Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

По курсу: "Архитектура ЭВМ"

Студент Сукочева Алис
Группа ИУ7-53Б
Название предприятия МГТУ им. Н. Э. Баумана, каф. ИУ7
Тема Изучение запросов. Шаблонизатор. Cookie.

Студент: _____ Сукочева А.
подпись, дата Фамилия, И.О.

Преподаватель: _____ Попов А. Ю.
подпись, дата Фамилия, И. О.

Москва — 2020 г.

TASK_1.

Цель работы:

— Создать сервер;
— Работа с POST запросами;
— Работа с GET запросами;
— Работа с CSS.

Задание 1

Создать сервер. Сервер должен выдавать страницу с тремя текстовыми полями и кнопкой. В поля ввода вбивается информация о почте, фамилии и номере телефона человека. При нажатии на кнопку "Отправить"введённая информация должна отправляться с помощью POST запроса на сервер и добавляться к концу файла (в файле накапливается информация). При этом на стороне сервера должна происходить проверка: являются ли почта и телефон уникальными. Если они уникальны, то идёт добавление информации в файл. В противном случае добавление не происходит. При отправке ответа с сервера клиенту должно приходить сообщение с информацией о результате добавления (добавилось или не добавилось). Результат операции должен отображаться на странице.

Задание 2

Добавить серверу возможность отправлять клиенту ещё одну страницу. На данной странице должно быть поле ввода и кнопка. В поле ввода вводится почта человека. При нажатии на кнопку "Отправить"на сервер отправляется GET запрос. Сервер в ответ на GET запрос должен отправить информацию о человеке с данной почтой в формате JSON или сообщение об отсутствии человека с данной почтой.

Задание 3

Оформить внешний вид созданных страниц с помощью CSS. Информация со стилями CSS для каждой страницы должна храниться в отдельном файле. Стили CSS должны быть подключены к страницам.

Листинг 1 — Код программы. TASK_1. Код программы

```
1  "use strict";
2
3  // импортируем необходимые библиотеки
4  const express = require("express");
5  const fs = require("fs");
6
7  const ENCODING = "utf-8"
8  const fileName = "data/user.json";
9
10 function main() {
11     // Запускаем сервер.
```

```
12      const app = express();
13      const port = 5000;
14      app.listen(port);
15      console.log(`Server on port ${port}`);
16
17      // Отправка статических файлов.
18      const way = __dirname + "/static";
19      app.use(express.static(way));
20
21      // Получение информации.
22      // Это GET запрос он получает
23      // В url некоторые аргументы.
24      // Не имеет тела.
25      app.get("/find", function (request, response) {
26          const mail = request.query.mail;
27          let res = "Нет информации.";
28
29          // Открываем файл и парсим.
30          const objInfo = fs.readFileSync(fileName, "utf-8");
31          const infoJson = JSON.parse(objInfo);
32
33          // Проверяем наличие.
34          for (let i in infoJson) {
35              if (mail == infoJson[i].mail) {
36                  res = infoJson[i];
37                  break;
38              }
39          }
40
41          response.end(JSON.stringify({
42              result: JSON.stringify(res)
43          }));
44      });
45
46      app.get("/get_info", (_request, response) => {
47          const fileContent = fs.readFileSync("static/" + "get_info.html",
                ENCODING);
48          response.end(fileContent);
49      });
50
51
52      // body
53      // Тут получаем данные тела.
54      function loadBody(request, callback) {
55          let body = [];
56          request.on('data', (chunk) => {
57              body.push(chunk);
```

```
58          }).on('end', () => {
59              body = Buffer.concat(body).toString();
60              callback(body);
61          });
62      }
63
64      // it is post
65      app.post("/save/info", function (request, response) {
66          loadBody(request, function (body) {
67              // Файл, содержащий бд пользователей.
68
69              // Получаем данные.
70              const obj = JSON.parse(body);
71              const mail = obj["mail"];
72              const surname = obj["surname"];
73              const phone_number = obj["phone_number"];
74
75              // Открываем файл и парсим.
76              const objInfo = fs.readFileSync(fileName, "utf-8");
77              const infoJson = JSON.parse(objInfo);
78
79              // true - нужно добавить.
80              // false - уже имеется юзер.
81              let flag = true;
82              // Текст, который увидит пользователь.
83              let text = "";
84
85              // Проверяем уникальность.
86              for (let i in infoJson) {
87                  if (mail == infoJson[i].mail) {
88                      flag = false;
89                      text = "Данная почта уже имеется в системе!"
90                      break;
91                  }
92                  if (phone_number == infoJson[i].phone_number) {
93                      flag = false;
94                      text = "Данный номер уже имеется в системе!"
95                      break;
96                  }
97              }
98
99              // Если уникальный, добавляем в нашу БД.
100             // И меняем сообщение на то, что инфа добавлена.
101             if (flag) {
102                 infoJson.push({ mail, surname, phone_number })
103                 fs.writeFileSync(fileName, JSON.stringify(infoJson, null,
                        4));
```

```
                text = "Информация успешно добавлена!"
            }

            // Ответ запроса.
            response.end(JSON.stringify({
                result: text
            }));
        });
    });
}


main();
```

