Министерство сельского хозяйства Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Пермский государственный аграрно-технологический университет

имени академика Д.Н. Прянишникова»

Кафедра Информационных технологий

и программной инженерии

**ОТЧЁТ ПО КУРСОВОМУ ПРОЕКТУ**

на тему: «Разработка REST-сервиса, предоставляющего доступ к информации о json и csv файлах»

Выполнил:

студент группы ПИб-3

направления подготовки

09.03.03 Прикладная информатика

Шипулин Александр Владимирович

Проверил:

доцент кафедры ИТиПИ, к.т.н., доцент

Беляков Андрей Юрьевич

Пермь – 2023

Оглавление

[Постановка задачи на проектирование 3](#_Toc125376034)

[Анализ технологий проектирования 4](#_Toc125376035)

[Анализ бизнес-процесса 7](#_Toc125376036)

[Заключение 9](#_Toc125376037)

[Список источников 10](#_Toc125376038)

[ПРИЛОЖЕНИЕ А 11](#_Toc125376039)

[ПРИЛОЖЕНИЕ Б 12](#_Toc125376040)

# Постановка задачи на проектирование

Программа выводит документацию при вводе пользователем «**/»**.

Программа выводит содержащиеся файлы в папках при вводе расширения файла «/type».

Программа выводит содержимое файла при вводе его названия «/type/filename.type».

Программа сортирует содержимое файла при вводе «/type/filename.type/?field=direct&field=direct».

Программа должна быть кроссплатформенной, а именно запускаться на любом устройстве с любой операционной системой.

# Анализ технологий проектирования

**C#** — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумотакак язык разработки приложений для платформы Microsoft .NET Framework и .NET Core. Впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

С#‎ разрабатывался как язык программирования прикладного уровня для CLR и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов С#‎, которая отражает BCL. Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, может ли конкретная языковая особенность быть транслирована в соответствующие конструкции CLR. Так, с развитием CLR от версии 1.1 к 2.0 значительно обогатился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем (однако, эта закономерность была нарушена с выходом C# 3.0, представляющего собой расширения языка, не опирающиеся на расширения платформы .NET). CLR предоставляет С#‎, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#‎, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на VB.NET, J# и др.

**Python** — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным в том плане, что всё является объектами. Необычной особенностью языка является выделение блоков кода пробельными отступами. Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации. Сам же язык известен как интерпретируемый и используется в том числе для написания скриптов. Недостатками языка являются зачастую более низкая скорость работы и более высокое потребление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на компилируемых языках, таких как C или C++.

На основе Python было создано несколько специализированных подмножеств языка, в основном предназначенных для статической компиляции в машинный код. Некоторые из них перечислены ниже.

RPython— созданная в рамках проекта PyPy сильно ограниченная реализация Python без динамизма времени исполнения и некоторых других возможностей. Код на RPython можно компилировать во множество других языков/платформ — C, JavaScript, Lisp, .NET, LLVM. На RPython написан интерпретатор PyPy.

Pyrex — ограниченная реализация Python, но несколько меньше, чем RPython. Pyrex расширен возможностями статической типизации типами из языка C и позволяет свободно смешивать типизированный и не типизированный код. Предназначен для написания модулей расширений, компилируется в код на языке C.

Cython — расширенная версия Pyrex.

Проект Shedskin — предназначен для компиляции неявно статически типизированного кода на Python в оптимизированный код на языке C++.

**Node** или **Node.js** — программная платформа, основанная на движке V8 (компилирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API, написанный на C++, подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на Node.js и десктопные оконные приложения (при помощи NW.js, AppJS или Electron для Linux, Windows и macOS) и даже программировать микроконтроллеры (например, tessel, low.js и espruino). В основе Node.js лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

Я решил выбрать Node.js так как им можно без проблем писать код как на Linux, так и на Windows.

# Анализ бизнес-процесса

API (англ. Application Programming Interface — программный интерфейс приложения) — это набор способов и правил, по которым различные программы общаются между собой и обмениваются данными.

На практике можно выделить четыре различных способа организации программного интерфейса приложения API (Application Programming Interface) в зависимости от поддерживаемого способа обмена данными:

1. SOAP API (Simple Object Access Protocol, то есть простой протокол доступа к объектам) − информационная система доступа к объектам, в которой клиент и сервер обмениваются сообщениями посредством XML-файлов.

2. RPC API (Remote Procedure Call, то есть удалённый вызов процедур) − организация системы удаленного вызова процедур, при котором клиент запускает функцию на стороне сервера, а сервер отправляет результат обратно клиенту.

