void Response\_init(Response\* self);  /\*\*  @brief forms Response element with 400 Bad request status  \*/  void Response\_badRequest(Response\* responce);  /\*\*  @brief forms Response element with 404 Not found status  \*/  void Response\_notFound(Response\* responce);  /\*\*  @brief forms Response element with 200 OK status  @param data - data string to be set to a Response  \*/  void Response\_success(Response\* responce, char\* data);  /\*\*  @brief forms a string with information about server  @returns json string with information about server  \*/  char\* Response\_serverInfo(void);  /\*\*  @brief forms a response message to a client  @param response - Response element with information due request  @param message - new message to be sent to client  \*/  void Response\_toMessage(Response\* response,NetMessage \* message); |

**Приклади результатів**

Відповідь на запит про інформацію сервера.

|  |
| --- |
|  |

Відповідь на запит на улюблені веб-сайти

|  |
| --- |
|  |
| Відповідь на запит на вебсайт по значенню поля |
| Відповідь на запит на вебсайт з неіснуючим полем. |

Відповідь на запит на вебсайт по Id.

|  |
| --- |
|  |
| Відповідь на запит на вебсайт по неіснуючому Id. |

Відповідь на запит на зміст текстового файлу.

|  |
| --- |
|  |

Відповідь на запит на дані про зміст текстового файлу.

|  |
| --- |
|  |

Відповідь на запит при неіснуючій команді.

|  |
| --- |
|  |

**Висновки**

Виконавши дану лабораторну роботу було написано TCP-сервер, що слухає запити на підключення TCP клієнтів і взаємодіє з ними за допомогою протоколу **HTTP**. Сервер оброблює лише HTTP **GET** запити і повертає коректно-сформовані HTTP-відповіді. Інтерфейс сервера дозволяє звертатися до нього по заданих HTTP-шляхах. У разі неіснуючої команди або запиту на неіснуючі дані сервер повертає відповідні коди помилок.

Також було проведено коментування коду у стилі Doxygen. Були зроблені коментарі для файлів, структур та функцій у звголовочних файлах. HTML документація була сформована за допомогою утиліти Doxygen.

Крім того було налагоджено безперервну інтеграцію (Contigious integration) за допомогою Travis-Ci та репозиторію на Github. У файл README.md було додано мітку із зображенням статусу збірки та посиланнями на історію збірок

Компіляція всього коду відбувалася за допомогою компілятора gcc.