Листинг 2 — Код программы. TASK_1. Дополнительный скрипт

```
    "use strict";

// onload - функция, которая вызывается когда собрался HTML.
// window - это глобальной объект.
window.onload = function () {
    // Получаем (ссылку) на поля.
    const field_find_mail = document.getElementById("field-get-info");

    // Получаем кнопку, при нажатии на которую должна выдаваться информация
        .
    const btn_get_info = document.getElementById("get-info-btn");

    // ajax get
    function ajaxGet(urlString, callback) {
        let r = new XMLHttpRequest();
        r.open("GET", urlString, true);
        r.setRequestHeader("Content-Type", "text/plain;charset=UTF-8");
        r.send(null);
        r.onload = function () {
            callback(r.response);
        };
    };

    btn_get_info.onclick = function () {
        const find_mail = field_find_mail.value;

        const url = `/find?mail=${find_mail}`;

        ajaxGet(url, function (stringAnswer) {
            const objectAnswer = JSON.parse(stringAnswer);
            const result = objectAnswer.result;
```

```
            alert(result);
        });
    }
};
```

Листинг 3 — Код программы. TASK_1. Дополнительный скрипт

```
"use strict";

// onload - функция, которая вызывается когда собрался HTML.
// window - это глобальной объект.
window.onload = function () {
    // Получаем (ссылку) на поля.
    const field_mail = document.getElementById("field-mail");
    const field_surname = document.getElementById("field-surname");
    const field_phone_number =
        document.getElementById("field-phone_number");

    // Получаем кнопку, при нажатии на которую должна добавляться инфор
        мация.
    const btn_add_info = document.getElementById("add-info-btn");

    // Метка, на которой будет отображен результат добваления
        (Добавилось/Не добавилось).
    const label = document.getElementById("result-label");

    function ajaxPost(urlString, bodyString, callback) {
        let r = new XMLHttpRequest();
        r.open("POST", urlString, true);
        r.setRequestHeader("Content-Type",
            "application/json;charset=UTF-8");
        r.send(bodyString);
        r.onload = function () {
            callback(r.response);
        }
    }

    // Событие: нажали на кнопку.
    btn_add_info.onclick = function () {
        const mail = field_mail.value;
        const surname = field_surname.value;
        const phone_number = field_phone_number.value;

        // Создаем POST запрос.
        // В тело предаем mail, surname, phone_number.
        ajaxPost("/save/info", JSON.stringify({
            mail, surname, phone_number
```

```
37              }), function (answerString) {
38                  const answerObject = JSON.parse(answerString);
39                  const result = answerObject.result;
40                  label.innerHTML = result;
41              });
42          };
43      };
```

Вывод:

— Был создан сервер;
— Была реализована работа с POST запросами;
— Была реализована работа с GET запросами;
— Была реализована работа с CSS.