3. Websocket API (пер. сетевая розетка) – способ организации постоянного соединения с двусторонним взаимодействием клиента и сервера в выделенном процессе, при котором не только клиент может отправлять запросы к серверу, но и сервер может отправлять сообщения обратного вызова клиенту.

4. REST API — это архитектурный подход, который устанавливает ограничения для API: как они должны быть устроены и какие функции поддерживать. Это позволяет стандартизировать работу программных интерфейсов, сделать их более удобными и производительными.

Клиент-серверная модель

Сервер — программа, в которой хранятся и обрабатываются ресурсы. Сервер может располагаться на одном или нескольких компьютерах; но даже в одном компьютере может быть несколько виртуальных серверов.

Клиент — программа, которая запрашивает у сервера доступ к ресурсам. Для этого она использует API.

На рисунке 1 показана клиент-серверная модель.



Рисунок 1 - клиент-серверная модель

# Заключение

Были проанализированы несколько языков программирования (Python, C#, Node.js), после чего был выбран нужный язык программирования для создания нужного функционала.

Была написана программа, которая реализует функции отображения API, поиска файлов по указанному расширению, отображения содержимого файлов и сортировку содержимого файла по запросу пользователя.

# Список источников

1.Node.js [Электронный ресурс] Режим доступа: https://ru.wikipedia.org/wiki/Node.js#Версии

2.Python [Электронный ресурс] Режим доступа: https://ru.wikipedia.org/wiki/Python#Реализации

3.C# [Электронный ресурс] Режим доступа: https://ru.wikipedia.org/wiki/C\_Sharp

4.Как работают веб-приложения [Электронный ресурс] Режим доступа: https://habr.com/ru/post/450282/

5. How to choose a Technology Stack for Web Application Development? – GeeksforGeeks. [Электронный ресурс] Режим доступа: https://www.geeksforgeeks.org/how-to-choose-a-technology-stack-for-web-application-development/

6. Введение в REST API — RESTful веб-сервисы / Хабр. [Электронный ресурс] Режим доступа: https://habr.com/ru/post/483202/

7. REST API: что это такое и как работает. [Электронный ресурс] Режим доступа: https://skillbox.ru/media/code/rest-api-chto-eto-takoe-i-kak-rabotaet/

8. Что такое API / Хабр. [Электронный ресурс] Режим доступа: https://habr.com/ru/post/464261/

9. Что такое API?. [Электронный ресурс] Режим доступа: https://aws.amazon.com/ru/what-is/api/

ПРИЛОЖЕНИЕ А

РЕЦЕНЗИЯ

на проект

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| Рецензент: |  |

# ПРИЛОЖЕНИЕ Б

**Листинг кода – app.js**

**const** http = require("http")  
**const** { about, get\_files\_filter, cloneDeep} = require('./module')  
**let** { WorkData } = require('./library')  
**let** { WorkCSV } = require('./library')  
  
**const** server = http.createServer()  
**const** port = 3000  
  
**let** select\_case = (args, response) => {  
 **try** {  
 **switch** (args.length) {  
 **case** 1: // http://localhost:3000  
 about(response)  
 **break  
  
 case** 2: // http://localhost:3000/json или http://localhost:3000/csv  
 **let** files = get\_files\_filter('./timefall', args[1])  
 response.write(files.join('**\n**'))  
 **break  
  
 case** 3: // http://localhost:3000/json/users.json  
 **const** path = (`./timefall/${args[1]}/${args[2]}`)  
 **if** (args[1] === 'csv'){  
 **let** wc = **new** WorkCSV(path)  
 response.write(JSON.stringify(wc.\_json, **null**, 4))  
 }  
 **else** {  
 **let** wd = **new** WorkData(path)  
 response.write(JSON.stringify(wd.\_json, **null**, 4))  
 }  
 **break  
  
 case** 4: // http://localhost:3000/json/users.json/?rating=desc&name=asc  
 **const** path\_1 = (`./timefall/${args[1]}/${args[2]}`)  
 **let** temp = args[3]  
 .toString()  
 .slice(1)  
 .split('&')  
  
 **let** t = 0  
 **let** fields = ''  
 **let** directs = ''  
  
 **while** (t < temp.length){  
 fields += temp[t].split('=')[0] + " "  
 directs += temp[t].split('=')[1]+ " "  
 t++  
 }  
  
 fields = fields  
 .slice(0, -1)  
 .split(" ")  
 directs = directs  
 .slice(0, -1)  
 .split(" ")  
  