Пример работы:

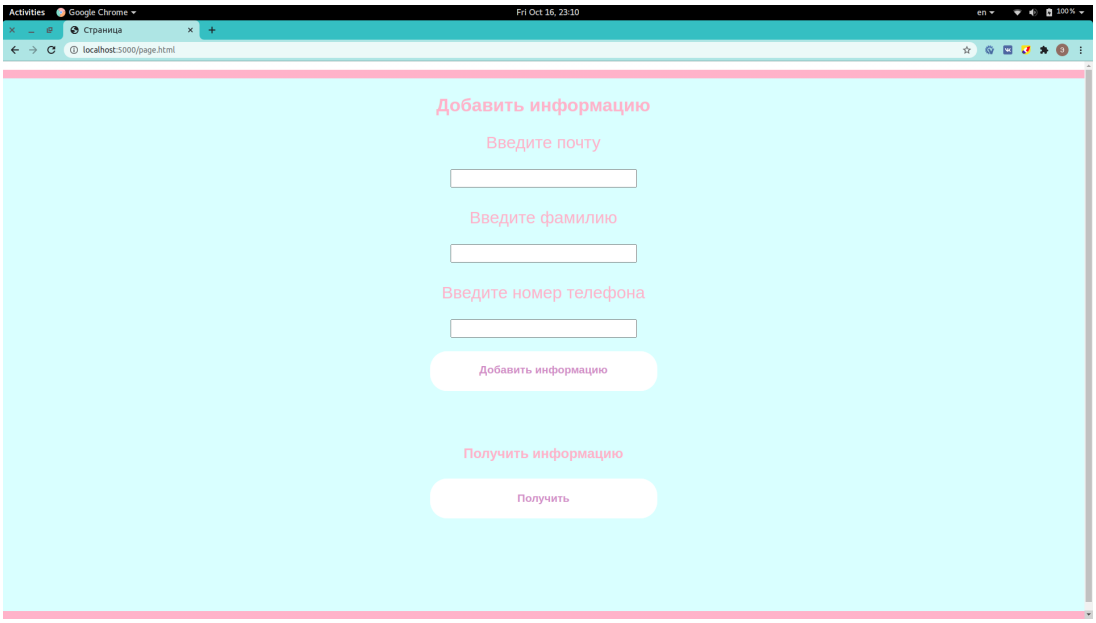

Рисунок 0.1 — Пример работы программы

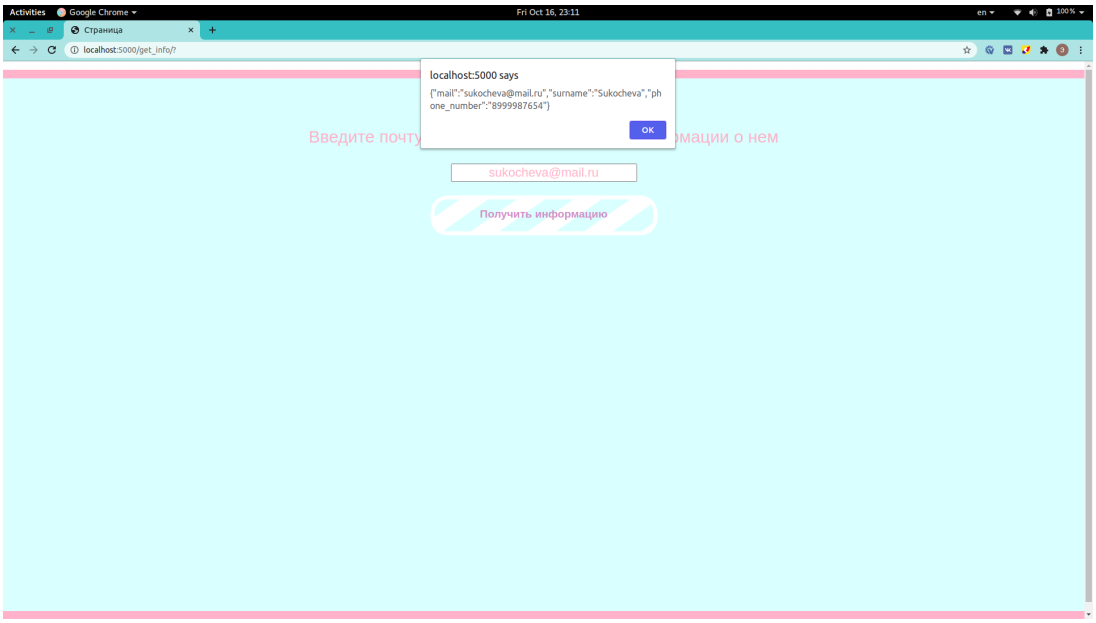

Рисунок 0.2 — Пример работы программы

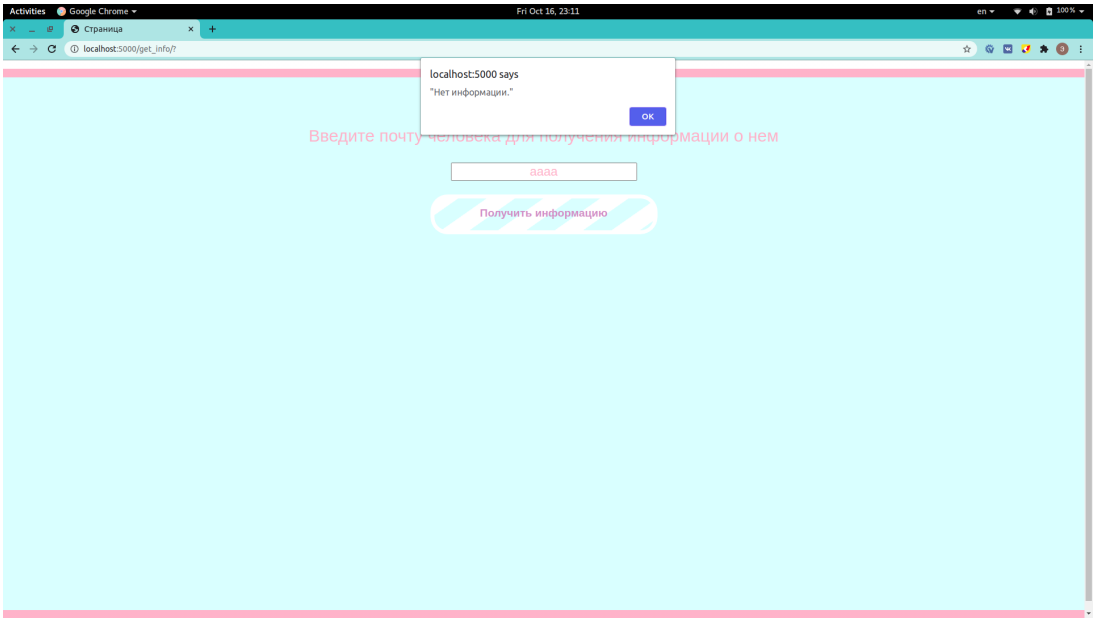

Рисунок 0.3 — Пример работы программы

TASK_2.

Цель работы:

— Создать сервер.
— Реализовать страницу с использованием шаблонизатора.
— Изучить и реализовать работу с cookie.

Задание 1

Создать сервер. В оперативной памяти на стороне сервера создать массив, в котором хранится информация о компьютерных играх (название игры, описание игры, возрастные ограничения). Создать страницу с помощью шаблонизатора. В url передаётся параметр возраст (целое число). Необходимо отображать на этой странице только те игры, у которых возрастное ограничение меньше, чем переданное в url значение.

Задание 2

Создать сервер. В оперативной памяти на стороне сервера создать массив, в котором хранится информация о пользователях (логин, пароль, хобби, возраст). На основе cookie реализовать авторизацию пользователей. Реализовать возможность для авторизованного пользователя просматривать информацию о себе.

Листинг 4 — Код программы. TASK_2. Реализация задания 1

```
1   "use strict";
2
3   // Импорт библиотек.
4   const express = require("express");
5   // Импорт библиотеки для работы с файлами.
6   const fs = require("fs");
7
8   function main() {
9       // запускаем сервер
10      const app = express();
11      const port = 5000;
12      app.listen(port);
13      console.log(`Server on port ${port}`);
14
15      // Активируем шаблонизатор.
16      app.set("view engine", "hbs");
17
18      // Выдаем страницу с массивом игр, у которых возрастное
19      // Ограничение младше, чем то, которое передано в url.
20      app.get("/page/pupils", function (request, response) {
21          // Получаем возраст,введенный пользователем из url.
22          let age = request.query.age;
23          age = parseInt(age);
```

```
24              // Если корявый url пришел, сообщаем обо этом и выходим из мето
                    да.
25              if (!age) {
26                  response.end("Age input error!");
27                  return
28              }
29
30              // Открываем файл с играми.
31              const FILE_NAME = __dirname + "/game.json";
32              const contentFile = fs.readFileSync(FILE_NAME, "utf-8");
33              const gamesArray = JSON.parse(contentFile);
34              // Создаем результирующий массив,
35              // В котором будут удовлетворяющие условию игры.
36              const resultArray = [];
37
38              // Пробегаемся по всем имеющимся играм.
39              for (let i = 0; i < gamesArray.length; i++) {
40                  // Если удовлетворяет условию
41                  // Добавляем в результурующий массив.
42                  if (gamesArray[i].age_limit < age)
43                      resultArray.push(gamesArray[i])
44              }
45              // Создаем объект, которые подставится в шаблонизатор.
46              const infoObject = {
47                  // Описание.
48                  descriptionValue: "Games list:",
49                  // Массив игр.
50                  gamesArray: resultArray
51              };
52
53              response.render("pageGames.hbs", infoObject);
54          });
55      }
56
57      main()
```

Листинг 5 — Код программы. TASK_2. Реализация задания 2

```
1       "use strict";
2
3       // импортируем библиотеки
4       const express = require("express");
5       const cookieSession = require("cookie-session");
6       const fs = require("fs");
7
8       function main() {
9           // запускаем сервер
```

```
10          const app = express();
11          const port = 5000;
12          app.listen(port);
13          console.log(`Server on port ${port}`);
14
15          // Работа с сессией.
16          app.use(cookieSession({
17              name: 'session',
18              keys: ['hhh', 'qqq', 'vvv'],
19              maxAge: 24 * 60 * 60 * 1000 * 365
20          }));
21
22          // Авторизация.
23          // Принимает два параметра:
24          // Логин и пароль.
25          app.get("/api/sign_in", function (request, response) {
26              // Если пользователь уже авторизован
27              // То сообщаем ему об этом.
28              if (request.session.login) {
29                  return response.end("You are sign in.");
30              }
31
32              // Получаем параметры запроса.
33              const login = request.query.login;
34              const password = request.query.password;
35
36              // Проверяем существование.
37              if (!login) return response.end("Login not set");
38              if (!password) return response.end("password not set");
39
40
41              // Если пользователь не авторизирован и
42              // Если такой пользователь есть в
43              // Нашем массиве, то выставляем для него cookie.
44              const FILE_NAME = "data.json";
45              const jsonString = fs.readFileSync(FILE_NAME, "utf-8");
46              const obj = JSON.parse(jsonString);
47
48              for (let i = 0; i < obj.length; i++) {
49                  if (obj[i].login === login && obj[i].password == password)
                        {
50                      request.session.login = login;
51                      request.session.password = password;
52                      return response.end("Ok!");
53                  }
54              }
55
```

```
56              response.end("Invalid Login or password.");
57          });
58
59          // Получить данные.
60          // Если пользователь авторизирован
61          // (Т.е. есть куки), то выдаем информацию о нем.
62          app.get("/api/get_info", function (request, response) {
63              // Получаем cookie пользователя.
64              let login = request.session.login;
65              let password = request.session.password;
66
67              // Контролируем существование cookie.
68              if (!login) return response.end("Not exists");
69              if (!password) return response.end("Not exists");
70
71              // Если пользователь авторизирован (cookie существуют)
72              // То выдаем информацию о нем (из файла).
73              const FILE_NAME = "data.json";
74              const jsonString = fs.readFileSync(FILE_NAME, "utf-8");
75              const obj = JSON.parse(jsonString);
76
77              for (let i = 0; i < obj.length; i++) {
78                  if (obj[i].login === login && obj[i].password === password)
                        {
79                      let answer = "Information:\nlogin:" + obj[i].login +
                            "\nAge:" + obj[i].age +
80                          "\nHobby: " + obj[i].hobby;
81                      return response.end(answer);
82                  }
83              }
84
85              response.end("No information!");
86          });
87
88          // Выход из системы.
89          app.get("/api/sign_out", function (request, response) {
90              // Удалить все cookie.
91              request.session = null;
92              response.end("You are sign out!");
93          });
94      }
95
96      main()
```