 **if** (args[1] === 'csv'){  
 **let** wc\_sort = **new** WorkCSV(path\_1)  
 wc\_sort.orderBy(fields,directs)  
 response.write(JSON.stringify(wc\_sort.\_json, **null**, 4))  
 }  
 **else**{  
 **let** wj\_sort = **new** WorkData(path\_1)  
 wj\_sort.orderBy(fields,directs)  
 response.write(JSON.stringify(wj\_sort.\_json, **null**, 4))  
 }  
 **break  
 default**:  
 response.statusCode = 404  
 response.write('Запрос ошибочный...')  
 **break** }  
 } **catch** (error) {  
 console.error(error)  
 }  
 response.end()  
}  
  
**let** callback = (request, response) => {  
 **let** args = request.url.split('/')  
 response.setHeader('Content-Type', 'text/plain; charset=utf-8')  
 **if** (args[args.length-1] === '') args.pop()  
 select\_case(args, response)  
}  
  
server.on("request", callback)  
  
server.listen(port, () => console.log(`localhost:${port}`))

**Листинг кода – library.js**

**const** {cloneDeep} = require('./module')  
**class** WorkData {  
 \_json   
 \_tmp  
 **get** json() {  
 **return this**.\_tmp  
 }  
  
 **constructor**(file\_name) {  
 **this**.\_json = require(file\_name)  
 **this**.\_tmp = cloneDeep(**this**.\_json)  
  
 }  
  
 orderBy(fields, directs) {  
 **this**.\_json.sort((a, b) => comparator(a, b, fields, directs))  
 **return this**.\_json  
 }  
  
}  
**class** WorkCSV {  
 \_json  
 \_tmp  
  
 **constructor**(file\_name, del) {  
 **this**.fs = require("fs")  
 **this**.csvjson = require("csvjson")  
 **this**.\_json = **this**.csv\_to\_json(file\_name, del)  
 .map(x => { **return** {"id": +x.id,"nameSt": x.nameSt, "sex": +x.sex, "age": +x.age, "idGr": +x.idGr }})  
 **this**.\_tmp = cloneDeep(**this**.\_json)  
 }  
  
 csv\_to\_json(nameFile, del=',') {  
 **let** textCSV = **this**.fs.readFileSync(nameFile, 'utf-8')  
 **return this**.csvjson.toObject(textCSV, { delimiter: del })  
 }  
 orderBy(fields, directs) {  
 **this**.\_json.sort((a, b) => comparator(a, b, fields, directs))  
 **return this**.\_json  
 }  
}  
module.exports = {  
 WorkData,  
 WorkCSV  
}

**Листинг кода – module.js**

**const** about = (res) => {  
 **let** about = require('./about.json');  
 res.write(JSON.stringify(about, **null**, 4));  
}  
  
**const** get\_files\_filter = (dir, ext) => {  
 **const** rd = require('fs').readdirSync;  
 **const** st = require('fs').statSync;  
 **const** path = require('path');  
 **const** rec = (dir) => {  
 **let** items = rd(dir, 'utf-8');  
 **for** (**let** item **of** items) {  
 **let** path\_item = path.join(dir, item.toString());  
 **if** (st(path\_item).isDirectory()) {  
 rec(path\_item, ext); // шаг рекурсии - если директория  
 }  
 **else** {  
 **if** (path\_item.split('.')[1] === ext)  
 name\_files.push(path\_item);  
 }  
 }  
 }  
 **let** name\_files = [];  
 rec(dir);  
 **return** name\_files;  
}  
  
comparator = (a, b, fs, ds) => {  
 **if** ((fs.length > 1) && (a[fs[0]] === b[fs[0]])) {  
 **return** comparator(a, b, fs.slice(1,), ds.slice(1,));  
 }  
 **let** d = {'asc': +1, 'desc': -1};  
 **return** d[ds[0]] \* (a[fs[0]] > b[fs[0]] ? +1 : -1);  
}  
**const** cloneDeep = (obj) => {  
 **let** result = {};  
 **for** (**let** key **in** obj) {  
 **if** (**typeof**(obj[key]) === 'object') {  
 result[key] = cloneDeep(obj[key]);  
 }  
 **else** result[key] = obj[key];  
 }  
 **return** result;  
}  
  
module.exports = {  
 about,  
 get\_files\_filter,  
 comparator,  
 cloneDeep  
}