Вывод:

— Был создан сервер.

— Была реализована страница с использованием шаблонизатора.

— Была изучена и реализована работа с cookie.

Пример работы:

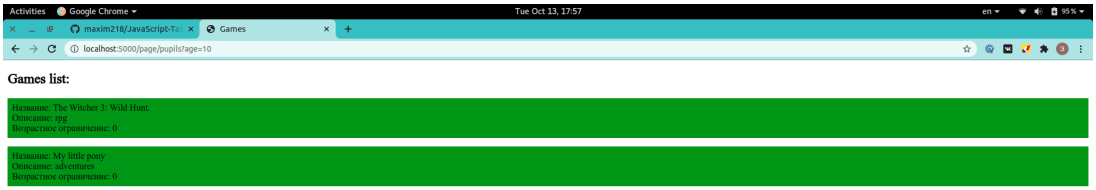

Рисунок 0.1 — Пример работы программы

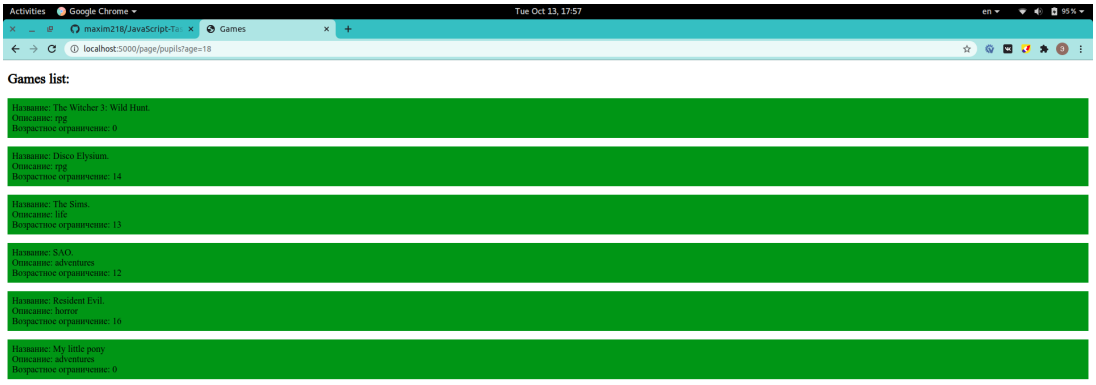

Рисунок 0.2 — Пример работы программы

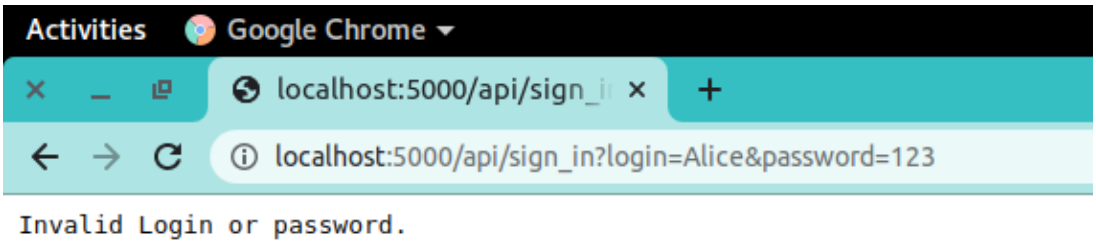

Рисунок 0.3 — Пример работы программы

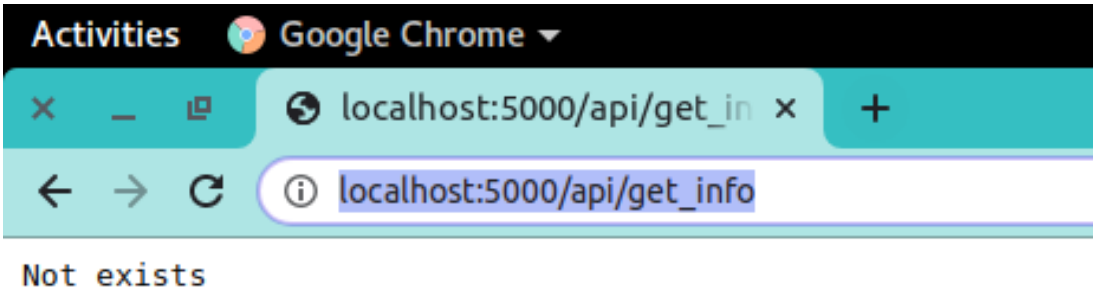

Рисунок 0.4 — Пример работы программы

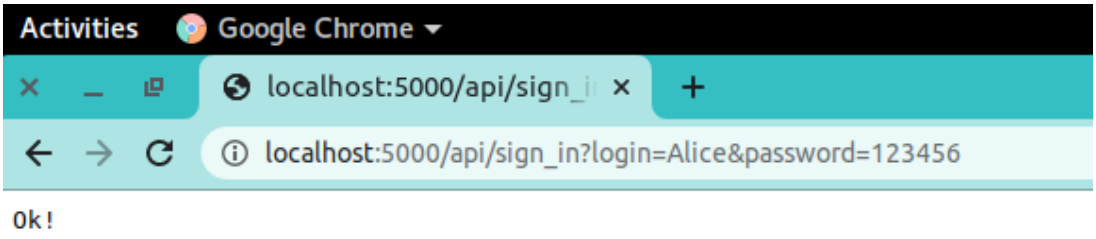

Рисунок 0.5 — Пример работы программы

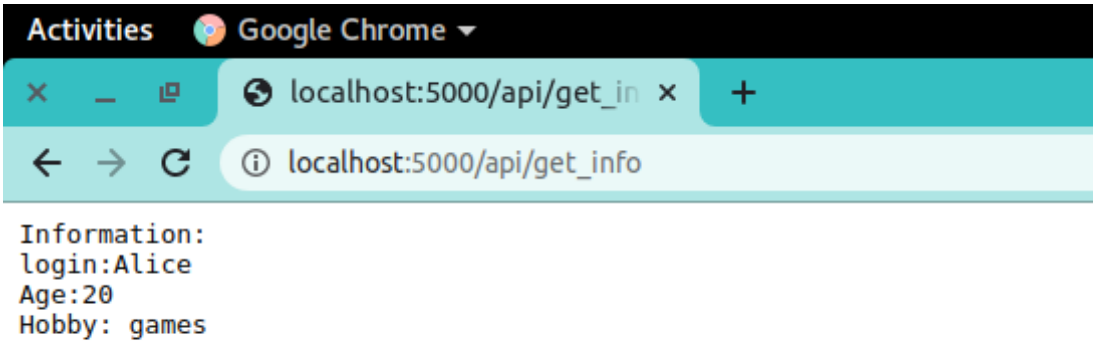

Рисунок 0.6 — Пример работы программы